UNIVERSITY OF SOUTHAMPTON
Faculty of Physical and Applied Sciences
Electronics and Computer Science

Evolving Ontologies with Online Learning and Forgetting Algorithms

by Heather S. Packer

Supervisors: Dr. Nicholas Gibbins and Prof. Nicholas R. Jennings
Examiner: Prof. David Robertson and Prof. Nigel Shadbolt

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

May 2011

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Heather S. Packer

Agents that require vocabularies to complete tasks can be limited by static vocabularies which cannot evolve to meet unforeseen domain tasks, or reflect its changing needs or environment. However, agents can benefit from using evolution algorithms to evolve their vocabularies, namely the ability to support new domain tasks. While an agent can capitalise on being able support more domain tasks, using existing techniques can hinder them because they do not consider the associated costs involved with evolving an agent's ontology. With this motivation, we explore the area of ontology evolution in agent systems, and focus on the reduction of the costs associated with an evolving ontology.

In more detail, we consider how an agent can reduce the costs of evolving an ontology, these include costs associated with: the acquisition of new concepts; processing new concepts; the increased memory usage from storing new concepts; and the removal of unnecessary concepts. Previous work reported in the literature has largely failed to analyse these costs in the context of evolving an agent's ontology. Against this background, we investigate and develop algorithms to enable agents to evolve their ontologies.

More specifically, we present three online evolution algorithms that enable agents to: i) augment domain related concepts, ii) use prediction to select concepts to learn, and iii) prune unnecessary concepts from their ontology, with the aim to reduce the costs associated with the acquisition, processing and storage of acquired concepts. In order to evaluate our evolution algorithms, we developed an agent framework which enables agents to use these algorithms and measure an agent's performance. Finally, our empirical evaluation shows that our algorithms are successful in reducing the costs associated with evolving an agent's ontology.

Contents

List of Figures

List of Tables

List of Algorithms

Academic Thesis: Declaration Of Authorship

I, Heather Stephanie Packer declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Either none of this work has been published before submission, or parts of this work have been published as: [please list references below]:

Signed:

. . . . . .

Date:

. . . . . .

Publications of this work to date

Packer, H. S., Gibbins, N. and Jennings, N. R. **Collaborative Learning of Ontology Fragments by Co-operating Agents.** In the Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Toronto, Canada, volume 2, pages 89-96, 1–3 September 2010.

Packer, H. S., Gibbins, N. and Jennings, N. R. **Forgetting Fragments from Evolving Ontologies.** In the Proceedings of the International Semantic Web Conference (ISWC), Shanghai, China, volume 6496(11), 7–11 November 2010.

Packer, H. S., Gibbins, N. and Jennings, N. R. **Ontology Evolution through Agent Collaboration.** In the AISB 2009 Convention, Edinburgh, Scotland, UK, 6–9 April 2009.

Packer, H. S., Payne, T., Gibbins, N. and Jennings, N. R. **Evolving Ontological Knowledge Bases through Agent Collaboration.** In the Sixth European Workshop on Multi-Agent Systems (EUMAS), Bath, UK, November 2008.

Acknowledgements

First and foremost I would like to thank Nick Jennings and Nick Gibbins for their guidance and support throughout my PhD. I am sure that without them I would not have been so successful. Thank you to David Robertson and Nigel Shadbolt who provided many interesting discussions, suggestions and new perspectives. I would also like to thank Terry Payne for the discussions that began me on this journey.

I would like to thank the people I've met in conferences, workshops and summer schools for their suggestions, criticisms and awards. Thank you to my peers for their insight and support.

Last but not least I would like to thank my family and friends who have been there for me. In particular, I would like to thank Daniel whose support was unwavering throughout this entire process.

Chapter 1

Introduction

Software agents that require vocabularies to complete tasks are dependent on the knowledge within their vocabulary, and cannot successfully complete tasks that require knowledge that is not in their vocabulary (Genesereth and Ketchpel, 1994). In order to support tasks, an agent can evolve its vocabulary so that it can complete unforeseen domain tasks, and reflect its changing needs and environment (Bailin, 2004). However, existing approaches that evolve vocabularies (van Diggelen et al., 2004; Doherty et al., 2004; Bailin and Truszkowski, 2002) do not consider optimising their techniques to reduce the associated costs involved with evolving an agent's vocabulary. Therefore, while evolving a vocabulary is beneficial to the agent by enabling it to complete more domain tasks, it can also hinder its performance. This is because agents have finite resources and limitations, such as memory and performance requirements, and large knowledge structures such as vocabularies require resources to host, manage and use. With this motivation, we explore the evolution of vocabularies while focusing on reducing the costs associated with evolving a formal knowledge base.

In order for an agent to use its vocabulary it must be expressed formally, ontologies fulfil this role because they are formal structures used to encapsulate the vocabulary of a domain; where a domain is a representation of an area of interest within the world. In the context of computer science, Gruber (2003), defines an ontology as "an explicit specification of a conceptualization". An ontology defines concepts, instances and the relationships that hold among them (Genesereth and Nilsson, 1987); these elements form the "explicit specification". In other words, it specifies the vocabulary of a domain and its constraints on the use of terms in the vocabulary. An ontology uses axioms to constrain the possible interpretations for a concept (Gruber, 2003). An axiom is an assertion in a logical form that describes a concept's application domain.

In more detail, an ontology is designed by one or more ontology engineers who are either an expert in the ontology's domain or who consult domain experts. An ontology's purpose is to be queried to support tasks. Over its lifetime it is common for an ontol-

ogy to evolve because it is unlikely that it has been designed to support all the tasks related to its domain (Klein and Fensel, 2001). Evolving an ontology so that it contains information that supports unforeseen tasks benefits agents by enabling them to complete more domain tasks and make informed decisions, thus improving the outcome of their actions. Hence, an agent may evolve an ontology through use either manually, semi-automatically, or automatically. Primarily, ontologies evolve manually or semi-automatically, by humans augmenting them with new conceptualisations or through an interface that enables agents to reuse concepts defined in other ontologies. An ontology may also evolve automatically with the use of online algorithms, where the algorithm automatically locates, retrieves, and augments the ontology with a concept when it does not contain it. Specifically, online algorithms are designed to handle problems which have incomplete information at the beginning of the problem (Borodin and El-Yaniv, 1998). Using an online algorithm to automatically evolve an ontology is desirable because not all users are experts or are even familiar with ontologies, but using such an algorithm allows them to benefit from newly augmented knowledge which supports unforeseen domain tasks. Given these benefits of using online algorithms, in this thesis we focus on the development of online ontology evolution algorithms.

In order for an online evolution algorithm to augment an ontology with new concepts, it requires ontologies to provide definitions of the required concepts. There is an abundance of ontologies (Wang et al., 2006) from which an online algorithm can automatically augment an evolving ontology. However, uncontrolled evolution may lead to an ontology that is large in size. Ontologies that are large in size require more resources to host, manage, and use compared with smaller ontologies (Wang et al., 2009). Thus, larger ontologies may critically degrade response times in time-critical systems where fast responses are required to achieve a positive outcome. Therefore, controlling the evolution of an ontology so that it contains only concepts that support domain tasks, benefits its users by supporting new domain tasks and reducing the potential associated costs of using it. This is because controlling the evolution of an ontology enables the ontology to contain only concepts and relationships that are related to its domain. Now, in order to control the evolution of an ontology, learning and forgetting algorithms can be used to: add concepts to an ontology by augmenting the ontology; and remove concepts from the ontology by pruning the ontology. In more detail, in the context of our work we use learning to describe the process of incorporating axioms from one ontology into another. However, learning can also describe processes that include inductive learning or ontology learning. Where the former describes an agent learning by example through observing percepts such as instances or the behaviour of other agents. The latter describes the creation of ontologies from documents that describe concepts and their relationships which are incorporated into an ontology (Gruber, 2003). We focus on one aspect of learning, sharing explicitly stated knowledge, because we can reuse standard vocabularies allowing entities (such as agents) to communicate using standard shared terminology.

Learning algorithms augment evolving ontologies with concepts from third-party ontologies. In some cases, only a subset of axioms from the third-party ontology is augmented into the evolving ontology. A subset of axioms may not contain all logical dependencies from an ontology, and thus reasoning on an evolving ontology is not guaranteed to be sound or complete. However, this is not always a requirement, and a "good enough" answer is appropriate where a response is required quickly, as discussed in the following example which we use to motivate evolving an ontology.

A fire brigade is tasked with extinguishing a fire, but they are unable to reach it because rubble is blocking access. The heavy rescue vehicles that are usually used to remove rubble are over an hour away. However, located on a building site there are construction vehicles that are designed to remove rubble. The fire brigade uses an ontology to decide which vehicles and equipment to use when completing tasks, and because this scenario was unforeseen then its ontology does not support reasoning about which vehicle in the building site is most appropriate for the task. If the fire brigade could augment its ontology with information about these vehicles then it could reach the fire faster, thus reducing the damage and risk to lives. Augmenting the ontology with new concepts increases the fire brigades ontology's complexity, and increases the time it takes to use the ontology. If the fire brigade could forget concepts that were no longer useful, then it could decrease the time spent using the ontology, thus also improving response times.

The above benefits of learning and forgetting information are applicable to any system that encounters new information over time, and thus can be beneficial to software agents that use ontologies (Bailin and Truszkowski, 2002; van Diggelen et al., 2004; Laera et al., 2006), and hence we use agents to show how they can benefit from using online evolution algorithms. In this context, a software agent can be described as "a self-contained problem solving system capable of autonomous, reactive, proactive, social behaviour" (Wooldridge and Jennings, 1995), and a multi-agent system contains a set of such agents. In such systems, each agent is usually designed to complete specific tasks. For example, the standard RoboCup Rescue framework contains fire brigade, ambulance and police agents which specialise in extinguishing fires, rescuing civilians, and removing rubble from the roads (Kitano and Tadokoro, 2001). In more detail, agents can use the concepts contained in their ontology to describe and reason about the entities contained in their knowledge base, and as a communication vocabulary. When an agent does not have the capability to perform a task, because it does not know which resources it requires, it can augment its vocabulary using an online algorithm so that it can complete the task. This assumes that the agent can communicate with other agents that contain the required knowledge. Since agents are designed for specific domain related tasks, their ontologies contain definitions of entities from those domains; the level of detail to which a concept is modelled is determined by the intended use of the ontology. This means that two ontologies that have intersecting concepts are likely to have different levels of detail. For example, consider two ontologies, one containing knowledge about features

on a vehicle to be sold, such as its `colour` and `registration`, and another designed for emergency rescue containing knowledge about the purpose of a vehicle, such as the scenarios where it could be used. Both ontologies contain vehicle information such as the make and model of the vehicle, but they contain different information about the vehicles. Therefore, it is unlikely that agents that have been independently designed will use the same ontology to communicate (Euzenat and Shvaiko, 2007), because the structure of their ontologies or the symbolic representation of equivalent concepts will differ. Thus, it is desirable for an agent to be selective when choosing the concepts with which to augment into its ontology.

Given this background, we focus on developing learning and forgetting algorithms that enable an agent to evolve its ontology. Thus, in the next section we describe online ontology evolution algorithms. Followed by Section 1.2 in which we describe a motivating scenario for the use of learning and forgetting algorithms. In Section 1.3, we present our research requirements in order to develop learning and forgetting algorithms. Then, in Section 1.4 we detail our research contributions. Finally, in Section 1.5 we outline the structure of the remainder of this thesis.

1.1 Online Ontology Evolution

An online algorithm can be used to evolve an ontology by expanding and contracting it, by respectively learning and forgetting concepts. When an agent learns about new concepts, it augments its ontology with additional axioms describing these concepts, and when it forgets it prunes axioms from its ontology. In this thesis, we focus on evolving an ontology online so that it can evolve through use to support additional tasks that were not foreseen when it was designed. This enables an ontology to overcome the limitation that at design time it is not possible to have perfect foresight into the tasks that an ontology is required to support. To arrive at an approach that is suitable for evolving an ontology, we consider alternative methods for ontology evolution. Specifically, ontology evolution is not the only way to achieve understanding amongst agents, for example an agent could import entire ontologies from other agents or utilise mediators for each request. However, both of these mechanisms suffer from a lack of efficiency: importing entire ontologies means that an agent acquires irrelevant as well as relevant concepts and the amount of memory used by the agent increases, thus requiring more resources and processing time which degrade the agent's performance. Similarly, by using a mediator for each task, the agent uses potentially costly resources of a mediator which include costs to use, increasing latency and communication. An agent also becomes dependent on the mediator, and thus will fail to fulfil tasks if the mediator becomes unavailable. Given this lack of efficiency, we consider two types of complementary algorithms for evolving an ontology: learning and forgetting algorithms. In the next two sections, we discuss learning and forgetting and their associated costs.

1.1.1 Concept Learning

The main benefit from incorporating new concepts into an ontology is that it can support new tasks. To date, a number of techniques have been developed for adding to an agent's ontology (more details of which are provided in Chapter 2). Broadly speaking, these techniques can be categorised into three types: i) translation, whereby a concept and its definition is translated from one agent's ontology to another; ii) learning through examples, whereby a teaching agent provides a set of training examples so that the learning agent can learn by classifying the examples; and iii) utility based techniques, where the performance of the learning agent is monitored to determine which concepts are utilised most and therefore should be incorporated into the agent's ontology, or provide the most utility to the agent. While these approaches can be used to augment an agent's ontology for the purpose of improved collaboration, they do not consider the overhead costs of acquiring and storing new concepts, nor how to reduce such costs. In order to evaluate these costs, we break down the steps involved in learning new concepts:

1. Locate the knowledge from other agents that is required to answer a task. This may involve the identification of equivalent concepts, which can be costly due to the algorithms used to translate concept names because of the differences in structure or symbolic representation (Euzenat and Shvaiko, 2007);

2. Acquire knowledge from another agent. The sharing of an ontology can be costly due to the available bandwidth and network cost. Specifically, if an agent is in an environment where network traffic is metered, or where network bandwidth is low, it is beneficial to the agent to reduce the size and frequency of messages;

3. Augment the ontology with the acquired knowledge. While learning new concepts increases an agent's vocabulary, storing concepts can adversely affect the agent's performance because of the associated time required to host, manage and use large ontologies (Seidenberg and Rector, 2006). This detriment of performance also relates to an agent's memory requirements, such as RAM and disk capacity, the complexity of reasoning with a larger ontology, and the relevance and correctness of the obtained concepts (Markovitch and Scott, 1988).

These costs can be mitigated because there is a trade off between the benefit of learning new concepts and the costs. In order for an agent to benefit from learning new concepts, we present two online learning algorithms:

1. A reactive learning algorithm, which incorporates new concepts into an ontology on-demand. This algorithm is reactive because it is used when an agent encounters a task which requires the definition of a concept;

2. A proactive learning algorithm, which incorporates concepts which are anticipated to be useful during a task, before they are actually required. This algorithm is proactive because it acts in advance by predicting and learning about concepts which will be required for a task.

We have developed a reactive learning algorithm in order to reduce the number of messages compared with other state-of-the-art approaches, and selectively augment concepts into an ontology to reduce the potential size of an ontology, compared with learning about all the concepts that it receives. We build upon our reactive algorithm by enabling it to also use prediction to select concepts to request information about and augment an ontology with concepts that have a high likelihood of use, given a specific task. This proactive learning algorithm aims to further reduce the number of required messages and increase the likelihood of augmenting an ontology with useful concepts.

1.1.2 Concept Forgetting

While there are benefits to learning new concepts, there are also limitations which include the costs associated with hosting, managing, and using large ontologies. Therefore by reducing the number of concepts in an agent's ontology, these costs can be reduced. In addition to the costs associated with large ontologies, there are also costs incurred to prune them. In order to evaluate these costs, we break down the steps involved in forgetting concepts from the ontology:

1. Decide whether forgetting is required by analysing the performance, response times, or using a utility measure. This may be costly because the performance and response times of an ontology will be evaluated every time it is used, and a utility function can be costly to calculate in terms of time and memory requirements;
2. Identify the concepts which would be most beneficial to forget, for example concepts that have not been used frequently or recently. Locating which concepts to forget requires analysing all concepts within the ontology, therefore in a large ontology such a function may be costly to calculate in terms of time and resources;
3. Remove the selected concepts from the ontology. Again this stage may also be costly in terms of time and resources because it involves removing all axioms that describe the concepts and prevents the agent from consulting the ontology while the ontology is being pruned, respectively. There is a possibility that this action can cause inconsistencies within the ontology.

Hence, we consider how to develop an online algorithm to forget concepts that balances the trade-offs of the costs associated with forgetting and the benefit from reducing an ontology's size.

1.2 Motivating Scenario

In order to motivate learning and forgetting consider the following scenario which builds upon the scenario in the beginning of our Introduction (see Chapter 1):

Assume a fire brigade uses an ontology to hold its knowledge, used to infer the best actions to take at a fire site. Specifically, its ontology contains information about fire vehicles' capabilities and is stored on a mobile device. Rubble is blocking access to the building, and the closest fire brigade vehicle that is capable of removing it is an hour away. A fast decision is critical so that damage to surrounding areas can be minimised, and it is important that the fire brigade select the most efficient vehicle for removing the rubble. A nearby construction site has suitable vehicles, but because the use of non-rescue vehicles was unforeseen during the fire brigade's ontology design, this information is not contained in its ontology. Information about these vehicles is contained in the Construction Vehicles' Manufacturer's (CVM) ontology. The CVM ontology can disseminate information in the form of an ontology fragment or by sharing its entire ontology. The fire brigade can access the CVM ontology remotely over a low-bandwidth mobile internet connection.

The fire brigade aims to learn about the vehicles in the construction site so that it can assess whether they can be of use. Thus, they send requests for fragments of the CVM ontology representing the vehicles in the building site, and required information about them such as their abilities, equipment and capacity. The fire brigade receives fragments which contain the information that they require with extraneous information such as the vehicle's date of last service and the available paint colour options for that vehicle. The fire brigade would benefit most from learning about concepts and their attributes that would help them to complete their task. Therefore, the fire brigade would ideally learn a subset of the received fragment containing the relevant information, rather than the complete fragment or even the whole ontology. Requesting fragments from the CVM ontology compared to the whole ontology is more desirable to the fire brigade because it does not require all the information and it is using a low-bandwidth mobile internet connection. The fire brigade benefits most by balancing the trade-off between learning too many concepts and too few to evolve a relatively small ontology, in terms of the possible number of concepts contained in an uncontrolled evolving ontology, which contains the information required to complete tasks. A controlled evolution results in a slower growing ontology, in terms of size, thus reducing the increase in time required to use it.

The fire brigade aims to forget about concepts from its ontology because over time it has has increased in size and hinders its response time. Thus, to improve response time there are two choices, it can: remove all concepts it has augmented; or, remove concepts that are deemed the least useful. There is a trade-off between these two choices as removing too few concepts will minimally affect the response time, and removing too many will not enable the fire brigade to make informed decisions. For example, if the agent removes

capacity information from vehicles, it will not know if a vehicle can carry large objects, whereas forgetting the paint colour options from a vehicle will not affect its decision to use that vehicle. As a fast decision is critical, the fire brigade evaluates concepts in its ontology and identifies concepts that are not frequently or recently used, to remove from its ontology. Thus, enabling the fire brigade to make faster decisions about vehicles to use, without removing recently and frequently used concepts, so that this rescue is unaffected.

We model this scenario with a multi-agent system where agents have their own ontologies, and an agent can request fragments of other ontologies which belong to other agents. In particular, we use Figure 1.1 to show the steps involved in learning and forgetting. In this figure, the environment is modelled by a *user* and provides the fire brigade agent or *task agent* with a task. In order to complete the task the agent requires more information, and attempts to learn about new concepts from the *specialist agents* in the environment. Specialist agents have their own ontologies that model a domain that intersects with the domain of the task agent, in this case one particular agent maintains the CVM ontology.

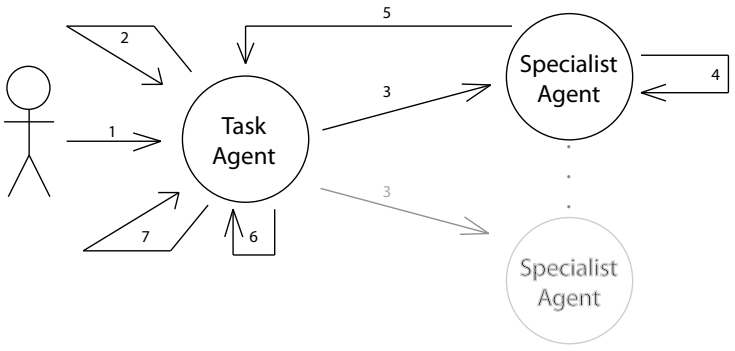


FIGURE 1.1: A sequence of messages between a user, a task agent and specialist agents.

In more detail, we specify the steps shown in Figure 1.1 between these three agents:

- In step 1 our emergency rescue agent receives a task from a user to remove rubble from a collapsed building. This task requires a vehicle that can lift a specific weight and volume of collapsed material, the agent's ontology does not contain a vehicle that can remove the rubble. Therefore it desires to learn about a vehicle that can;
- Given that the agent has a model that represents previous outcomes of the knowledge required to complete tasks and the probability of that knowledge being required, it considers learning about the concepts that have the highest probability of use. In this case, the agent desires to learn about `truckMountedForklift` and `truckForklift` indicated by step 2;
- In step 3, a request is sent to one or more specialist agents in the environment for knowledge about a vehicle that can remove the rubble, and for definitions of

`truckMountedForklift` and `truckForklift`;

- One of the specialist agents finds that its ontology contains the concept `truckMountedForklift` and that this vehicle has the required features, and sends a fragment representing a `truckMountedForklift`. This fragment contains the concepts `vehicle`, `liftingCapacity`, `building`, `reachTruck`, `handPalletTruck`, and `truckMountedForklift`, as shown in step 4;
- In step 5, the task agent receives the fragments from the specialist agents;
- The task agent then selects which concepts to learn. In order to select these concepts, the agent prioritises concepts that relate to its domain and are predicted to be useful (shown in step 6);
- The task agent determines whether it contains concepts that have not been utilised recently or frequently, such as the paint colour options, and dates of last service, thus it forgets these concepts from its ontology, in order to reduce the amount of unnecessary concepts in its ontology, (shown in step 7);
- The agent now has the knowledge about the resources necessary to remove the rubble. The agent then aims to locate an instance of the concept `truckMountedForklift` to complete the task.

In steps 6 and 7 in particular, a task agent evolves its ontology, by processing the fragments it has received and augmenting its ontology, and removing unnecessary concepts from its ontology. The above process enables the task agent to find which vehicle from the nearby construction site that is best suited to remove the building rubble.

1.3 Research Requirements

This thesis focuses on the design of two types of algorithms, which learn and forget concepts from an agent's ontology. In order to identify our requirements, we first introduce the assumptions we make about our environment and ontologies, as follows:

1. The set of ontologies in an environment do not contain any inconsistent axioms. Therefore our approach does not handle inconsistencies.
2. Agents communicate using sequential messages, and their decisions and actions are dependent on responses. Therefore, in our case we assume that minimising the number of messages is beneficial, while in other scenarios this may not be the case.
3. Messages sent within the environment are not affected by network latency or message loss, and therefore this is not a major consideration in this work.

4. Our learning and forgetting algorithms can be used with OWL Lite and OWL DL ontologies. Our evolution algorithms will modify the concept subsumption hierarchy of an ontology and do not specifically modify any other semantic components. Thus, there will be no prerequisite of expressive features that an ontology should contain before our algorithms are applied. However, ontology evolution algorithms have not typically been performed with OWL Full ontologies, because they are not fully decidable. Their lack of decidability means that full verification cannot be performed, and thus checking for inconsistencies is not possible.

We use these assumptions to focus the scope of our work, and we require the following to develop efficient learning and forgetting algorithms: a suitable framework to evaluate the ontology evolution algorithms; online learning and forgetting algorithms; and a demonstration of the effectiveness of using the algorithms. Next, we discuss each of these requirements in detail which serve as requirements for the literature survey in Chapter 2.

1.3.1 Requirement 1: Evaluation Framework for Evolving Ontologies

We require a framework that enables agents to complete tasks that require domain knowledge, and to evolve their ontologies when they do not contain the knowledge to complete a task. Our framework must also support the evaluation of agents' performance so that we can compare our approach to other state-of-the-art approaches. More specifically, the design requirements are as follows:

1. **Task Environment:** We require an environment that simulates tasks for agents to perform, and the agents' actions directly affect the outcome of the task;

2. **Agent Model:** We require a suitable model for evolving agents' ontologies. They use their own ontologies to decide on their actions when encountering tasks, and enable them to evolve their ontologies during their lifetime;

3. **Domain Ontologies contained within the Environment:** We require that task agents can learn from domain ontologies in the environment. Domain ontologies should provide task agents with fragments of ontologies to enable them to complete tasks that they could not before;

4. **Performance Evaluation Measures:** We also require an evaluation measure for comparing the performance of the agents using learning and forgetting algorithms, in terms of the outcome of the task environment.

1.3.2 Requirement 2: Online Evolution Algorithms

We require algorithms to evolve agents' ontologies so that they can complete unforeseen domain tasks. These algorithms should allow agents to learn additional knowledge when it is required to complete a task, and to forget knowledge when the response time from the ontology hinders their performance. As an agent's ontology evolves it should remain consistent: an inconsistent ontology contains axioms that describe a concept which contradict. This is required because an agent uses its ontology to complete tasks about specific concepts and if it contains contradictions about a concept then it may provide incorrect information about instances. We focus on reducing the costs associated with learning and forgetting algorithms so that they perform efficiently with low resource overheads. This should enable agents which use these approaches to perform better than other state-of-the-art approaches. In order to evolve an ontology efficiency, we require the following types of algorithms:

(a) **An algorithm for incorporating new relevant knowledge into an agent's ontology (learning).**

We require that agents can augment their ontologies so that it can fulfil domain specific tasks which it could not before. To this end, we propose an algorithm that selects and augments concepts into an agent's ontology. In particular, this algorithm should reduce the cost of acquiring regularly required knowledge because each time an agent acquires a concept there are overhead costs;

(b) **An algorithm for predicting which knowledge will be required in a scenario.**

We require that agents can predict which concepts will be required in the future so that they can select concepts to augment into their ontology based on these predictions, thus reducing the frequency of acquiring concepts. This objective aims to enable agents to augment their ontologies with concepts that have a high probability of occurring in their environment, and avoiding augmenting concepts that are unlikely to be used. The focus of this algorithm is to enable an agent to build a domain focused ontology;

(c) **An algorithm for removing irrelevant knowledge from an agent's ontology (forgetting).**

We also require that an agent can reduce the size and potential complexity of its ontology, and therefore reduce the cost of hosting, managing and using it. With this objective, we aim to develop an algorithm that can select which concepts are the least useful, and remove them from its ontology. The focus of this technique is to reduce the agent's ontology in order to prevent it from expanding without limit.

1.3.3 Requirement 3: Empirical Evaluation

In order to evaluate the effectiveness of evolving ontologies with learning and forgetting algorithms, we require success measures. Such success measures are required to highlight the variances between the outcomes of the different learning and forgetting algorithms. In detail, we require the following aspects to be assessed:

1. **Framework Performance:** We require the ability to evaluate the overall performance of the agents using different evaluation algorithms, in terms of the outcome of the task environment. This enables us to analyse which algorithm is the most beneficial to the agents;
2. **Algorithm Success Measures:** We require the ability to breakdown the success of agents' actions, so that we can analyse the rationale behind their overall success within the framework. We identify two types of measures:
   (a) **General Measures:** These types of measures can be used to evaluate all ontology evolution algorithms within our framework. They can be used to directly compare agents' ontology evolution algorithms, because they are common to all evaluated algorithms;
   (b) **Algorithm-Specific Measure:** These types of measures are specific to types of algorithms and cannot be used to directly compare ontology evaluation approaches. Each evolution algorithm can be assessed against different measures specific to the nature of the algorithm. For example, learning and forgetting algorithms can be assessed against the number of times they learn and forget concepts, respectively.
3. **Ontology Complexity:** We also require the ability to analyse the complexity of an ontology, because it affects the resources required to use it. The complexity of an ontology is affected by its size and structure.

1.4 Research Contributions

In order to achieve the goals listed in Section 1.3, we designed an online algorithm to evolve an ontology by both learning and forgetting fragments of concepts based on the tasks the ontology is required to support. We situate our learning and forgetting algorithms in an adaptation of the standard RoboCup Rescue (RCR) framework (Kitano and Tadokoro, 2001). We use RCR because it provides agents with an environment that generates tasks and enables the comparison of the performance of other state-of-the-art approaches. Our extension also provides agents with ontologies to support the completion of tasks within the environment. Using our learning and forgetting algorithms,

agents can benefit from reducing the time spent on retrieving required information and forgetting concepts, so that it can spend more time performing its actions. Our learning algorithm focuses on reducing the overhead of acquiring knowledge by augmenting an agent's ontology with a desired concept and a selection of its neighbouring concepts. We build upon our learning algorithm by enabling agents to predict which concepts have a high probability of occurrence to complete tasks given the information that was previously required to complete the same tasks. Our forgetting algorithm focuses on reducing the size of the ontology, thus reducing loading times and the ontology complexity.

By achieving these objectives, we advance the state-of-the-art in the domain of agent learning and ontology evolution, in the following ways:

1. Develop the first fully automated technique to enable an agent to learn and forget concepts from its ontology (this is in line with the requirement presented in Section 1.3.2 and is built on the framework proposed in Requirement 1, in Section 1.3.1). In more detail, we develop three algorithms that:

   (a) **Learn reactively** by selecting concepts to learn from information received from other agents. The structure of the received information is compared to the agent's ontology, and relationships between concepts are analysed, in order to select related concepts to augment into the agent's ontology;

   (b) **Learn proactively** by requesting the definition of concepts that have a high likelihood of future use given a specific task, and selecting concepts to augment into an agent's ontology based on their likelihood of use;

   (c) **Forget** by pruning a selection of concepts from an ontology, which are the least frequently and recently used once an agent determines that its performance is hindered by using its ontology.

2. Demonstrate for the first time that ontology evolution, in terms of an agent learning and forgetting concepts from its ontology, is an effective way for an agent to complete tasks in a specific domain (this is in line with Requirement 2, presented in Section 1.3.2 and is supported by the requirement presented in Section 1.3.3). In more detail, we show that evolving an ontology is effective using algorithms which:

   (a) **Learn reactively** by augmenting an agent's ontology with a select set of concepts. Thus, balancing the trade-off between an ontology's potential size and complexity, and the costs associated with acquiring new knowledge;

   (b) **Learn proactively** by augmenting an agent's ontology with concepts that are predicted to be useful. This algorithm is designed to also balance the trade-off between an ontology's potential size and complexity, and costs associated with acquiring new knowledge. However, this algorithm builds upon

the reactive learning algorithm and further reduces an ontology's potential size and complexity;

(c) **Forget** concepts from an ontology by removing concepts that are neither frequently or recently used, thus reducing the ontology's size and potential complexity. Therefore the time and resources an agent requires to use its ontology are reduced, which increases the agent's productivity.

3. Provide the first framework that enables the research community to develop ontology evolution strategies that facilitate the reuse of standard vocabularies by agents in order to complete unforeseen domain tasks with limited resources. It extends RCR by providing additional variables about the rescue targets. These variables enable agents to deliberate about their actions in a more realistic environment than RCR, and thus these strategies are more easily transferable to the real world. This extension provides a general framework which furthers RCRs contribution to enable the development of strategies to allocate and co-ordinate resources. Within RCR there are standard success measures, which include the RCR score and the *score vector*, thus supporting the requirement presented in Section 1.3.1.

These contributions have led to the following peer reviewed papers:

1. Heather S. Packer, Terry Payne, Nicholas Gibbins and Nicholas R. Jennings. **Evolving Ontological Knowledge Bases through Agent Collaboration**. In the Proceedings of the Sixth European Workshop on Multi-Agent Systems (EUMAS2008), Bath, UK, 2008.

   This paper introduces a preliminary version of our reactive online learning algorithm for agents that collaborate by sharing their vocabulary (Contribution 2a).

2. Heather S. Packer, Nicholas Gibbins and Nicholas R. Jennings. **Ontology Evolution through Agent Collaboration**. In the Proceedings of the AISB Convention (AISB2009), Edinburgh, Scotland, UK, 2009.

   This paper extends Packer et al. (2008) by describing our reactive learning algorithm and provides new results (Contributions 2a and 3a).

3. Heather S. Packer, Nicholas Gibbins and Nicholas R. Jennings. **Collaborative Learning of Ontology Fragments by Co-operating Agents.** In the Proceedings of the IEEE / WI-IAT / ACM International Conference (WI-IAT2010), Toronto, Canada, 2010.

   This paper presents, in detail, our reactive online learning algorithm alongside, for the first time, our RoboCup OWLRescue framework. Using the RoboCup OWLRescue context we evaluate our reactive learning algorithm (Contributions 1, 2a and 3a). This paper was awarded best paper.

4. Heather S. Packer, Nicholas Gibbins and Nicholas R. Jennings. **Forgetting Fragments from Evolving Ontologies.** In the Proceedings of the International Semantic Web Conference (ISWC2010), Shanghai, China, 2010.

   This paper presents, in detail, our forgetting algorithm and our RoboCup OWLRescue framework. Using the RoboCup OWLRescue context we evaluate our forgetting algorithm (Contributions 1, 2c and 3c). This paper was awarded best student paper.

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

- In Chapter 2, we review the existing literature in the areas of interest, which are: ontology evolution; augmenting and diminishing agents' ontologies; how we can improve learning using prediction; and how to evaluate learning and forgetting algorithms and agents using them;
- In Chapter 3, we present our investigation into how we can evaluate ontology learning and forgetting algorithms. Particularly, we focus on investigating methods to evaluate the complexity of an ontology using measures discussed in Chapter 2. Following that, we focus on the implementation of an evaluation multi-agent framework, RoboCup OWLRescue, to compare the performance of agents using different ontology learning and forgetting algorithms;
- This is then followed by Chapter 4, in which we present our reactive ontology learning algorithm, its implementation, and the results from our empirical evaluation using the Robocup OWLRescue simulation. The reactive algorithm focuses on learning fragments of ontologies based on tasks an agent encounters in RoboCup OWLRescue;
- In Chapter 5, we present our proactive ontology learning algorithm, its implementation, and the results from our empirical evaluation using the RoboCup OWLRescue simulation. The proactive algorithm focuses on learning fragments of ontologies based on the outcome of previous tasks encountered in other RoboCup OWLRescue simulations using a Markov model;
- Chapter 6, we present our ontology forgetting algorithm, its implementation, and the results from our empirical evaluation using the RoboCup OWLRescue simulation. The forgetting algorithm focuses on removing fragments of concepts that are not frequently or recently used, and are cheap to acquire in terms of time;
- We conclude in Chapter 7 with a summary of our research and an outlook on future work.

- Finally, our Appendices: Appendix A provides additional results from our reactive learning algorithm evaluation; Appendix B provides additional results from our proactive learning algorithm evaluation; Appendix C provides additional results from our forgetting algorithm evaluation; Appendix D provides an extract from an environment ontology; Appendix E provides example fragments from the environment ontologies and Appendix F is a glossary of all the terms used in the thesis.

Chapter 2

Literature Review

This chapter presents an overview of ontologies and ontology evolution, current algorithms that enable agents to evolve their ontologies, and probability models that can be used to predict which knowledge will be required for future domain tasks. We also discuss the use of RoboCup Rescue (RCR) as a framework to situate the evaluation for our algorithms, and consider measures that can be used in an empirical evaluation. In more detail, RCR is a standard search and rescue environment for the development of strategies for task allocation and agent co-ordination, and we analyse its capability for enabling the evaluation of learning and forgetting algorithms, thus supporting our second and third requirements (presented in Sections 1.3.2 and 1.3.3).

In more detail in Section 2.1, we outline what an ontology is, detail ontology languages, and discuss how inference can used to determine implicit entailments over an ontology. Section 2.2 discusses methodologies and issues encountered while evolving ontologies. Next, in Section 2.3, we detail approaches for agents to expand their ontologies by augmenting them with additional concepts and their descriptions. In Section 2.4, we describe techniques that can be used by agents to predict knowledge that has a high likelihood of future use. Following that, in Section 2.5, we present techniques that can be used to evaluate ontology evolution algorithms, we consider associated costs with evolving an ontology, ontology complexity measures, and the RCR framework. Finally, in Section 2.6 we summarise the chapter by highlighting the work that provides a foundation for our study and also discuss issues that will need to be addressed in order to meet our requirements specified in Section 1.3.

2.1 Ontologies, Ontology Languages and Reasoners

We aim to develop an algorithm that evolves an agent's ontology in order to support tasks that were not foreseen at design time (see Requirement 2 in Section 1.3.2). Agents

benefit from using an algorithm to evolve their ontology because they can answer queries that they could not before. In this section, we introduce ontologies, their components and uses. This background on ontologies provides the foundation on which we will develop our algorithm, and the terminology used to describe it. In more detail, we survey the current literature on ontologies, ontology languages, and ontology reasoning.

2.1.1 Ontologies

An ontology encapsulates the vocabulary of a domain, where a domain is a representation of an area of interest within the world. For example, in the domain of fire and rescue the following subset of 'things' that fall within that domain are: `fire_engine`, `K422MM`[1], `fire`, `chemical_fire`, `water`, and `dry_agent`. In order to define the semantics of 'things' in an ontology, they are classified as either concepts (also known as classes), individual objects or relationships. The former is a description of a specific type of entity, for example the concept `fire_engine` is not an instance of a specific fire engine but rather relates to a generalisation of a fire engine. The latter is a resource that belongs to a specific class (concept), in our fire rescue domain example, the fire engine with license plate `K422MM` is of type `fire_engine`. Concepts and individuals are colloquially defined as being contained in an Assertional-Box (A-Box) and a Terminological Box (T-Box), respectively (Baader, 2003) (despite the fact that they are not actually segregated in an ontology). An ontology contains a set of axioms, which are self-contained statements of fact. For example, an ontology about fire engines might contain the axioms:

- `fire_engine` coloured `red`;
- `fire_engine` equipped with `fire_hose`.

In order to describe a concept, an ontology imposes restrictions on its use. Restrictions are important because they prevent the concept being used nonsensically. For example, the following two statements are nonsense and without restrictions may erroneously occur when defining a domain:

- `fire_engine` is extinguished by `K422MM`;
- `chemical` is extinguished by `chemical_fire`.

The first statement is nonsensical because it specifies that a particular `fire_engine` extinguishes an instance of a `fire_engine`. Building upon our example, we require that a `fire` will be extinguished by `fire_engine`s only, thus restricting the range of the property. The second statement is nonsensical because it specifies that a `chemical`

[1] The registration number of a fire engine.

is extinguished by a `chemical_fire`. Building upon our example, only `fire`s can be extinguished. This imposes a restriction on the objects to which the property can be applied, thus restricting the domain of the property.

Ontologies also establish relationships between concepts, such as subsumption (also known as a subclass relationship). For example, the classes `fire` and `chemical_fire` are related to each other: a `chemical_fire` is a subclass of `fire`, and equivalently, that `fire` is a superclass of `chemical_fire`. Subsumption defines a hierarchy of classes, where A is a subclass of B if every instance of A is also an instance of B. In addition to subsumption relationships, properties can also be organised in a hierarchy with sub-property relationships. For example, the "is extinguished by" relationship is a sub-property of "acted on by". If a `fire` f is extinguished by a `fire_engine` e, then f is acted on by e. In general, P is a sub-property of Q if two objects are related by Q whenever they are related by P.

More succinctly, an ontology is "an explicit specification of a conceptualization" (Gruber, 1995). An ontology defines objects, instances and the relationships that hold among them; these elements form the "explicit specification". Together these elements also represent an interest domain, a view of a world or "conceptualization", which was built specifically for some purpose. The term "ontology" originates from philosophy, where Ontology is the study of existence (Grossmann, 1992); in computer science, what 'exists' in a world can be represented in an ontology (Gruber, 1995). In this thesis, we refer to an ontology as "a formal specification of a shared conceptualization" (Borst, 1997) which is a refinement of Gruber's (1995) definition. This definition expands Gruber's by adding that an ontology is a machine-readable domain model that captures consensual group knowledge. In this work, we use Borst's definition because we require that software agents are able to 'read' an ontology, and our algorithm is required to augment shared conceptualisations (see Requirement 2a in Section 1.3). In order for an ontology to be readable by machines, and therefore to be used by agents, it needs a formal representation (Studer et al., 1998). A formal representation enables our framework to share knowledge with task agents (da Silva et al., 1998).

To this end, there are currently several languages with different levels of formality. We consider these different levels of formality so that we can analyse which formalities are applicable to our ontology evolution algorithms, these include:

1. **Informal** ontologies, which are expressed in natural language and are not machine readable. For example, 'potassium is suppressed with a dry agent';
2. **Semi-Informal** ontologies, which are expressed in a structured form of natural language. For example, 'potassium FIRESUPPRESSANT dry agent';
3. **Semi-Formal** ontologies, which are expressed in a formally defined language, but which only model partial knowledge about a domain (Gruber, 2003). In other

words, such an ontology does not aim to represent a complete domain, but only expresses concepts that are required in a specific task. For example, the concept `Potassium` is semi-formally described with the Web Ontology Language (OWL):

```
<owl:Class rdf:about="Potassium">
            <rdfs:label>Potassium</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:about="fireSuppressant"
      rdfs:label="Fire Suppressant">
            <rdfs:range rdf:resource="Potassium" />
            <rdfs:domain rdf:resource="Dry_Agent" />
</owl:ObjectProperty>
```

4. **Formal** ontologies, which define complete knowledge of all terms within a domain, formally with established theorems and proofs. For example, the concept `Potassium` is described formally with the Web Ontology Language (OWL), where we assume that the concept `Potassium` has only two properties, and thus define complete knowledge:

```
<owl:Class rdf:about="Potassium">
            <rdfs:label>Potassium</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:about="fireSuppressant"
      rdfs:label="Fire Suppressant">
            <rdfs:range rdf:resource="Potassium" />
            <rdfs:domain rdf:resource="Dry_Agent" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="chemicalSymbol"
      rdfs:label="Chemical Symbol">
            <rdfs:range rdf:resource="Potassium" />
            <rdfs:domain rdf:resource="K" />
</owl:ObjectProperty>
```

   Formal ontologies aim to express a complete domain with complete definitions of concepts, meaning that all of the properties and relationships are defined for each concept. This type of ontology usually has thousands of concepts due to the number of concepts used to express a domain. Such an ontology also requires a greater number of types of axioms than an ontology that represents a smaller part of a domain. The GENE Ontology is an example of a formal ontology that has been designed to represent the entire domain of genetics (Ashburner et al., 2000). In practice, the distinction between formal and semi-formal ontologies is largely arbitrary, depending on the domain and application of the ontology.

Although semi-formal ontologies are limited in the information they represent compared with formal ontologies, they are noted by Sheth and Ramakrishnan (2003) as being more practical and useful, and that formal ontologies have "yielded little value in real-world applications." Both semi-formal and formal ontologies are designed to meet functional objectives (Gruber, 2005), but the nature of formal ontologies, which require the complete definition of all concepts, can focus them on describing a taxonomy, unlike semi-formal ontologies which instead focus on supporting specific tasks. The concepts within a semi-formal ontology are described to support a functional objective, however, these functional objectives may change over time and thus an ontology may be altered in order to support them. Given this, we propose that semi-formal ontologies are the most suitable application for our proposed online evolution algorithms because they model a domain incompletely and therefore there is more scope for them to evolve than formal ontologies. However, there is no limit to the type of ontology that our proposed algorithms can be applied to, and we focus on developing algorithms which can be applied to any ontology.

Both semi-formal and formal ontologies use constructs to express the relationships and restrictions contained within them. The range of constructs used in an ontology can vary, and the expressivity of an ontology depends on the types of constructs used within it. Levels of expressivity in an ontologies differ depending on its use because not all 'conceptualisations' require the same expressive features (described in Table 2.1). In order for an ontology to express such features, logic is often used to express their classes, instances and their relationships (Sowa, 2000). The family of logic used to represent knowledge are called Description Logics (DL) (Baader, 2003) and they model concepts, roles and individuals, and their relationships. There are many variants of DLs which are categorised with an informal naming convention describing the constructs of operators used in a DL knowledge representation language. Table 2.1 details the different operators which can be used to express relationships and roles within a DL. The different levels can affect how concepts are described. For example, an OWL Lite ontology can describe that a fire engine operator can drive fire engines, and an OWL DL ontology can describe that only one fire engine operator can drive fire engines. The OWL DL ontology contains a cardinality restriction, while the OWL Lite ontology does not because its expressivity does not allow it. Therefore the OWL DL ontology contains more axioms, and thus there will be different operations to remove concepts from an ontology with different levels of expressivity (we detail different forgetting techniques in Section 2.2.2). This therefore will be a consideration when developing a forgetting algorithm (see Requirement 2c 1.3.2).

In the next section, we describe ontology languages which provide a standard for publishing ontologies and are used to formalise the concepts, roles, individuals, and relationships.

Notation	Feature
$\mathcal{F}$	Functional properties.
$\mathcal{E}$	Full existential qualification.
$\mathcal{U}$	Concept union.
$\mathcal{C}$	Negation.
$\mathcal{S}$	An abbreviation for $\mathcal{ALC}^a$ with transitive properties.
$\mathcal{H}$	Role hierarchy.
$\mathcal{R}$	Limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness.
$\mathcal{O}$	Nominals, which are enumerated classes.
$\mathcal{I}$	Inverse properties.
$\mathcal{N}$	Cardinality restrictions.
$\mathcal{Q}$	Qualified cardinality restrictions.
$(\mathcal{D})$	Use of datatype properties, data values or data types.

[a]The notation $\mathcal{ALC}$ does not fit in with this naming convention. It denotes an attributive language that uses atomic negation, concept intersection, universal restrictions, limited existential quantification and a complement of any concept.

TABLE 2.1: The DL feature notation, taken from (Baader, 2003)

2.1.2 Ontology Languages

There are various ontology languages used to express ontologies. The development of these languages began in the early 1990's, when research explored how knowledge representation from the field of artificial intelligence could be utilised on the World Wide Web. This research resulted in languages based in Extensible Markup Language (XML) (Bosak and Bray, 1999), which is a set of rules for encoding documents in machine-readable form. There are also other ontology languages that have been developed for specific systems (such as the CycL (Lenat et al., 1990), Loom (MacGregor, 1991), and SHOE (Heflin et al., 1998)). In this section however, we focus on describing Web ontology languages because they provide a standard language for expressing ontologies. This is relevant to the work presented in this thesis because we require a framework in which agents have their own ontologies that they can augment and prune, and therefore in this section, we analyse which ontology language to use (see Requirement 1 in Section 1.3.1).

In order to analyse which ontology language to use, we first consider the versions of the Web Ontology Language (OWL). OWL is actually a family of knowledge representation languages of which there are three subspecies: OWL Lite, OWL DL and OWL Full. Similar to DL (see Section 2.1.1), each subspecies has different characteristics, which define different levels of expressivity (McGuinness and van Harmelen, 2004). For example, only certain subspecies of OWL can express negation, cardinality, and transi-

tive properties. OWL is an extension of the Resource Description Framework Schema (RDF(S)) language which enables the description of classes and relationships. In more detail, we describe the features of RDF(S) and the three subspecies of OWL:

RDF(S) is intended for users primarily needing a classification hierarchy with typing of properties and meta-modelling facilities (McGuinness and van Harmelen, 2004). This language can be used to define classes and instances of classes, binary properties between class instances, class and property hierarchies, and types for properties (domain and range restrictions);

OWL Lite was designed for creating classification hierarchies (McGuinness and van Harmelen, 2004). It differs from the expressivity of RDF(S) because it restricts the use of its constructs. These restrictions do not allow classes to be used as individuals, and the language constructors cannot be applied to the language itself (Horrocks et al., 2003). It has a DL expressivity of $\mathcal{SHIF}(\mathcal{D})$ (see Table 2.1), where: $\mathcal{S}$ denotes that this language can express negation of any concept in an ontology, the intersection of concepts where the conjunction of two concepts can describe a single concept ($A \sqcup B$), universal restrictions where the restriction applies to all instances of a concept, and limited existential quantification where the restriction applies to some instances of a concept; $\mathcal{H}$ denotes that the language can express role hierarchy where properties have a hierarchy using subproperties; $\mathcal{I}$ denotes that inverse properties can be expressed where $P1(x, y)\ iff\ P2(y, x)$, and $P1$ and $P2$ are properties; $\mathcal{F}$ denotes that functional properties can be used where $P(x, y)$ and $P(x, z)$ implies $y = z$, and P is a property; $\mathcal{D}$ denotes that datatype properties, data values or data types can be expressed, such as Booleans. Compared with the other subspecies of OWL, OWL Lite has a limited use of properties use to define classes, for example it has limited cardinality values of 0 and 1;

OWL DL was designed to guarantee computational completeness with the maximum expressiveness possible (McGuinness and van Harmelen, 2004). This language has a DL expressivity of $\mathcal{SHOIN}(\mathcal{D})$ (see Table 2.1), which builds upon the features expressed in OWL Lite with the addition of the features denoted by $\mathcal{O}$ which expresses nominals (enumerated classes) and $\mathcal{N}$ which enables cardinality restrictions;

OWL Full uses different semantics from the OWL Lite and DL languages, it is not actually a subclass of OWL as it provides unconstrained use of RDF constructs (McGuinness and van Harmelen, 2004). Unlike OWL Lite and OWL DL, it is the only true extension to RDF(S) because it does not restrict the use of its constructs. OWL Full can be used to express any features described in Table 2.1 and it cannot guarantee computational completeness; therefore it cannot be guaranteed that it can be used to infer logical consequences from the ontology.

The above description of the family of languages including OWL Lite, OWL DL, and OWL Full, are collectively and retrospectively referred to as OWL 1 with the advent of OWL 2. Specifically, OWL 2 is a more expressive version of OWL 1 because it adds a number of new features, and defines three profiles for particular application scenarios [2]. It defines the following new features:

1. **Keys**: a class can be defined as having a particular property, or set of properties as its key, which uniquely identifies that individual;
2. **Property chains**: are a chain of properties that can be inferred to as a single property. For example, the property chain `a:hasMother a:hasSister` infers `a:hasAunt`, because the the sister of someone's mother is that person's aunt;
3. **Richer datatypes, data ranges**: are used to define a datatype with lower and upper bounds (such as an integer greater than 18), the support of booleans, as well as maximum/minimum length of strings;
4. **Qualified cardinality restrictions**: extends the support in OWL 1 for cardinality constraints to fully qualify those constraints. For example, OWL2's cardinality restrictions allow the specification that a fire vehicle must have 4 parts, where those parts must be of type *wheel*;
5. **Asymmetric, reflexive, and disjoint properties**: for specifying relationships that can be one-way only (asymmetric), that apply to an object itself (reflexive), and that must have different values (disjoint);
6. **Enhanced annotation capabilities**: enables the ability to annotate specific relationships. For example, the annotation capabilities can be used to explain why an individual is the subclass of another specific individual.

OWL 2 defines three sub-languages, known as *profiles* that offer advantages in different application scenarios:

1. **OWL 2 EL**: Enables polynomial time algorithms for all the standard reasoning tasks, and particularly suits vary large ontologies that require performance guarantees;
2. **OWL 2 QL**: Enables conjunctive queries to be answered in logarithmic space using relational database technology, so that the maximum memory required to query an ontology is equal to the logarithm of its full size. For example, an ontology that is 100MB will only use 2MB of memory during queries, since $log(100) = 2$. This enables querying of ontologies that are larger than the memory available. OWL

[2] OWL 2 Web Ontology Language Overview: `http://www.w3.org/TR/owl2-overview/`

2 QL suits applications with lightweight ontologies that contain a large number of individuals, and that need to access the data directly using relational query languages such as SQL;

3. **OWL 2 RL**: Enables implementation of polynomial time reasoning algorithms using rule-extended database technologies directly on RDF triples. OWL 2 RL is suited to applications with lightweight ontologies that contain a large number of individuals, and that need to operate on the data directly as RDF triples.

OWL 2 became a W3C recommendation on 27th October 2009[3]. In 2007, a survey was performed to evaluate the popularity of ontology languages, and OWL was found to be the most popular ontology language in use (Cardoso, 2007) (see Figure 2.1). More recently, the popularity and success of OWL has been noted by Aslani and Haarslev (2010), Hartmann et al. (2010), and Chou and Fan (2010).

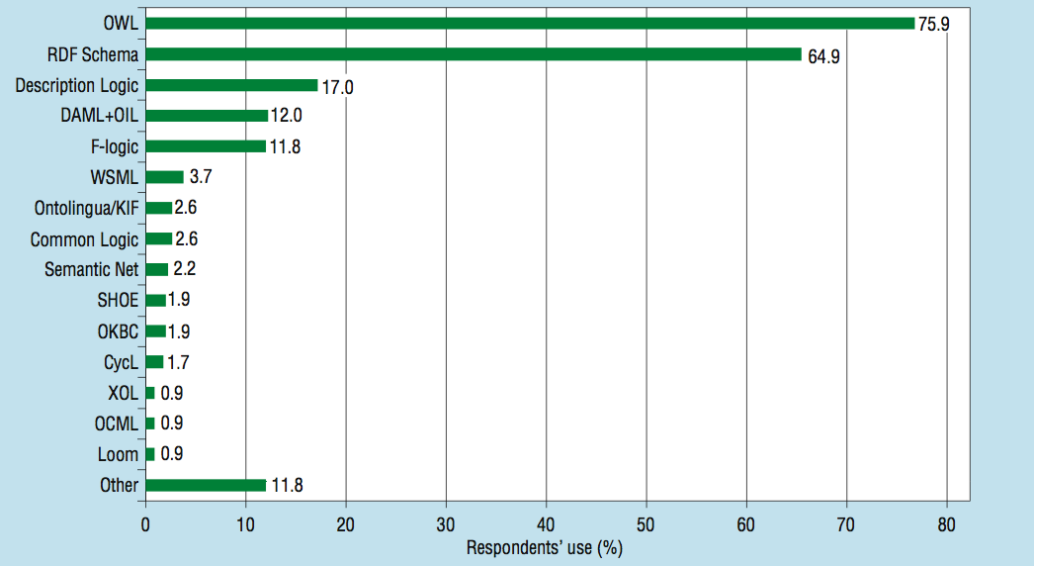


FIGURE 2.1: Ontology languages used in 2007, taken from Cardoso (2007).

In summary, we will focus on developing a framework that uses OWL ontologies because of their widespread adoption within the Semantic Web community, thus fulfilling the need for a standard ontology language for our framework (see Requirement 1 in Section 1.3.1). However, we require that our evolution algorithms will not use any OWL specific features because this will enable our evolution algorithms to be translated to other ontology languages. Therefore, we will consider common structural elements (such as concepts, subsumption relationships, and relationships) in order to determine what information should be augmented and pruned from ontologies (see Requirement 2c in Section 1.3.2).

[3]OWL2 Syntax: `http://www.w3.org/TR/owl2-syntax/`

2.1.3 Reasoning with OWL Ontologies

In this section, we first introduce reasoning over an ontology and second discuss how the composition of an ontology affects the resources and time required for reasoning over it. We consider how the composition of an ontology affects reasoning because we require measures for evaluating the effectiveness of augmenting and pruning an ontology with learning and forgetting algorithms. Developing an algorithm which creates a complex ontology can negatively affect the time it takes to receive a response from the ontology, we require that we minimise the time taken to receive responses (see Requirement 2, in Section 1.3.2). In this section, we aim to evaluate ontology complexity measures for use in our framework (see Requirement 3, in Section 1.3.3), so that we can use them in an evaluation.

Given the above, we introduce how to infer implicit facts from an ontology. In more detail, an implicit fact is not explicitly stated in the ontology but can be inferred. For example, consider the following two facts: 1) '`fire_engine`s are only driven by `fire_fighter`s'; and 2) '`Fred` drives a `fire_engine`'. We can therefore infer that '`Fred` is a `fire_fighter`', because only `fire_fighter`s can drive `fire_engine`s and therefore `Fred` must be a `fire_fighter`. The fact that '`Fred` is a `fire_fighter`' is not an explicitly stated fact in the ontology, and it is therefore an implicit fact. It is possible to deduce implicit facts from ontologies because of their formal semantics. In order to deduce a statement, it must be true for all of the possible instantiations of the domain that are compatible with the statement. These different instantiations are known as models. The models of an ontology represent the only possible realisable situations. For example, if an ontology states that `chemical_fire` is a subclass of `fire` (i.e. in any possible situation, each `chemical_fire` is also a `fire`), and if it is known that `fire3875` is a `chemical_fire` (i.e. `fire3875` is an instance of the `chemical_fire` class), then in any possible situation it is necessarily true that `fire3875` is a `fire`, since the situation in which it would not be a `fire` is incompatible with the constraints expressed in the ontology.

In more detail, given the features of an ontology: concepts, individuals, relationships, subsumption and sub-properties (see Section 2.1.1), it is possible for the following types of deduction (also known as inferences):

- **Classification**: This type of inference deduces whether an object is an instance of a class. For example, if in the ontology it is stated that `fire8734` is an instance of `chemical_fire`, and that `chemical_fire` is a subclass of the `fire` class, then we can infer that `fire8734` is an instance of `fire`, because this latter statement is true in all the models of the ontology;
- **Subsumption**: This type of inference deduces all the subclass relationships between the existing classes in the ontology. For example, if in the ontology it is stated that the class `reactive_chemical_fire` is a subclass of the `chemical_fire`

class, and that `chemical_fire` is a subclass of the `fire` class, then we can infer that `reactive_chemical_fire` is a subclass of `fire`. This deduction holds because in any model of the ontology the extension of `reactive_chemical_fire` is a subset of the extension of `chemical_fire`, and the extension of `chemical_fire` is a subset of the extension of `fire`. Therefore, in any model the extension of `reactive_chemical_fire` is a subset of the extension of `fire`, and in any model the statement that `reactive_chemical_fire` is a subclass of `fire` is true;

- **Equivalence of Classes**: This type of inference deduces whether two classes are equivalent. For example, if class `fire_engine` is equivalent to class `fire_truck`, and class `fire_truck` is equivalent to class `fire_fighting_truck`, then `fire_engine` is also equivalent to `fire_fighting_truck`. This is an example of mutual subsumption;

- **Consistency of a Concept**: This type of inference deduces whether a concept is consistent, where consistent means there is no contradictory axiomatic statements about a concept; that is to say that an inconsistent concept, by definition, can have no instances. For example, given an ontology in which the class `civilian-firefighter` is defined to be a subclass of two disjoint classes `civilian` and `firefighter`, it can be inferred that the class `civilian-firefighter` is inconsistent, since in every model of the ontology its extension is empty. All instances of `civilian-firefighter` would oppose the constraints imposed by the ontology because they are disjoint. In this case, it would be possible to remove the inconsistency for the `civilian-firefighter` class by removing from the ontology the disjointness statement between `civilian` and `firefighter`;

- **Consistency of the Ontology**: This type of inference deduces whether an ontology is consistent, where consistent means there are no contradictory axiomatic statements about the concepts in an ontology. For example, suppose the following statements are in the ontology:

  1. `John` is an instance of both the class `civilian` and the class `firefighter`;
  2. `civilian` and `firefighter` are two disjoint classes.

  Then we have an inconsistency because the above two constraints cannot be satisfied simultaneously. Statement 2 states that the extensions of the two concepts cannot be the same concept type because they are disjoint, but statement 1 states that `John` is an instance of both classes. These two statements are contradictory and indicates that there is an error in the ontology.

The inferences can benefit agents by enabling them to identify classification of instances, subsumption relationships, and the consistency of concepts and its ontology. For example, an agent's ontology contains the concept `fire_engine` and augmented its ontology

with the concept `water_tender`, these two concepts are equivalent, but this is not explicitly stated in its ontology. Using a reasoner the agent can identify the equivalence of these two concepts because they have the same property relationships. Therefore, the agent can assign more fire fighters to fire vehicles, thus increasing the number of fire vehicles attending fires. This is not currently a research objective, we require techniques that will enable agents to select which concepts to augment into their ontology. However, agents using a reasoner may be more efficient, and therefore we will consider extending our framework to enable agents to reason over their ontologies (see Section 7.3).

While we have identified that the agents in our framework will not perform reasoning tasks over their ontologies, we require that our algorithms are effective at augmenting and pruning an ontology (see Requirement 2c, in Section 1.3.2). However, we also require that our approach is generally applicable (see Requirement 2, in Section 1.3.2), and therefore our developed algorithms should not be confined to frameworks that do not enable their agents to perform reasoning tasks. We also require the effectiveness of our algorithm to be compared with state-of-the-art approaches (see Requirement 3, in Section 1.3.3), with respect to the complexity of the evolving ontology. To this end, we now analyse how the composition of an ontology affects its use by providing an overview of how reasoning tasks are performed. A semantic reasoner (also known as simply a reasoner) can be used to make the above inferences, and is a piece of software which applies such rules to a set of asserted statements within an ontology. The rules used for inference differ between reasoners, this is because ontologies can have many expressivity features (see Table 2.1 for a list of possible features) which are used to infer knowledge. Therefore, the rules that reasoners use can be optimised for ontologies with specific sets of features, and thus the reasoner used affects the time complexity of inference of an ontology.

In order to evaluate complexity measures for ontologies, we first consider the basic procedure of reasoning over an ontology. DL reasoners (used for OWL ontologies) use tableaux which use a set of rules to break down complex statements into smaller and simpler pieces in order to detect contradictory information within the statement. If no more simplification or replacement rules can be applied and there are no contradictions then these statements are satisfiable and the tableau is closed. In more detail, in order to find a closed tableau a search method starts with an empty tableau and recursively applies every possible clause in the ontology to each literal. In general, search algorithms expand the tableaux tree breadth-first, examining siblings of the current tableau before examining their children, because an ontology's branch can be infinitely deep.

The above search algorithms can require a large amount of memory space because the breadth of a tree can grow exponentially, rendering the reasoner unable to determine entailments. The space complexity, which is the worst case computer memory space required to complete an algorithm, for breadth-first search of a tree is $O(b^d)$ where b is the branching factor and d is the edge depth from the root. This complexity considers a

graph with nodes and edges which is representative of the subsumption in an ontology, and does not consider the expressive features and their implications in an ontology. Bao and Honavar (2005) note that the factors (breadth and depth) identified by the breadth-first search are limited in modular ontologies and this limitation may be used to optimise the time complexity of reasoners. In more detail, a modular ontology is an ontology that is divided into modules (also known as fragments) which contain a subset of axioms from the ontology. Therefore, using a limited amount of axioms to reason over reduces the breadth and width of the ontology. In contrast to Bao and Honavar's search algorithm, a reasoner uses the properties of the concepts in its ontology and its structure, and this affects the tractability and the resources required to perform an inference. For example, the number of concepts, the branch factor (the number of children of a concept) and the concept hierarchy depth, affect the search space of the tableaux and the time complexity of the reasoner. The work of Lin and Sirin (2008) looks at how such features in an ontology adversely affect the time complexity of a reasoner. They present the Pellint tool, which aims to address performance issues in reasoners by identifying such factors in an ontology. Specifically, this tool detects and corrects ontology axioms that will cause performance problems of the internal tableaux for DL reasoner Pellet (Sirin et al., 2007). While the optimisations differ for each DL reasoner, there are types of axioms that will generally affect tableaux-based reasoners (Lin and Sirin, 2008). The two main sources of complexity in tableaux reasoning are non-determinism in 'completing' a graph, and the size of the graph built. Non-determinism occurs due to disjunctions when using the `unionOf` axiom constructs, which may cause the reasoner to backtrack if a non-deterministic choice is made in one point in the graph that is determined to be false elsewhere. Having the reasoner backtrack and undo changes made due to this decision increases the time complexity of the reasoner, particularly when this occurs multiple times in a single execution. The size of the completion graph depends on the number of asserted instances and on the size of existential restrictions. The number of nodes in the complexity graph increases when axiom constructs such as `SomeValuesFrom`, `MinCardinality`, and `ExactCardinality` are present in an ontology. Applying tableaux rules to these new nodes may increase the non-determinism involved with reasoning, causing potential performance problems.

There have also been criticisms of the brittleness of OWL reasoners, with regard to the predictability of the time complexity of reasoning (Rector and Stevens, 2008), particularly as small changes to an ontology can cause the time complexity for classification to expand exponentially. While the discussed factors affect the time taken to reason over an ontology there is no agreed or defined mapping between a specific constraint, and the time and/or space overhead that it creates. These issues directly affect the development of an agent learning or forgetting algorithm because we require that it can reason logical entailments from its ontology (see Requirement 2, in Section 1.3.2). The ability to determine the time and space complexity would enable an agent to determine whether its ontology has become too complex and thus it can aim to reduce this complexity so

that it can infer entailments from it once again. In addition to this, we also require the ability to compare the time and space complexity of an ontology so that we can compare ontology evolution techniques (Requirement 3, see Section 1.3.3), where a technique that evolves an ontology with the required knowledge with the lowest ontology complexity is optimal. Given this, we will further investigate techniques (see Section 3.4) that can evaluate the time and space complexity of an ontology so that we can compare ontology evolution approaches.

2.2 Evolving Ontologies

The process of learning and forgetting concepts from an ontology is referred to as "ontology evolution" (Haase and Völker, 2008; Wang et al., 2009). Ontology evolution is a process that involves modifying an ontology by either adding or removing axioms. The evolution of an ontology's T-Box is caused by a change in either the domain, conceptualisation, or specification of the ontology (Klein and Fensel, 2001). The T-Box of an agent's ontology specifies the vocabulary for a domain; we require that when the agent's domain changes (see Requirement 2a, in Section 1.3.2), the T-Box evolves to remain representative of that domain (Stojanovic et al., 2003). In other cases, an ontology might evolve due to a change of the ontology's purpose (Noy and Klein, 2004) or there may be the desire to incorporate additional functionality according to changing requirements (Haase and Stojanovic, 2005a). Furthermore, new knowledge may become accessible or different features of the domain may become important (Heflin et al., 1999).

Evolving an ontology is either an addition or subtraction of axioms. In particular, augmenting an ontology increases the risk of inconsistent concepts (described in Section 2.1.3) where there are contradicting axioms about a concept. Both augmenting and pruning from an ontology can change the intended meaning of its concepts. Despite these potential disadvantages, the research community has proposed approaches that augment and prune ontologies while taking into account the risk of contradictions. In general, augmentation approaches are categorised in two ways: i) as merging, whereby two ontologies are combined together to form one ontology; and ii) as extending, whereby additional axioms are added to an existing ontology (see Section 2.2.1). There are also general approaches for pruning an ontology, which all remove a set of axioms from an ontology (see Section 2.2.2). These processes are necessary for developing a technique that can evolve an agent's ontology so that it remains consistent.

Ontology evolution can therefore be summarised with the following tasks:

1. **Merge**, where two ontologies o_i and o_j are combined into a new ontology o' (see Figure 2.2);

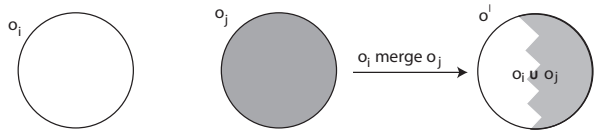


FIGURE 2.2: The **merge** task of ontology evolution.

2. **Extend**, where an ontology has additional axioms augmented into it (see Figure 2.3);

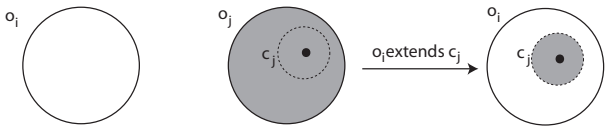


FIGURE 2.3: The **extend** task of ontology evolution.

3. **Forget**, where an ontology has axioms pruned from it (see Figure 2.4);

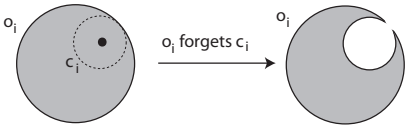


FIGURE 2.4: The **forget** task of ontology evolution.

4. **Extract Module**, where a subset of axioms, known as a module, is extracted from an ontology (see Figure 2.5).

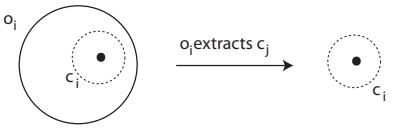


FIGURE 2.5: The **extract module** task of ontology evolution.

The above methods describe basic operations that can be performed on an ontology and enable axioms to be manipulated in ways that are used during ontology evolution. Operations that learn or forget axioms from ontologies can be abstracted to the above tasks, and therefore any learning or forgetting algorithms, including our proposed learning and forgetting algorithms, will use them. In all of these cases, most research which evolves ontologies focuses on human interaction to resolve conflicts: tools presented by Stojanovic et al. (2002) and Kalfoglou and Schorlemmer (2003), which merge two ontologies by allowing users to confirm which concepts will be merged. However, we proceed by surveying literature that merges ontologies or forgets concepts from ontologies without human interaction, and consider how our approach can benefit from learning and forgetting fragments or modules of ontologies. This is because we aim to develop

evolution algorithms that are automatic, as we require that an ontology can provide fast responses, whereas relying on manual or semi-manual processes can significantly increase response time or block the ontology's use (see Requirement 2 in Section 1.3.2). In particular, Section 2.2.1 considers approaches to merging ontologies automatically and the considerations needed to effectively augment ontologies. Following that, in Section 2.2.2 we detail approaches which remove concepts from ontologies.

2.2.1 Merging Ontologies

Ontologies are created independently, and there is no requirement for authors to collaborate. This distributed nature of the development of ontologies means that many have overlapping domains, and in order to reuse the knowledge that they contain, they need to be merged or aligned. Merging ontologies results in a single ontology that contains all the information from the merged ontologies, whereas aligning ontologies results in mappings between concepts in the ontologies. We are interested in developing a technique that can automatically evolve agents' personal ontologies so that they can benefit from localised knowledge thus not having to request information each time it is required (fulfilling Requirement 2a, see Section 1.3.2). Therefore, we proceed by focusing on literature that not only merges ontologies but supports automatic evolution, and do not discuss manual or semi-automatic merging tools or alignment techniques.

When merging ontologies, it is important that the merging process maintains logical consistency because an inconsistent ontology can provide a user with incorrect information; McGuinness (2000) details the issues of maintaining consistency of evolving ontologies. These issues are fundamental to our research because they enable an ontology to be augmented with new knowledge so that it can facilitate reasoning and the automatic exchange of axioms (see Requirement 2a, in Section 1.3.2). We consider two ontologies, o_i and o_j, where the aim is for a concept c_j contained in o_j to be merged with o_i, creating a new ontology noted as o'_i, where $o'_i = o_i \cup axioms(c_j)$, and the function $axioms(c)$ returns all of the axioms describing concept c. McGuinness identifies the following four issues that affect logical soundness:

- Concept c_j needs to be **verified** before it is merged with o_i. This can be achieved by using a DL-reasoner over the ontology. Depending on the scenario, it may be important to verify that the merged ontologies have the same domain so that there are no problems with polysemy. Polysemy occurs when a term has multiple different meanings, and therefore it is important to verify that these different meanings are not incorrectly reconciled, which can be addressed by automatic ontology similarity measures;
- Concept c_j needs to be **validated** with ranges and/or unit measures. For example, if an ontology states that a `CivilianCar` can transport a maximum load capacity

of `500KG`, then all instances of `CivilianCar`s must be checked to ensure that they do not define load capacities over `500KG`;

- The use of **ontology versioning** to ensure that c_j is referenced as intended. A future merge operation may alter the context of c_j, so it is important that any references are inferred logically;

- Recognising **ontology patterns** that suggest invalidity, such as cyclic definitions. For example, `fire_engine` subclass of `fire_vehicle`, `fire_vehicle` subclass of `vehicle` and `vehicle` subclass of `fire_engine`; and inverted relationships, for example `fire_vehicle` equipment of `fire_hose`. However, not all cyclic definitions are invalid but should be checked for logical soundness.

Verifying that a set of concepts is consistent with an ontology before they are added to an ontology is particularly useful during an agent's learning process because it minimises the risk that an agent will make its ontology inconsistent. While it is important to ensure that an evolving ontology is consistent, for now our work focuses on which concepts to augment and prune from an agent's ontology. In order to evaluate ontology evolution algorithms, we will use a closed environment which negates the need to check for consistency. This is because you can ensure that the union of all the ontologies' axioms contains no inconsistencies. However, this would not be the case in an open environment where there is no control over the ontologies from which an evolving ontology can learn. We will consider developing an algorithm for evolving ontologies in an open world environment, such as the World Wide Web, for future work (see Section 7.3). The other techniques suggested by McGuinness (2000) are not fully implemented, or require human interaction to specify and control changes, therefore we will not be using them because we require that we augment ontologies automatically (see Requirement 2a, in Section 1.3.2).

2.2.2 Forgetting Concepts from Ontologies

The importance of discarding irrelevant information has long been recognised to be beneficial to humans and intelligent agents alike (Wang et al., 2005). In this section, we focus on forgetting from ontologies, where forgetting from a DL ontology happens either natively in the DL formalism, or via translation to another formalism, such as logic programming.

There are three main motivations to remove concepts from an ontology:

1. **Storage**: A large ontology requires more resources (such as disk space and memory usage) than a smaller ontology;

2. **Maintenance**: A large ontology is often more complex in terms of relationships and thus requires more computational power to check the consistency of new concepts, and remove irrelevant or out of date concepts;

3. **Use**: A large ontology requires more memory resources to load and locate information than a smaller ontology. Inferring knowledge from large expressive ontologies is also computationally expensive.

These motivations are recognised by Wang et al. (2009): they note that it is "expensive to construct large ontologies" but it is "even more expensive to host, manage, and use large" ontologies. They also note that it is desirable to have the ability to reduce large ontologies to smaller ontologies that meet the specific needs of a specific application. This translates to our requirements because agents have different domains, and thus an agent will require concepts that aid its function and large ontologies that contain concepts unnecessary to its function will impede its performance. In our work, we require that agents can build a small ontology and can manage and use its ontology within a reasonable time frame. Specifically, our work aims to reduce the number of concepts in an ontology to maintain a small usable ontology, and hence want to adopt a semantic forgetting technique to remove concepts from an ontology (see Requirement 2c in Section 1.3.2). In order for us to adopt a semantic forgetting technique, we consider: different techniques that can be used to select a subset of concepts to forget; the basics and notation of forgetting; and how to remove concepts from an ontology.

A technique that aims to address our requirements of removing concepts from an ontology (see Requirement 2c, in Section 1.3.2), can apply the following techniques to remove a subset of concepts from an ontology:

1. Remove one concept at a time as described by Eiter et al. (2006) and Wang et al. (2008) (see Figure 2.6);

2. Remove using a subtree or branch removal, or subtree extraction from an ontology. These techniques are used for graphs that can be modelled as trees. In more detail:

   (a) **Subtree Removal:** A subtree is a set of nodes that are hierarchically linked, the subtree root node has a parent/s in the graph and all of the nodes under the root node are included in the subtree (Kaushik et al., 2005). The subtree is then removed from the graph, as shown in Figure 2.7 part i);

   (b) **Branch Removal:** A branch removal selects the root node of the graph and a set of nodes that relate to it hierarchically (Kaushik et al., 2005), as shown in Figure 2.7 part ii);

   (c) **Subtree Extraction:** A subtree of concepts is a hierarchical tree structure with a set of linked nodes where the root node of the subtree is not necessarily

the root of the original structure, and does not necessarily include all the leaf nodes of the original structure (Kaushik et al., 2005) (illustrated in part iii) of Figure 2.7). The selection of a subtree is described by Shasha and Zhang (1997) and to date has been predominately used for tree editing.

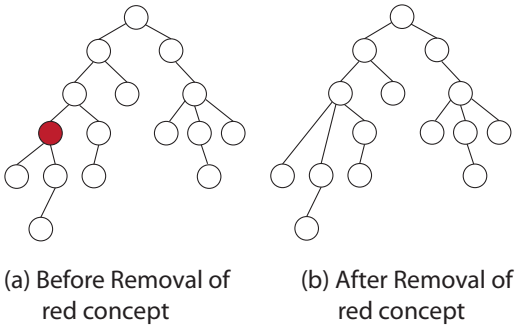


FIGURE 2.6: Concept removal, where a single concept depicted in red is removed from an ontology.

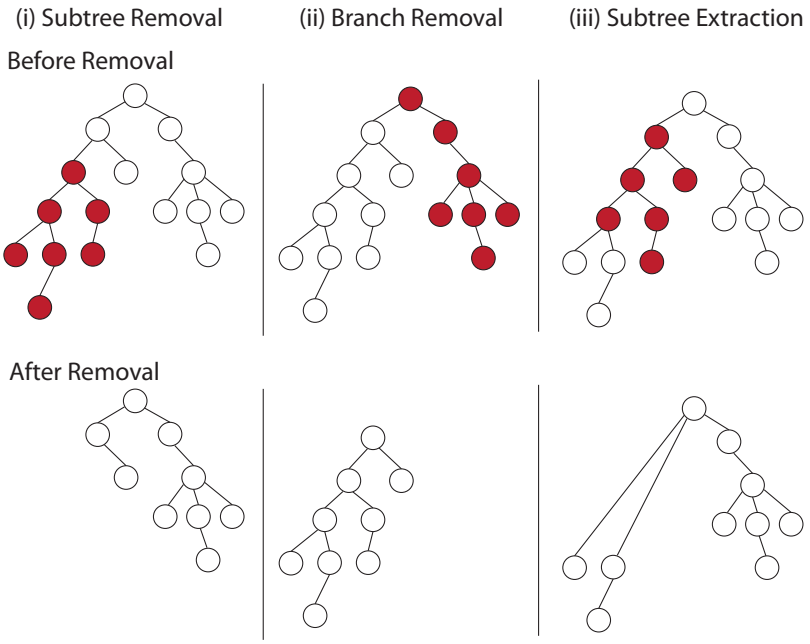


FIGURE 2.7: Subtree removal, branch removal and subtree extraction, where the red nodes are removed.

While the above two approaches can be used to remove concepts from an ontology, they do not consider how to reduce the costs associated with forgetting, as stated in Requirement 2c (see Section 1.3.2). In more detail, the first approach removes one concept at a time and thus requires many removal operations to remove more than one concept. An agent reducing the size of its ontology would benefit from removing more than one concept at a time because it would be able to avoid a thrashing scenario where concepts are swapped in and out of the ontology. For each forgetting operation there is an overhead cost associated with the removal of the concept, which includes the time taken to load, remove, and save the ontology. The second approach does not consider

optimising which concepts to remove because it is limited by the ontology's subsumption relationships and can only remove concepts that are connected through subsumption. We require that our approach is able to remove concepts that are the least useful to the agent (see Requirement 2c, in Section 1.3.2) and neither of these two approaches are able to achieve this efficiently. We will develop an algorithm that can remove more than one concept that is deemed not to be useful at a time. Next, we consider different approaches to remove concepts from ontologies. While these approaches only remove one concept at a time, we can use their approaches to remove more than one concept and maintain the consistency of an ontology.

There has been a literature focus on forgetting in logic programming (Wang et al., 2005; Eiter and Wang, 2006), but its application to OWL ontologies is in its early stages. Specifically, Eiter et al. (2006) state that "an explicit notion of forgetting has not been given yet for this class of languages". In order to address this, Eiter et al. adapted their approach for forgetting rules from HEX-programs (where "HEX" stands for Higher-order with EXternal atoms) (Eiter et al., 2005). Such HEX-programs were introduced as a generic rule-based language fostering a hybrid integration approach towards implementing the Rule Layer of the Semantic Web. In particular, Eiter et al. (2006)'s approach re-factors OWL Lite ontologies into Horn logic, simplifies them, and removes a specified concept. They use four stages to remove an atom r from an OWL/RDF ontology o: First, the axioms of o are translated into rule representation, P_o, using the Tables 2.2 and 2.3. Second, apply weak unfolding on P_o. This unfolds the entailments made within the program, where entailments are expanded so that they can be independent. For example, given that a fire brigade requires the removal of the relationship *providesSupport* from its ontology because it no longer desires to assign vehicles to provide support based on whether they have the same suppressant, using the above approach it first unfolds the rules $providesSupport(X, Y) \leftarrow haveDryAgent(X, Y)$, and $haveDryAgent(X, Y) \leftarrow extinguishReactiveFires(X, Y)$ results in $providesSupport(X, Y) \leftarrow extinguishReactiveFires(X, Y)$. In this example, we remove rules containing *haveDryAgent*; without unfolding these rules, then we would lose the entailment $providesSupport(X, Y) \leftarrow extinguishReactiveFires(X, Y)$. The following examples are based on those from Eiter et al. (2006), where they aim to remove a rule, l, in this example *providesSupport*, from a program P_L defined in Table 2.4.

First they apply weak unfolding to P_o, and expand the rules on lines 4 and 5 to result in the rule on line 6. Second, the most general unifiers X and Y are replaced with `Vehicle546` and `Vehicle127`, respectively. This process results in P'_o, as shown in Table 2.5.

Statement	**DL syntax**	**Rule Representation**
subClass	$D \sqsubseteq C$	$C(X) \leftarrow D(X).$
subPropertyOf	$P \sqsubseteq Q$	$Q(X,Y) \leftarrow P(X,Y).$
domain	$T \sqsubseteq \forall P^{-}.C$	$C(X) \leftarrow P(X,Y).$
range	$T \sqsubseteq \forall P \cdot C$	$C(Y) \leftarrow P(X,Y).$
class-instance	$a : C$	$C(a) \leftarrow .$
property-instance	$< a, b >: P$	$P(a,b) \leftarrow .$
class-equivalance	$D \equiv C$	$D(X) \leftarrow C(X),$ $C(X) \leftarrow D(X).$
property-equivalance	$P \equiv Q$	$P(X,Y) \leftarrow Q(X,Y),$ $Q(X,Y) \leftarrow P(X,Y).$
inverseOf	$P \equiv Q^{-}$	$P(X,Y) \leftarrow Q(Y,X),$ $Q(X,Y) \leftarrow P(Y,X).$
transitiveProperties	$P^{+} \sqsubseteq P$	$P(X,Y) \leftarrow P(X,Z), P(Z,Y).$

TABLE 2.2: The mappings between ontology statements to rules, taken from Eiter et al. (2006).

Constructor	**DL syntax**	**Rule Representation**
conjunction	$C_1 \sqcap C_2 \sqsubseteq D$	$D(X) \leftarrow C_1(X), C_2(X).$
	$C \sqsubseteq D_1 \sqcap D_2$	$D_1(X) \leftarrow C(X);$ $D_2(X) \leftarrow C(X).$
disjunction	$C_1 \sqcup C_2 \sqsubseteq D$	$D(X) \leftarrow C_1(X);$ $D(X) \leftarrow C_2(X).$
existential restriction	$\exists P.C \sqsubseteq D$	$D(X) \leftarrow P(X,Y), C(Y).$
universal restriction	$S \sqsubseteq \forall P.C$	$C(Y) \leftarrow P(X,Y), D(X).$

TABLE 2.3: The mapping of the ontology constructors to rules, taken from Eiter et al. (2006)

1	$haveDryAgent($`Vehicle546`,`Vehicle127`$)$	$\leftarrow$	.
2	$providesSupport(X,Y)$	$\leftarrow$	$providesSupport(Y,X).$
3	$providesSupport(X,Z)$	$\leftarrow$	$providesSupport(X,Y),$ $providesSupport(Y,Z).$
4	$extinguishReactiveFires(X,Y)$	$\leftarrow$	$providesSupport(X,Y).$
5	$providesSupport(X,Y)$	$\leftarrow$	$haveDryAgent(X,Y).$

TABLE 2.4: An example of a program expressed in rule representation.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
2	$providesSupport(X, Y)$	$\leftarrow$	$providesSupport(Y, X)$.
3	$providesSupport(X, Z)$	$\leftarrow$	$providesSupport(X, Y)$, $providesSupport(Y, Z)$.
4	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$providesSupport(X, Y)$.
5	$providesSupport(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
6	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
7	$providesSupport(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
8	$extinguishReactiveFires(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.

Table 2.5: An example program which has been weakly unfolded.

Third, remove all rules containing r. In our example, removing the rule *providesSupport* from P_o would result in the rules presented in Table 2.6.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
2	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
3	$extinguishReactiveFires(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
4	$providesSupport(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.

Table 2.6: An example program after rule *providesSupport* has been removed.

Fourth, transpose the resulting ontology from the rule representation to their axioms using the Tables 2.2 and 2.3, and is shown in Table 2.7.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$.
2	$haveDryAgent \sqsubseteq extinguishReactiveFires$.
3	$extinguishReactiveFires(\texttt{Vehicle546},\texttt{Vehicle127})$.
4	$providesSupport(\texttt{Vehicle546},\texttt{Vehicle127})$.

Table 2.7: An example program after being transposed from rule representation in DL syntax.

In addition to the removal of predicates, concepts will also have to be removed, and as such, we provide the following example which walks through the process of removing a concept from an ontology. For example, we require the removal of the concept $Vehicle361$ from the program defined in Table 2.8.

First they apply weak unfolding to P_o. Second, the most general unifiers X and Y are replaced with: `Vehicle546` and `Vehicle127`; `Vehicle127` and `Vehicle361`; and `Vehicle361` and `Vehicle546`, respectively. This process results in P'_o, as shown in Table 2.9.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
2	$haveDryAgent(\texttt{Vehicle127},\texttt{Vehicle361})$	$\leftarrow$	.
3	$haveDryAgent(\texttt{Vehicle361},\texttt{Vehicle546})$	$\leftarrow$	.
4	$providesSupport(X, Y)$	$\leftarrow$	$providesSupport(Y, X)$.
5	$providesSupport(X, Z)$	$\leftarrow$	$providesSupport(X, Y)$, $providesSupport(Y, Z)$.
6	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$providesSupport(X, Y)$.
7	$providesSupport(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.

TABLE 2.8: A second example of a program expressed in rule representation.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
2	$haveDryAgent(\texttt{Vehicle127},\texttt{Vehicle361})$	$\leftarrow$	.
3	$haveDryAgent(\texttt{Vehicle361},\texttt{Vehicle546})$	$\leftarrow$	.
4	$providesSupport(X, Y)$	$\leftarrow$	$providesSupport(Y, X)$.
5	$providesSupport(X, Z)$	$\leftarrow$	$providesSupport(X, Y)$, $providesSupport(Y, Z)$.
6	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$providesSupport(X, Y)$.
7	$providesSupport(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
8	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
9	$providesSupport(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
10	$extinguishReactiveFires(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
11	$providesSupport(\texttt{Vehicle127},\texttt{Vehicle361})$	$\leftarrow$	.
12	$extinguishReactiveFires(\texttt{Vehicle127},\texttt{Vehicle361})$	$\leftarrow$	.
13	$providesSupport(\texttt{Vehicle361},\texttt{Vehicle546})$	$\leftarrow$	.
14	$extinguishReactiveFires(\texttt{Vehicle361},\texttt{Vehicle546})$	$\leftarrow$	.

TABLE 2.9: A second example program which has been weakly unfolded.

Third, remove all rules containing r. In our example, removing $Vehicle361$ from P_o would result in the rules presented in Table 2.10.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
2	$providesSupport(X, Y)$	$\leftarrow$	$providesSupport(Y, X)$.
3	$providesSupport(X, Z)$	$\leftarrow$	$providesSupport(X, Y)$, $providesSupport(Y, Z)$.
4	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$providesSupport(X, Y)$.
5	$providesSupport(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
6	$extinguishReactiveFires(X, Y)$	$\leftarrow$	$haveDryAgent(X, Y)$.
7	$providesSupport(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.
8	$extinguishReactiveFires(\texttt{Vehicle546},\texttt{Vehicle127})$	$\leftarrow$	.

TABLE 2.10: A second example program after $Vehicle361$ has been removed.

Fourth, transpose the resulting ontology from the rule representation to their axioms using the Tables 2.2 and 2.3, and is shown in Table 2.11.

1	$haveDryAgent(\texttt{Vehicle546},\texttt{Vehicle127})$.
2	$providesSupport \equiv providesSupport^{-}$.
3	$providesSupport^{+} \sqsubseteq providesSupport$.
4	$providesSupport \sqsubseteq extinguishReactiveFires$.
5	$haveDryAgent \sqsubseteq providesSupport$.
6	$haveDryAgent \sqsubseteq extinguishReactiveFires$.
7	$providesSupport(\texttt{Vehicle546},\texttt{Vehicle127})$.
8	$extinguishReactiveFires(\texttt{Vehicle546},\texttt{Vehicle127})$.

TABLE 2.11: A second example program after being transposed from rule representation in DL syntax.

A drawback to Eiter et al.'s approach is that the rules or concepts to be forgotten must be explicitly specified; the algorithm does not decide what should be forgotten. In Requirement 2c (see Section 1.3.2) we require a forgetting technique that can reduce the complexity of an ontology, but this algorithm does not perform analysis to determine which rules or concepts to forget. In Requirement 2, we also require algorithms to perform efficiently, and with low overhead resource costs. This approach is not optimised for efficiency, and incurs a large memory overhead because it translates DL syntax into a rules language, and back again, whereas an algorithm which is capable of forgetting from an ontology in-place does not incur costs relating to translation. We also note that although Eiter et al.'s definition of forgetting preserves meaning, it "has some restriction on axioms in the ontologies" (Qi et al., 2008). In particular, the list of statements in Table 2.2 and the literature suggest that this approach does not handle contra-positives. Their rules dictate that logical implication can only be defined in one direction, the positive direction (from conditions to consequences, i.e. $D \sqsubseteq C$), and not in the contra-

positive direction (the negation of the consequence entails the negation of the condition, i.e. $\neg D \sqsubseteq \neg C$). For example, consider the following: if it is known that vehicles are produced by a company and that all of their fire engines are only bought by fire departments, then the fire engine can be used to put out fires. Using the implication from this rule, in the positive direction, we conclude that a vehicle can be used to put out fires, provided that it was bought by a fire department. If the negation of the consequence is given, it cannot lead to the conclusion that one of the rule conditions does not hold. However, the approach of Eiter et al. (2006) does not consider contra-positive rules, which will affect the completeness of reasoning tasks (Qi et al., 2008). Thus, any approach that utilises this technique will not be suitable for creating ontologies that may be reasoned over. To avoid applying this constraint to our approach, and due to the technique not meeting our requirements we will not use this technique in our approach.

Similar to Eiter et al. (2006), Wang et al. (2008) also propose a technique to forget concepts from an OWL ontology. They define the resulting TBox, TB', after removing a concept c_A from a TBox TB. After the removal of c_A:

- TB' should not contain any occurrence of c_A;
- TB' is weaker than TB because it does not contain c_A and therefore can make fewer entailments to the instances in its knowledge base;
- TB' should give the same answer to any query that is irrelevant to c_A.

In order to remove c_A from TB, Wang et al. use the algorithm shown in Algorithm 1. To understand this algorithm we first must define c_B as a basic concept, and c_C as a general concept. The former is a concept on the left hand side of the subsumption, and the latter is on the right hand side of the subsumption.

Algorithm 1 Algorithm to compute the result of forgetting in DL-Lite, where c_A is an atomic concept that we wish to remove from TB, c_B is a basic concept, and c_C is a general concept.

Step 1. Remove axiom $c_A \sqsubseteq c_A$ from TB if it is present.
Step 2. If axiom $c_A \sqsubseteq \neg c_A$ is in TB:
    remove each axiom $c_A \sqsubseteq c_C$ or $c_B \sqsubseteq \neg c_A$ from TB, and
    replace each axiom $c_B \sqsubseteq c_A$ in TB by $c_B \sqsubseteq \neg c_B$.
Step 3. Replace each axiom $c_B \sqsubseteq \neg c_A$ in TB by $c_A \sqsubseteq \neg c_B$
Step 4. For each axiom $c_{B,i} \sqsubseteq c_A (1 \leq i \leq m)$ in TB and each axiom $c_A \sqsubseteq c_{C,j} (1 \leq j \leq n)$ in TB:
    where:
        each $c_{B,i}$ is a basic concept, and
        each $c_{C,j}$ is a general concept
            if $c_{B,i} \sqsubseteq c_{C,j}$ is not in TB already:
                add $c_{B,i} \sqsubseteq c_{C,j}$ to TB.
Step 5. Return the result of removing every axiom containing c_A in TB.

For example, the `FireFighter` concept is removed from the following T-Box (see Table 2.12) because a fire brigade is introducing part-time fire fighters who are civilians and fire fighters, and we follow by explaining each step used to remove the concept `FireFighter`.

$$\begin{aligned} \exists inRescue &\sqsubseteq FireVehicle \\ \exists inRescue &\sqsubseteq FireFighter \\ FireFighter &\sqsubseteq Person \\ Civilian &\sqsubseteq \neg FireFighter \\ FireVehicle &\sqsubseteq \exists hasLicensePlate \\ \exists hasLicensePlate^{-} &\sqsubseteq LicensePlate \\ \exists hasLicensePlate \sqcap OutOfService &\sqsubseteq \neg FireVehicle \end{aligned}$$

TABLE 2.12: An example of a T-Box.

The following axioms will be affected if `FireFighter` is removed from the T-Box, hence we will only use the following axioms (see Table 2.13) while applying steps 1-4 of Algorithm 1.

$$\begin{aligned} \exists inRescue &\sqsubseteq FireFighter \\ FireFighter &\sqsubseteq Person \\ Civilian &\sqsubseteq \neg FireFighter \end{aligned}$$

TABLE 2.13: The axioms that contain the concept `FireFighter`.

The following list describes each step from Algorithm 1 applied to the axioms from the T-Box (see Table 2.12) in order to remove the concept `FireFighter`:

Step 1. In this example, there are no axioms that match the pattern $c_A \sqsubseteq c_A$. A matching example axiom would be $FireFighter \sqsubseteq FireFighter$;

Step 2. This step can also not be performed, because there are no axioms that match the pattern $c_A \sqsubseteq \neg c_A$. A matching example axiom would be $FireFighter \sqsubseteq \neg FireFighter$;

Step 3. The axiom $Civilian \sqsubseteq \neg FireFighter$ matches the pattern $c_B \sqsubseteq \neg c_A$, and is replaced by $FireFighter \sqsubseteq \neg Civilian$;

Step 4. The axioms $FireFighter \sqsubseteq Person$ and $FireFighter \sqsubseteq \neg Civilian$ match the patterns $c_{B,i} \sqsubseteq c_A (1 \leq i \leq m)$ and $c_A \sqsubseteq c_{C,j} (1 \leq j \leq n)$, respectively, where $FireFighter$ is c_A, $Civilian$ is a basic concept c_B, and $Person$ is a general concept c_C. Applying step 4 results in the following axioms: $\exists inRescue \sqsubseteq Person$ and $\exists inRescue \sqsubseteq \neg Civilian$;

Step 5. The axioms $\exists inRescue \sqsubseteq Person$ and $\exists inRescue \sqsubseteq \neg Civilian$ replace the axioms $\exists inRescue \sqsubseteq FireFighter$, $FireFighter \sqsubseteq Person$ and $Civilian \sqsubseteq \neg FireFighter$ in the T-Box.

This algorithm can only be applied to the removal of concepts from OWL Lite. This means that this technique can handle the following syntax:

$$\begin{aligned} B \leftarrow &\quad A|\exists R \\ D \leftarrow &\quad B|D_1 \sqcap D_2 \\ C \leftarrow &\quad B|\neg B \\ R \leftarrow &\quad P|P^- \\ S \leftarrow &\quad R|\neg R \end{aligned}$$

and the TBoxes contain axioms in the form of:

$$\begin{aligned} D \sqsubseteq &\quad C \\ R \sqsubseteq &\quad S. \end{aligned}$$

Both Eiter et al. (2006) and Wang et al.'s (2008) approaches can remove a concept from a TBox. In contrast to Wang et al., Eiter et al. require axioms to first be translated from DL syntax to rule representations (as shown in Tables 2.2 and 2.3), and translated back to DL syntax after the expansion of the rules and the removal of a concept has been performed, whereas Wang et al.'s approach can be applied to axioms without need of translation. Using Eiter et al.'s approach is restrictive in that it does not support contrapositives, and only supports OWL Lite and a subset of OWL DL (it does not support all of OWL DL because some description logic constructs such as cardinality constraints have no direct representation in logic programming rules), whereas Wang et al.'s approach is restrictive in that it only supports OWL Lite. Thus, in order to minimise constraining our approach, we will use Eiter et al.'s approach.

The above approaches enable an agent to remove concepts from an ontology. However, they do not provide recommendations in selecting which concepts to remove. Specifically, Alani et al.'s (2006b) approach provides an algorithm to select which concepts to remove based on the use of concepts and properties contained in the ontology. In order to determine whether concepts and properties are used, Alani et al. use a log to record their use. They determine the concepts to remove based on whether the ontology contains any instantiations of a concept, and if any application has used a concept or relationship. In more detail, they use the following process:

1. Retain concepts and relationships that are instantiated in the ontology;
2. Retain concepts and relationships used in application queries;

3. Remove all other concepts and relationships.

This process is called "winnowing". While this process considers whether concepts and relationships are of use and thus could be used to remove concepts that are not useful to an agent (which meets Requirement 2c, in Section 1.3.2), it is not suitable for our work. This is because we require that an agent's performance is not hindered by an agent's ontology with regards to the time taken to query information, therefore an agent may desire to remove concepts from its ontology even though a concept has been used in the past or has instances. In this case, this approach does not enable an agent to select which concepts are best to remove, and therefore we will not be using this approach in our work. However, recording the concepts which are used in an ontology is useful and we will consider methods to analyse which concepts are the least useful.

The decision to forget a concept is closely related to the value of knowledge. In particular, Markovitch and Scott (1988) identify four factors of the value of knowledge: i) relevance, ii) correctness, iii) memory requirements, and iv) influence on search time. These factors are relevant to our scenario as follows:

1. Concepts that have no **relevance** to future queries cannot contribute to any utility. Therefore, we will use a relevance measure in our algorithm to determine which concepts are the least useful. Also, it is important to remove incorrect relevant knowledge because it can cause a detrimental effect on an agent's ability to answer queries;

2. In our case, this may cause the agent's answers to be **incorrect** or prevent them from augmenting their ontologies with new concepts;

3. The **storage costs** associated with retaining relevant knowledge are based on the cost of the memory used to store it, and the amount of storage space required. Typically in most computer systems this is the least costly factor because of the increase of computing power and cheapness of components. However, there is a limitation on a reasoner's capability to infer logical consequences from a T-Box about entities from an A-Box;

4. As identified in Section 2.1, the greater the number of features expressed in an ontology, then broadly speaking the greater the space and time complexity, which increases the **search time**. It is therefore desirable for an agent to keep the size of its vocabulary low, while still retaining knowledge required to complete its tasks. These factors used to measure the value of knowledge are particularly useful to determine when an agent could forget a concept from its ontology.

The above literature (Alani et al. (2006b); Markovitch and Scott (1988)) consider removing concepts when they are not useful in answering queries, and therefore we will use

a relevance measure in our algorithm to determine which concepts are the least useful to an agent. We will also use the fourth recommendation from Markovitch and Scott of removing concepts when there is a negative influence on search time, in our case when an agent cannot perform an action in a given timeframe (see Requirement 2c in Section 1.3.2).

In conclusion, we have identified an approach that can be used to remove concepts from an ontology (Eiter et al., 2006), and determine when and why to remove concepts. We have also identified state-of-the-art forgetting approaches, which remove one concept at a time (Eiter et al. (2006) and Wang et al. (2008)) and remove a subtree (Shasha and Zhang, 1997). However, in order to meet Requirement 2c in Section 1.3.2, we require an approach that can remove a fragment of axioms based on a concept. While subtrees remove more axioms than individual concepts, they will not remove axioms that represent a concept that relates to concepts outside of a subsumption subtree, and as such they do not meet our requirement. Therefore we cannot use them to develop our online forgetting algorithm; however, these alternative techniques (removing one concept and subtree extraction) provide benchmarks for our approach (see Requirement 3 in Section 1.3.3).

2.2.3 Module Extraction to aid Learning and Forgetting from Ontologies

In addition to merging whole ontologies and forgetting single concepts, it is possible to reuse fragments of an ontology to augment an evolving ontology, and prune fragments of concepts that are not required. In more detail, a fragment of an ontology is a subset of axioms relevant to a set of given terms and which guarantees to completely capture these terms (Grau et al., 2007). Ontologies may be made modular during the design phase, or can be divided into modules with an automatic algorithm which selects a subset of axioms when required. The process is called ontology module extraction (Cuenca Grau et al., 2006), and results in a subset of axioms of the ontology being extracted. We explore how modules or fragments can be used to in our framework to disseminate information (see Requirement 1 in Section 1.3.1).

The primary purpose of module extraction is to allow a small self-contained portion of an ontology to be loaded in order to reduce the cost of resources to use. Such modularisation techniques are used on large domain ontologies, for example the GENE[4] and GALEN[5] ontologies. These ontologies contain thousands of concepts, but ontologies of such a size can be computationally expensive to load and infer logical consequences of entities. This scalability problem of large ontologies is noted by Seidenberg and Rector (2006), and they state that ontologies with over ten thousand concepts suffer from

[4]The Gene Ontology: `http://www.geneontology.org/`
[5]GALEN Medical Ontology: `http://www.co-ode.org/galen/`

scaling problems, which affects the time taken to infer new (implicit) information. The majority of modularisation techniques separate an ontology based on its structure and not based on representing a specific concept in each module. We describe modularisation techniques which extract a fragment about a target concept from an ontology, because we require that an agent can specify which concept it desires to learn about (see Requirement 1 in Section 1.3.1). In order to compare the discussed modularisation techniques, we extract a fragment representing the target concept `rescue_worker` from the ontology represented in Figure 2.8.

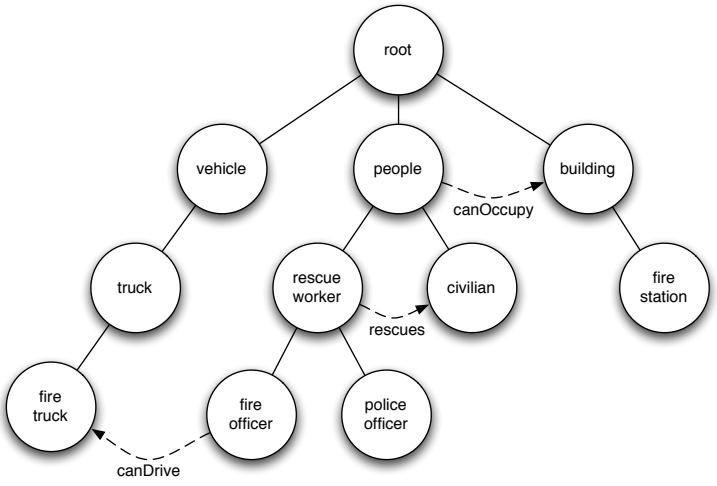


FIGURE 2.8: An example ontology based on the fire rescue domain, showing a subsumption, and range and domain relationships depicted in solid and dashed lines, respectively.

The PROMPTFACTOR algorithm (Noy and Musen, 2003) generates a fragment of an ontology through the use of subclass and existential relationships that impact on the target concept. In order to select which concepts to extract from the ontology, their algorithm follows the following steps:

Step 1: Traverse and select all of the subclasses of the target concept;

Step 2: Traverse and select all superclasses, up to and including the root node, of the target concept, and the classes selected in Step 1;

Step 3: Traverse and select all existential relationships, and the classes related through them, of the target concept, and the classes selected in Step 1.

The only relationship treated differently in Step 3 is `superClassOf`, which is ignored, because the algorithm does not traverse the children of classes. This prevents the whole ontology being selected, which would happen if the children of classes were traversed from the root node.

Given our rescue worker example, we generate a fragment from our example ontology (see Figure 2.8). This approach first selects the target concept `rescue_worker`. It then selects its subclasses: `fire_officer` and `police_officer`. Next, it has to evaluate all relationships from these three nodes, excluding subclass relationships. In this case it selects the concept `fire_truck`, because it is related by the `canDrive` relationship, and it selects the concepts `truck` and `vehicle` because they are related through the `subClassOf` (parent) relationship. The approach also selects the concept `civilian` because it is related through the `rescues` relationship, and the concept `people` which is related through the `subClassOf` (parent) relationship. It then selects the concept `building`, which is related through the `canOccupy` relationship. It does not select the concept `fire_station` because the `superClassOf` (child) relationships are not evaluated. The selected concepts are illustrated in Figure 2.9.

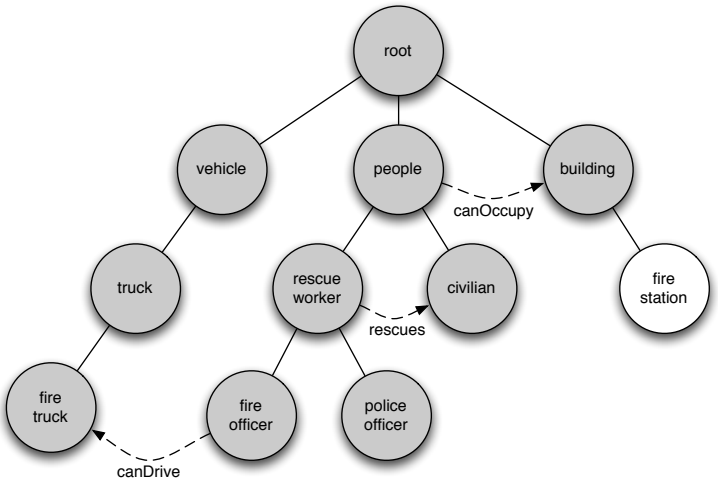


FIGURE 2.9: An example ontology based on the fire rescue domain, showing the concepts selected by the PROMPTFACTOR modularisation technique in grey.

This example shows that this algorithm can create large ontology fragments, this is because it traverses existential relationships recursively without limit. This is illustrated in our example above where ten concepts out of the eleven total were present in the fragment; the removal of a single concept does not represent a significant reduction compared to a full ontology. In Noy and Musen (2003) it is explained that the approach is not designed to guarantee how small or focused the fragment is, and that it is primarily a utility to aid common tasks for ontology engineers. Therefore, the large fragments that this approach generates do not meet our requirement of learning fragments while reducing the overhead cost (see Requirement 2a in Section 1.3.2), and we cannot use this approach in our work.

The modularisation technique presented by Grau et al. (2007) aims to create ontology fragments of reduced size based on a set of signature (S) concepts to be incorporated into the fragment. An axiom from the union of inferred axioms and explicitly stated axioms in an ontology is determined to have "syntactic-locality" with the signature, and

therefore included in the fragment, when it contains a concept from the signature. The extracted module contains two types of axioms:

1. Axioms that explicitly reference a concept from the signature;
2. Axioms which were used to infer implicit facts about concepts from the signature, but may not directly reference a concept from the signature.

Given our rescue worker example, we use this approach to generate a fragment based on the `rescue worker` concept from our example ontology (see Figure 2.8). This approach first selects the target concept `rescue worker`. Next it selects axioms that explicitly reference this concept. In this case, they are the subclass and superclass relationships, and the `rescues` relationship. Thus the algorithm selects the superclass concept `people`, the subclass concepts `fire officer` and `police officer`, and the related concept `civilian`, which is related by the `rescues` relationship. Next, the algorithm infers axioms that implicitly reference the target concept. In this case, because the concept `people` relates to the concept `building` through the `canOccupy` relationship, the algorithm infers that all of the subclasses of `people` implicitly hold this relationship too. This means that through inference, the `canOccupy` relationship and the concept `building` are also selected. The selected concepts are illustrated in Figure 2.10.

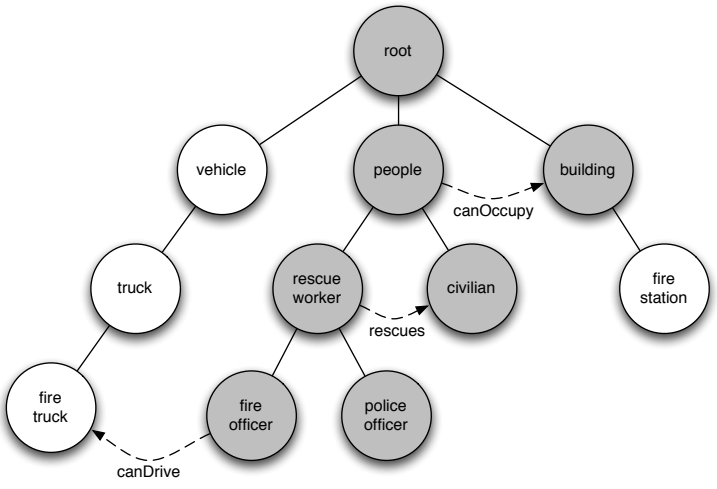


FIGURE 2.10: An example ontology based on the fire rescue domain, showing the concepts selected by the syntactic-locality modularisation technique in grey.

This approach produces smaller fragments than the approach of Noy and Musen, because instead of traversing the structure of the ontology, this approach uses syntactic-locality as a basis for inclusion of an axiom. However, this technique requires that a reasoner performs inferences over the whole ontology and as discussed in Section 2.1.3 the time taken to reason over an ontology is dependant on the constructs used in an ontology and thus can vary greatly from ontology to ontology. We require that our agents can

receive fragments quickly so that they can make fast decisions, therefore because this approach does not guarantee how quickly the fragments can be generated we will not be using this approach.

In addition to the above approaches, Seidenberg and Rector (2006) also endeavour to extract a fragment from an ontology which expresses all of the relevant axioms related to a specific concept. Their paper details a basic segmentation process, and then describes a more advanced technique. In particular, the basic segmentation process requires a target concept which the segment will represent. This process starts with the target concept and traverses the ontology's concept hierarchy upwards all the way to the root class. It then traverses downwards to the targets' leaf classes. Additionally, any links across the hierarchy from any of the traversed classes are followed upwards but not downwards. Once there are no more concepts to traverse, the concepts that were traversed form a segment representing the target concept (see Figure 2.11). Given our rescue worker example, we create a fragment based on `rescue worker` from our example ontology (see Figure 2.8). This approach first selects the concept `rescue worker`, its subclasses: `fire officer` and `police officer`, and its superclass: `people`. It then selects the concept `civilian` because it is related by the `rescues` relationship. The selected concepts are illustrated in Figure 2.12.

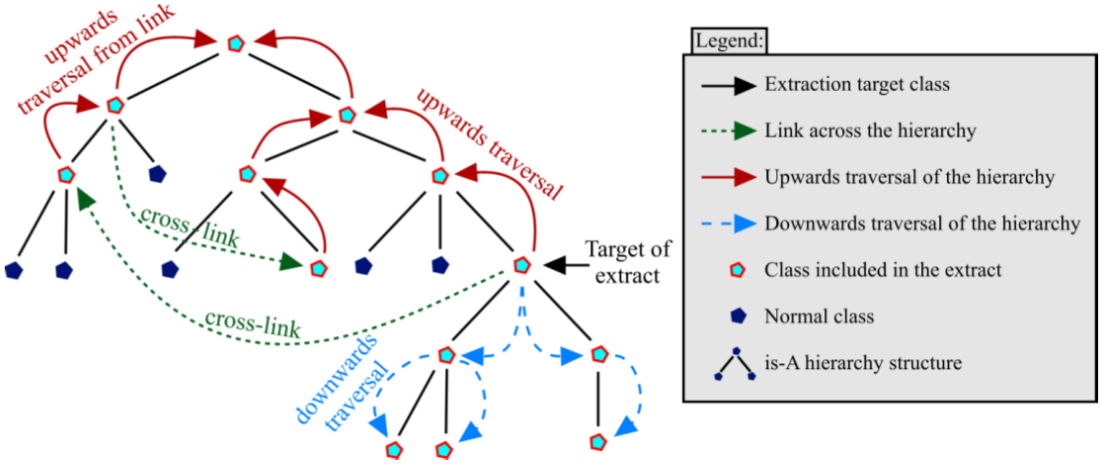


FIGURE 2.11: The basic segmentation algorithm, taken from Seidenberg and Rector (2006).

Using their advanced segmentation technique and the medical ontology GALEN, Seidenberg and Rector create a segment based on the concept 'Heart', which reduces the number of axioms by 26% compared with the complete ontology. Their results also show that on average, generated segments are reduced by 20% against the complete ontology. When taken together, this research highlights that agents with an ontological background need to be selective when learning concepts, otherwise their knowledge base will not be able to infer knowledge in real-time. This approach enables agents to determine a segment relating to a specific concept, which can be used to partially

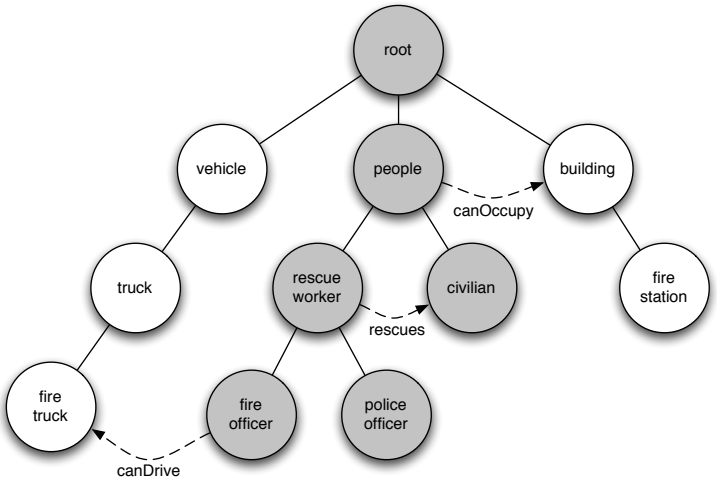


FIGURE 2.12: An example ontology based on the fire rescue domain, where the concepts selected by Seidenberg and Rector's modularisation technique are grey.

address Requirement 1 (see Section 1.3.1) by providing an automated technique that can be used to facilitate knowledge exchange. Moreover, this supports the assumption made in Section 1.2 where an agent does not require another agent's complete ontology to successfully perform a task.

We note that Seidenberg and Rector's work provides a technique that can produce an ontology fragment containing concepts and relationships about a specified concept. It is more selective than Noy and Musen's approach and therefore produces smaller fragments. Unlike the approach presented by Grau et al., Seidenberg and Rector's does not require a reasoner which incurs additional costs associated with extracting a fragment (these include memory and time). However, generating a fragment of an ontology based on a concept is useful, because we assume that an agent does not require the entire vocabulary of another agent. Seidenberg and Rector's paper provides a basic algorithm to create a fragment of an ontology, this basic technique can be used to provide a fragment that contains more context than their advanced technique and fewer concepts than Noy and Musen's approach, thus supporting Requirement 2a (see Section 1.3.2). While their advanced technique provides a smaller fragment than the basic technique, we require the additional context to compare axioms contained in our agent's ontology, so that we can determine the domain of the fragment. This is because the more advanced technique is better suited to large ontologies, such as the GALEN and Gene ontologies, however, our proposed agents' ontologies will be relatively small and therefore Seidenberg and Rector's advanced technique would produce a small fragment that would not provide sufficient context from an agent's ontology. Thus, we will use Seidenberg and Rector's algorithm to provide our agents with concepts and relationships to select from when choosing concepts to learn. We use this basic algorithm as an enabling technique which is not the focus of our research: our focus is to create an algorithm that enables agents to augment concepts that have a high likelihood of future use and that are useful in a

current task in order to reduce the costs associated with acquiring knowledge. Seidenberg and Rector's approach supports our requirements because we can specify the domain of a fragment using a concept on demand and it does not require complex analysis in terms of the time and memory resources required. This technique sufficiently meets our requirements, therefore we progress our literature review by exploring algorithms which have been designed specifically to augment an agent's ontology.

2.3 Evolving Agents' Ontologies using Online Algorithms

There has been an increase in the use of agents, which automatically evolve their own ontologies. In order for an agent to automatically evolve its ontology they can implement online algorithms (van Diggelen et al., 2004; Doherty et al., 2004; Bailin and Truszkowski, 2002). Specifically, online algorithms are designed to handle problems which have incomplete information at the beginning of the problem. In particular, it focuses on a common scenario where the input is serialised, and upon receiving a piece of the input, the algorithm takes an irreversible action without the knowledge of future inputs (Borodin and El-Yaniv, 1998). Once an action is taken, it either cannot be undone, or the undoing will incur a cost which adversely affects the overall performance of the algorithm. This type of algorithm may be forced into making decisions that may later turn out not to be optimal. Our scenario is designed to model real-life situations where future information is unknown (from Requirement 1 in Section 1.3.1), and therefore we must use online algorithms in order to determine our learning strategy. However, in Winoto et al. (2002) they evaluate their work by comparing a benchmark approach that simulates the best possible performance. The benchmark approach achieves this by having "perfect foresight", which means it has access to all future states and outcomes, and that therefore it can make decisions that maximise achieving the best outcome. In Requirement 1 (see Section 1.3.1), we specify that our approaches should be evaluated against other approaches and using an approach which has perfect foresight should provide an optimal solution. Therefore, we will use an approach that has "perfect foresight" as a benchmark against our proposed online ontology evolution algorithms (see Requirement 3 see Section 1.3.3), which aim to provide an optimal solution without "perfect foresight". Our Requirement 2 (see Section 1.3.2) states that our algorithms should reduce overhead costs, and therefore in this section, we investigate how related work has previously approached this requirement.

In Section 2.3.1, we discuss work that enables an agent to evolve its ontology online, where it is not assumed that there is a common vocabulary shared with the communicating agents. We then continue in Section 2.3.2 to introduce techniques that adapt an agent's evolution to percepts, so that we can determine when agents should collaborate and learn new concepts.

2.3.1 Agent Learning Algorithms

Several online algorithms have been developed to enable an agent to augment its ontology with additional concepts from other agents' ontologies. This is because there are many real time or simulated real time systems which require a technique that can handle serial inputs which are initially unknown. Different algorithms can be classified into three main types:

1. **Teaching agents**, whereby a specialist agent shares part of its ontology with a task agent;
2. **Mediation**, whereby an agent uses a mediator to provide translations between two concepts;
3. **Training agents**, whereby a specialist agent shares positive and negative examples of entities contained in its A-Box of the requested concept.

First, we consider **teaching agents**. Most teaching agents require a common vocabulary such as the approaches of van Diggelen et al. (2004), Van Eijk et al. (2001), and Yu and Singh (2002), which discuss strategies for augmenting concepts into ontologies through sharing concept definitions. The approaches presented by van Diggelen et al. and Van Eijk et al. augment their ontologies with one concept at a time, while Yu and Singh's approach augments its ontology with concepts from multiple agents that match a user's query. None of these approaches consider how to optimise learning new concepts into their ontologies, and more importantly rely on sharing a common vocabulary. We require that our algorithm supports agents with different interest domains represented in their ontologies and thus may not share a common vocabulary (see Requirement 2, in Section 1.3.2). Therefore, we will not discuss their approaches because they are dependent on their agents sharing a set of concepts.

The approach of Doherty et al. (2004) provides an algorithm that enables an agent to be taught the definitions of concepts held by other agents' ontologies. Their algorithm enables the teaching agent to share conditions that describe the semantics of a concept. These conditions contain information about a concept and describe its use by specifying properties, such as existential restrictions, universal value restrictions, and cardinality restrictions (discussed in Section 2.1.1). A condition can either be classified as: sufficient where the condition can be used to identify an entity in an agent's knowledge base that does not always hold true, or necessary where the condition always holds true. For example, "today is the first of April" is a necessary and sufficient condition for "today is the first day of the month". The agent receiving these conditions compares the sufficient or necessary conditions to the sufficient and necessary conditions defined for the concepts in its ontology. Furthermore, the agents created by Doherty et al. seek to augment their ontology when they encounter a concept, c_j, in a message that is not contained in their

ontology, $c_j \notin o_i$. It is assumed that agents model the same domain, and can experience events that belong to a finite set. Additionally, it is assumed that the agents' ontologies, o_i and o_j, are interconnected, and if merged would create a domain specific combined ontology, go, where $go = o_i \cup o_j$. The following list details the teaching process:

1. Agent a_i identifies a concept that it does not understand (c_j);
2. Agent a_i generates an approximation to c_j by comparing the conditions contained in c_i and c_j. If the necessary conditions of c_i and c_j are the same, and contain a subset of related sufficient conditions, they are identified as being approximately equivalent, and are incorporated into the agent's ontologies as equivalent;
3. Agent a_i uses the closest approximation to c_j contained in its knowledge base (c_i), and asks another agent to clarify the properties of c_j.

This approach focuses on comparing the conditions which describe a concept in order to identify whether an agent's ontology already contains a similar concept. This technique also requires agents to contain accurate information, because it is reliant on the definition of conceptual knowledge contained within the ontology. For example, the concepts `table` and `stool` have conditions that specify that they have four legs and a platform. The difference between the `stool` and `table` is their size and height, if these properties are undefined then they may become incorrectly equivalent using this technique. This approach also requires that the agents model the same domain in their ontology. If the agents do not model the same domain then the likelihood that concepts which are falsely found to be equivalent increases, as described in the example above. It is also possible that the agents may not be able to teach each other new concepts because they are unlikely to share concepts with the same conditions, or even define the conditions with the same terminology. Thus using this algorithm would predicate that both agents would share similar logical frameworks, such that if axioms are shared, then both agents would be able to reason over these axioms and produce the same entailments. These problems would not enable an agent to expand its vocabulary to its full potential. This approach therefore does not fully support Requirement 2a (see Section 1.3.2), and so we will not adopt this approach in this work.

Second, we consider **mediation agents**. This is a type of approach that addresses the above weakness by considering the semantics of a concept by using thesauri and alignment services (Bailin and Truszkowski, 2002; Reed and Lenat, 2002; Grenon et al., 2004). While the use of thesauri facilitates obtaining the synonyms of terminology, they are limited in their ability to align two semantically equivalent concepts. In particular, alignments determined by thesauri are limited to synonymy only, and do not take the context of the lexicons into consideration. Additionally, thesauri are also prone to the problem of ambiguity; a symbol can correspond to multiple meanings (polysemy), and therefore an alignment that has a similar meaning but is not equivalent may be

obtained. However, the use of thesauri can help to relate semantically similar concepts. To this end, the work of Laera et al. (2006, 2007) and Trojahn et al. (2008) focuses on a negotiation framework where agents can obtain mappings between concepts from such a mediator, and their agents deliberate in order to determine which vocabulary to use. Here, however, we do not focus on their negotiation processes because they do not address our research requirements of expanding our agent's vocabulary to diversify communication and inference of logical consequences (see Requirement 2a, in Section 1.3.2).

Bailin and Truszkowski (2002) propose a technique that allows agents to request documents by specific concepts used to classify them. In particular, these agents have the ability to label documents with additional concepts for classification. The authors recognise that natural language semantics can be exploited to assist in the generation of alignments between linguistically similar concepts, and therefore they explored the use of WordNet (Miller, 1995) as a thesaurus to aid understanding between communicating agents. WordNet, is a semantic lexicon that models relationships between words and it contains a rich structure (known as 'synsets') of linguistic relationships between nouns, verbs, adverbs, etc., including synonym relationships. Given that Bailin and Truszkowski consider the situation where agents a_i and a_j describe a document d with concepts c_i and c_j respectively; a_i aims to determine if c_j should also be used to describe d. Specifically, they propose a four-phase communication protocol to facilitate communication between two agents:

1. Agent a_i sends a set of concepts C that describe d, to a_j. Agent a_j checks that its own ontology (o_j) contains $c_i \in C$. When c is not contained in o_j then a_i queries WordNet for a synonym syn, and checks if o_j contains syn. If syn is contained in o_j then it can be used to represent an interpretation of the keyword. Finally, a_j requests a confirmation from a_i, that syn is an interpretation of c_i;

2. If a_j can interpret most of $c \in C$, then it can request a clarification from a_i. This clarification includes the following techniques:

   (a) Locate synonyms in the source agent's ontology;

   (b) Provide a set of specialisations or instances from the source's ontology;

   (c) Provide a weak generalisation from the source's ontology;

   (d) And provide a definition in formal logic.[6]

3. **Relevance analysis** is used to evaluate c_i's match to d, (details of the evaluation process can be found in Bailin and Truszkowski (2002)). The outcome of the evaluation decides if c_i relates to d;

[6] Taken from Bailin and Truszkowski (2002).

4. If a_i interprets that c_i relates to d, a_i will augment the ontology to include c_i so that $c_i \in o_i$.

This approach differs from the others discussed so far, in that it exploits a linguistic resource (i.e. WordNet) to approximate semantically similar terms. Although the semantic analysis gained from WordNet is limited, synonyms contained within the linguistic ontology can be loosely related to the keyword. However, this approach provides no guarantee of semantic similarity, although some algorithms have since been developed to determine the similarity between terms found in WordNet[7]. This approach attempts to identify semantically similar concepts between agents that have different ontologies modelling the same domain, thus enabling them to incorporate domain related concepts and their relationships. The work on ontology alignment (Euzenat and Shvaiko, 2007) attempts to expand Bailin and Truszkowski's aim, of providing a resource to enable concept matching, by exploring techniques that compare the semantic and lexical similarity of concepts contained in different ontologies. In order to aid fulfilling Requirement 2a in Section 1.3.2, our learning approach has to optimise the costs associated with learning and acquisition of concepts. None of the approaches used here consider optimising costs, and none of them consider learning more than one concept at a time to reduce acquisition costs. Therefore we will not be using these techniques in our approach.

Finally, we consider **training agents**. In this context, the approach of Wiesman and Roos (2004) enables agents to incorporate additional conceptual knowledge into their ontologies, and aims to map two concepts: c_i and c_j. The following describes how an agent augments its ontology:

1. An agent a_i desires to incorporate c_j;

2. Agent a_i sends a request to agent a_j for a set of instances I that represent c_j, where $i_i \ldots i_n \in I$ (where n is the number of instances in the set I), and the descriptions of any properties and restrictions associated with c_j;

3. Then agent a_i classifies each instance of I with its existing ontology and the received properties and restrictions. These classifications are then used to determine the mapping and corresponding relationships (such as equivalence, subsumes or disjoint) to the other concepts in o_i. In this context, the ratio of successful and unsuccessful matches from the classification are used to determine which concept in the agent's ontology has the closest match to c_j;

4. Consequently, if all I are identified as being a subset of c_i, a_i will incorporate c_j into its knowledge base as equivalent to c_i.

[7]The WordNet::Similarity Perl module: `http://search.cpan.org/dist/WordNet-Similarity/`.

As can be seen, this technique focuses on augmenting an agent's ontology, it does not facilitate the necessary reasoning required to solve domain related queries, or enable agents with the same domain represented with different models to align semantically equivalent terms. Additionally, the mapping protocol used to identify the relationship between two concepts is dependent on the agent's domain of interest and on the communicating agent's ontological knowledge. Thus, agents that do not have intersecting knowledge will be unable to correctly classify the instances and will require a prerequisite of intersecting domain knowledge. This technique also requires agents to contain accurate information, because it is reliant on the definition of conceptual knowledge contained within the ontology. As with Doherty et al.'s approach it is possible that a `table` and `stool` can be defined with similar atomic components, and if their `size` is undefined then `table` and `stool` may become incorrectly equivalent using thisf technique. Introducing incorrect equivalences into an ontology will cause agents to make incorrect decisions, which does not support an agent completing domain tasks. Since Requirement 2a (see Section 1.3.2) requires new domain tasks to be completed, this technique is not suitable for use in our approach.

Another example of such an approach, is that of Afsharchi et al. (2006) who also use instances to define a concept. In more detail, their methodology is based on use of positive and negative examples of a class to enable an agent to decide to augment its ontology. Specifically:

1. Agent a_i contacts all agents in A, with the aim of identifying c_i, when $c_i \notin o_i$;
2. Agents that contain c_i in their knowledge base respond by sending the strongest positive and negative examples of c_i. These are determined by the responding agent (a_j's) knowledge base and c_i's properties and restrictions contained in o_j. The positive example contains all of the associated features of c_i, and the negative example of c_i is far removed from this feature set;
3. If there is a conflict between the set of positive or negative examples, a_i asks the agents that responded, to classify the conflicting example. The majority ruling of this set of agents determines if a_i incorporates the features from the conflicting example.

This approach allows the agents to collaboratively determine the vocabulary; it is assumed that if a_i can classify the same individuals that are representative of c_i, that a_i has successfully incorporated c_i. This technique focuses on incorporating knowledge to develop a common vocabulary, and is not designed to facilitate the necessary reasoning required to solve domain related queries or enable agents with the same domain represented with different models to align semantically equivalent terms. Although, it does not fulfil the required aims, it does provide a consensual technique to align the communication vocabularies; however, this approach has a weakness, with a small number of

collaborating agents there are few training examples provided, therefore its classification of c_i is not as comprehensive compared with concepts that are described with a large number of collaborators. These two training approaches do fully support Requirement 2 (see Section 1.3.2), because these techniques require a set of agents that contain the same vocabulary for a specific concept and that contain positive and negative examples of the required concept. In our scenario, we consider that our agents are heterogeneous and will therefore use different ontologies; these techniques are more accurate with a greater number of sample set of examples.

	Requires common vocabulary	**Supports heterogeneous domains**	**Learns more than one concept**	**Considers optimising costs**
Author of Teaching Technique				
van Diggelen et al. (2004)	✔	✗	✗	✗
Van Eijk et al. (2001)	✔	✗	✗	✗
Doherty et al. (2004)	✔	✗	✗	✗
Author of Mediation Technique				
Bailin and Truszkowski (2002)	✔, WordNet	✔	✗	✗
Laera et al. (2006)	✗	✔	✗	✗
Laera et al. (2007)	✗	✔	✗	✗
Trojahn et al. (2008)	✗	✔	✗	✗
Author of Training Technique				
Wiesman and Roos (2004)	✔	✔	✗	✗
Afsharchi et al. (2006)	✔	✔	✗	✗

TABLE 2.14: A comparison of agents which use learning approaches to evolve their ontologies.

To summarise, the above approaches used to augment an agent's ontology provide a variety of algorithms. In order to evaluate these algorithms, we compare them in Table 2.14. In more detail, all of the teaching techniques require a common vocabulary because the 'teacher' uses it to describe concepts to other agents. We require that an online algorithm be able to learn new concepts regardless of a shared vocabulary so that the agent can apply new knowledge to complete tasks that were not foreseen as a requirement during the design of the ontology. All of the mediation and training algorithms enable an agent to learn about new concepts and support heterogeneous domains because they do not require a common vocabulary, with the exception to Bailin and Truszkowski which uses WordNet, a vast library which does not limit an agent's vocabulary. The focus of our work is to develop an algorithm that considers optimising costs associated with learning, including acquiring, hosting, managing and using. None of the above approaches consider optimising costs, and none of them consider learning more than one concept at a time to reduce the acquisition costs. Therefore, we will not be using these approaches in our work. Nevertheless, the approaches described in this section provide a benchmark approach of augmenting an agent's ontology with a single concept and its

definition per augmentation.

2.3.2 Adaptive Agent Learning Approaches

In contrast to the previous section where we present approaches that always decide to augment concepts, this section focuses on approaches that allow their agents to decide whether to augment their ontologies based on their percepts and goals.

We evaluate three adaptive techniques, that consider:

1. When an agent should increase its vocabulary;
2. When an agent should collaborate with other agents;
3. When to broadcast new services provided by agents, as a result of service requests from users;
4. How an agent can describe a common experience with the use of a shared vocabulary.

First, Soh and Chen (2005) propose an approach that enables agents to perform two tasks: first, collaborate on tasks with other agents (t_c); and second, augment their ontology with logical entailments (t_l). Only one of these tasks can be performed at once, and the agent decides which type of task to perform based on a collaboration utility (see Equation 2.1).

$$\begin{aligned} \textbf{Collaboration Utility} = (&\mathit{successRate} + \mathit{helpRate} + \\ &\mathit{requestToRate} + \mathit{requestFromRate} + \\ &(1 - \mathit{nowCollaborating})) \,/\, 5. \end{aligned} \tag{2.1}$$

The collaboration utility is used to enable an agent to decide whether to collaborate with another agent. Specifically, the collaboration utility equally weights the percentage of five measures: successfully answered queries and is denoted by *successRate*; collaboration it has participated in and is denoted by *helpRate*; the percentage of times it is asked to rate a document and is denoted by *requestToRate*; the percentage of requests from the agent that it collaborated with and is denoted by *requestFromRate*; and whether it is currently participating in a collaboration denoted by the boolean *nowCollaborating* which has the range of 0 and 1. Once an agent's collaboration utility falls below a threshold then that agent is required to expand its knowledge base so that it fulfils its utility threshold. Specifically, collaborating with other agents yields a higher utility to

the agent than augmenting its ontology. Their agent architecture considers the rewards of short and long-term learning, and its desired level of performance. The logical inference tasks improve an agent's knowledge base by providing ontological mappings to concepts contained in its ontology, and increases the possibility of future collaborations, which ultimately improves its collaboration utility. However, collaboration requests increase the collaboration utility immediately, using the existing knowledge contained in the ontology. Given this, Soh and Chen use a desired level of performance threshold, to determine the type of task to perform, t_c or t_l, to respond according to t and the utility gained. This threshold enables a user to specify different preferences if speed is more important than accuracy. For an agent to make an informed choice about which agents to collaborate with, in the environment, it stores a profile in a 'neighbour database' of agents it has interacted with, and contains the neighbour's relationship utility with other agents. Soh and Chen's system uses agents to retrieve documents, and their knowledge of ontological concepts describes these documents. A user will query a_i for documents, if it does not contain enough references to documents that match the query c_i, it will collaborate with other agents to retrieve relevant documents. Depending on a_i's state it can react with the following approaches:

1. If a_i knows that its neighbours have documents related to the concept c_i, it sources a specified ratio of documents from each neighbour depending on their collaboration utility, see Equation 2.1, and credibility score[8], to retrieve a sample set of documents related to c_i;

2. If a_i does not contain a reference to c_i, it uses a translation table to relate c_i to concepts contained in neighbouring agents;

3. If a_i believes that none of the neighbouring agents' ontologies contain documents referred to by c_i, it will broadcast a request for documents referenced with c_i to all agents, A_{util}, prioritised by their collaboration utility, followed by A_{util}'s neighbouring agents with the highest collaboration utility. This technique allows the agent to discover more agents to collaborate with in the environment.

This approach uses ontologies with a single level of hierarchy, and the complexity of computations required for OWL DL, is not considered. This methodology enables agent a_i to select which agents to collaborate with, and it also allows a_i to initially consult agents that are known to be knowledgeable and or helpful, with the use of utility scores. This enables an agent to choose which collaborations, t_c and t_l, it will potentially benefit from the most. The ability to decide which collaborations to prioritise is beneficial in reducing the number of 'non-useful' collaborations, in terms of achieving the desired utility and therefore reducing the number of total messages sent. This approach does

[8]The credibility of a translation between two concept names is the average relevance between the two sets of associated documents.

not address Requirement 2a (see Section 1.3.2) because it does not enable agents to learn concepts from other agents' ontologies. Instead it uses a single-level hierarchy of concepts as a way to index documents with agents. As such we do not use this technique in our approach.

Second, Sensoy and Yolum's (2008) approach facilitates the visibility of new knowledge by broadcasting newly available services. Their framework consists of agents that provide services to users; the users and the agents both have access to a common ontology that contains properties that can be used to describe services. A subset of these properties defines a service, and hence a user can only request a service that is a subset of these properties. In Sensoy and Yolum's agent environment, a service is unique and can only be provided by one agent. When a_i receives a request for a new service (one that is not provided by any other agents) it generates a new service with the specified properties. Newly generated services are determined by the users, and created on-demand; for example, the purchasing of a book and delivery of that book are two components contained in the service ontology, an agent would be able to generate a new service to provide these two components. Once agent a_i's service x has been generated, a_i broadcasts x and x's property components to all other agents in the environment. By doing so, this eliminates the generation of equivalent services with different names. This approach broadcasts new knowledge to agents in an environment, augmenting their ontologies through the creation of new services, given that they share a common vocabulary. Similarly, it is possible to enable broadcasting of services outside of a controlled environment, using a distributed knowledge discovery framework. For example, the OpenKnowledge system (Siebes et al., 2007) describes a "discovery and team formation service" (DTS) that uses a peer-to-peer storage and retrieval system to allow agents and services to discover potential interactions. In such a system, agents can subscribe to keywords or ontological concepts, and when agents and services that are related to them become available, the framework co-ordinates an interaction, based on an interaction model (IM). An agent proactively searching for relevant collaborations to increase its utility would therefore be represented by an IM that described a query/response mechanism, which would be orchestrated automatically as new services come online. This research highlights the need for agents to advertise new ontological knowledge. In a system where agents are rated by their ability to collaborate, this would contribute to a positive utility as it promotes future collaboration. This approach could enable fire brigades to specialise in fighting fire with specific parameters, such as chemicals. Therefore, the agent could allocate specialist agents to different fire sites. Allocation problems however are not the focus of this work, but evolving ontologies efficiently is, therefore we will not be using advertising for our approach.

Third, Soh (2002) also investigates the use of utility to determine an agent's behaviour. Specifically, they use a utility to decide which concepts to use to describe an experience. In their example, agent a_i maintains a dictionary of its experiences (d_i) and a translation

table (t_i). The experiences are a finite set, and can be described by their properties, which are common to all agents. Specifically, agent a_i's knowledge is contained in a distinct set containing an ontology o_i, a dictionary of experiences d_i, and a translation table t_i and can be described with the tuple $grounded(a_i, o_i, d_i, t_i)$. In particular, d_i is used to compare relationships between c_i and c_j, and t_i contains the translations between c_i and c_j, and is used when a_i communicates with a_j. In addition to these features, a description vector is stored in t_i, which is a tally of agents that a_i has encountered, which use c_j to describe the same experience, and is used to calculate the percentage of agents using c_j to relate to c_i. Given this, agent a_i aims to evolve o_i so that it can refer to an experience, e, with the same concept as the majority of agents with which it communicates. In particular, e is described with properties involved in an experience, thus guaranteeing agents are talking about the same e. In order to achieve this, a_i transfers knowledge (c_j), from the t_i to o_i, so that $c_j \in o_i \wedge c_j \notin t_i$. The interaction between a_i and a_j is defined with the following steps:

1. Agent a_i requests an alternative concept for classifying c_i from a_j;
2. Agent a_j sends c_j to a_i, which refers to c_i;
3. Then agent a_i calculates a belief and plausibility rating of c_j (formulae are detailed in Soh (2002)). The closer the belief and plausibility values, the more credible the proposition. If c_j is credible then it is moved from t_i to o_i, so that $c_j \in o_i \wedge c_j \notin t_i$.

This method relies on agents having common experiences (experiences which are described with the same set of properties), which are referred to by properties of the experience. Therefore this approach is unsuitable for agents that do not share experiences, due to having different interest domains and performing different tasks, and consequently their ontologies do not contain knowledge about common experiences. Moreover, this technique focuses on augmenting an agent's knowledge about a common task, and does not provide any techniques to enable agents to communicate across different domains, as required by Requirement 2a (see Section 1.3.2), because agents with different domains will have different experiences.

Author of Teaching Technique	**Requires common vocabulary**	**Supports heterogeneous domains**	**Learns more than one concept**	**Considers optimising costs**
Soh and Chen (2005)	✗	✓	✗	✓
Sensoy and Yolum (2008)	✗	✓	✗	✓
Soh (2002)	✓	✗	✗	✗

TABLE 2.15: A comparison of agents which use adaptive learning approaches to evolve their ontologies.

To summarise, the above approaches differ from those in Section 2.3.1 because agents adapt their algorithm based on their observations. In order to evaluate these algorithms, we compare them in Table 2.15. In more detail, Soh and Chen and Sensoy and Yolum's approaches do not require a common vocabulary, whereas Soh's approach does. We require that our algorithm is able to learn new concepts from agents with different domains so that they can benefit from new knowledge that may not have been recognised as domain specific during the ontology's design time. Both Soh and Chen, and Sensoy and Yolum's approaches consider optimisation in their agent's algorithm, however they do not consider the costs associated with acquisition or whether to learn more than one concept at a time to reduce such costs. In conclusion, we will not be using the above techniques in our approach, although we will consider the use of utilities to allow an agent to make an informed choice about its actions. For example, network speeds will affect the cost of acquiring fragments, and it may be advantageous to learn concepts when it is not expensive to do so, in order to reduce acquisition costs. To this end, we consider the use of utilities to improve the efficiency of our approach, which addresses Requirement 2 (see Section 1.3.2).

2.4 Predictive Models to Aid Ontology Evolution

Predictive models can be used to aid ontology evolution because they can be used to predict which information will be required to complete a task based on past events that had the same or similar parameters. For example, a fire brigade requires information on which vehicles can move rubble, and previously found that an instance of the concept `mountedForkLiftTruck` was used to successfully remove rubble. However, the fire brigade removed the concept `mountedForkLiftTruck` from its ontology because it is rare for a fire brigade to be required to remove rubble. Therefore, the fire brigade can predict that because an instance of the concept `mountedForkLiftTruck` was useful to remove rubble before, it should be able to use an instance of the `mountedForkLift-Truck` to achieve the same outcome. The fire brigade requests information about the `mountedForkLiftTruck`, without this prediction the fire brigade would have requested information about vehicles that can remove rubble which returns many types of vehicles. Thus, using prediction can reduce the amount of information requested, thus lowering the acquisition cost. Furthermore it also reduces the number of concepts augmented into the fire brigade's ontology, thus lowering the ontology's potential size and the associated costs to use it. This prediction model must be able to evolve so that it incorporates the outcome of all the past concepts required for tasks. This enables the prediction model to handle the evolution of the vocabulary used in a scenario.

In order to fulfil Requirement 2b presented in Section 1.3.2, which requires that agents can predict the knowledge they require given their state, we consider the use of Bayesian probability. We will use probability to model uncertainty, as opposed to alternative

methods such as entropy, because the use of probabilities enables us to implement the agent Markov models using off-the-shelf optimised algorithms with little overhead. Therefore, we meet our Requirement 2, which states that algorithms must provide low overhead. We select this interpretation of probability because it enables an agent to represent its degree of belief in a statement, and is based on prior probability which is updated with new and relevant data. This is in contrast to Frequency Probability which requires dealing with experiments that are random and well defined. Our experiments are not random, because tasks are inter-linked. For example, if a building contains a chemical that is strongly reactive to water, it is not random that a fire brigade is required to use a dry suppressant to minimise oxidation. In the next section, we introduce Bayes' theorem to ground the use of Bayesian networks and Markov models. This is followed by an introduction to the use of probabilistic ontologies to enable prediction.

2.4.1 Bayes' Theorem

In more detail, Bayes' theorem (Jensen, 1997) shows the relation of the probability of a hypothesis given observed evidence and the probability of that evidence given the hypothesis, shown in Equation 2.2.

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \tag{2.2}$$

where H is a hypothesis, and D is the data, $P(H)$ is the prior probability of H: the probability that H is correct before the data D was seen, $P(D|H)$ is the conditional probability of seeing the data D given that the hypothesis H is true, $P(D|H)$ is the likelihood, $P(D)$ is the probability of D, and $P(H|D)$ is the posterior probability: the probability that the hypothesis is true, given the data and the previous state of belief about the hypothesis.

In our domain of search and rescue, the agent tries to determine the probability of a task occurring in the future, given the tasks that it has already seen. For example: $P(H)$ is the probability that the next task for an ambulance to deal with is a symptom of `Bronchitis`, which is 0.101; $P(D)$ is the probability of a casualty being a `Fire Casualty`, which is 0.9; $P(D|H)$ is the probability that the casualty was a `Fire Casualty` given the symptom is `Bronchitis`, which is 0.69; and $P(H|D)$ is the probability that the next task is a symptom of `Bronchitis`, given that the casualty is a `Fire Casualty`, which we can calculate using Bayes' Theorem, shown in Equation 2.3:

$$P(H|D) = \frac{0.69 \times 0.101}{0.9} = 0.077 \tag{2.3}$$

This formula can be used to build a model to represent events and the probability of

future events given prior probabilities of past events. These models can be categorised as a Bayesian Network.

2.4.2 Bayesian Networks

Bayesian networks are graphical models for reasoning under uncertainty, where the nodes represent variables and a set of arcs represent direct connections between two nodes (Jensen, 1997). The arcs are weighted with conditional probability distributions associated with each node. The only constraint on Bayesian networks is that there is not allowed to be any directed cycles; models that have directed cycles are known as directed acyclic graphs (DAG). In order to build a Bayesian network a knowledge engineer must identify the variables of interest and what values they should take. These values may include: Boolean nodes where the nodes can either have a true or false value; ordered values, for example a node `chemicalExposure` represents the level of exposure to chemicals and can take the values of {`high`, `medium`, `low`}; or integral values where nodes have a range of possible values, for example a node `chemicalExposure` represents the percentage of affected blood cells and the values can range from 0 to 100. It is important that the knowledge engineer chooses values that represent the domain efficiently, but with enough detail to perform the reasoning required. For example, an injured civilian is suffering from shortness of breath (dyspnoea) and an emergency rescue response team is worried that the civilian has Bronchitis. The response team knows that there are other diseases, such as tuberculosis, that are possible causes of dyspnoea. They also know that other relevant information includes whether or not the patient has been exposed to harmful chemicals or is a fire casualty (both increasing the chances of Bronchitis). The result of a blood test would indicate whether the patient has Bronchitis and the required form of treatment. This example is modelled by the Bayesian network shown in Figure 2.13.

The structure of a network should capture qualitative relationships between variables. In particular, two nodes should be connected directly if one affects or causes the other, with the arc indicating the direction of the effect. Once the topology of the Bayesian network is specified, the next step is to quantify the relationships between connected nodes: this is done by specifying a conditional probability distribution for each node. In order to build a conditional probability distribution we need to look at all the possible combinations of values of the parent nodes. Each combination is called an instantiation of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take each of its values. For example, consider the Bronchitis node of Figure 2.13: its parents are `fire casualty` and `chemical exposure` and take the possibility joint values $\{\langle F, T\rangle, \langle F, F\rangle, \langle T, T\rangle, \langle T, F\rangle\}$. The conditional probability table specifies in order the probability of Bronchitis for each of these cases to be: $\langle 0.05, 0.02, 0.03, 0.001\rangle$. In our example, the conditional probability tables

enable the reasoning of the likelihood that the Bronchitis is the cause of the dyspnoea experienced by the civilian.

Bayesian networks can be used to predict which concepts will be required for a task given the past events. However, this requires that all past events would need to be recorded, in order to use a Bayesian network, which is not optimal. Specifically, we know that the concepts required are not dependent on the past events, and instead only on the current state. In the above example the current state is that dyspnoea is a symptom, and that there are particular chemicals in buildings. The contents of other buildings that have been previously evaluated, or the symptoms of other civilians in other buildings do not affect the civilians being evaluated now, and as such this process is said to exhibit the Markov property (Russell and Norvig, 1995). The Markov property states that future states are only affected by the current state (as opposed to the full past states). Thus, we can avoid recording and modelling the full history, and can instead use a more efficient Markov model.

2.4.3 Markov Models

A Markov model is a specialised Bayesian Network which follows the assumption of the Markov Property (Dodge, 2006). This property states that there are no direct dependencies in the system being modelled which are not already explicitly shown via arcs in the present state. In other words, the conditional probability distribution of future states of the process depends only on the state given. In our Bronchitis case, for example, there is no way for a patient with chemical exposure to be influenced by dyspnoea except by way of the chemical exposure causing Bronchitis. There has to be a particular cause to link between chemical exposure and the symptoms of dyspnoea.

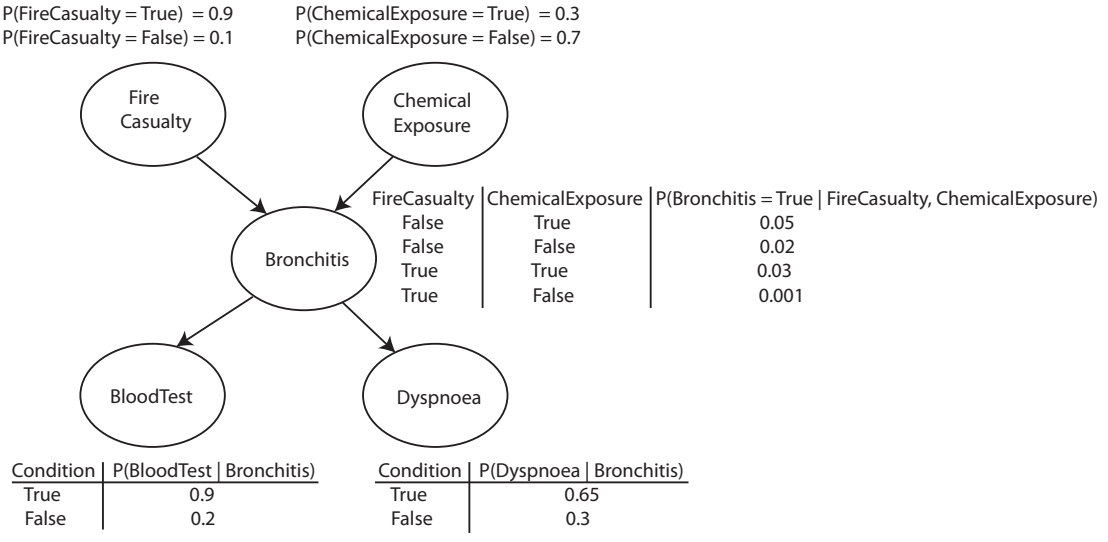


FIGURE 2.13: An example Bayesian network.

There are four different Markov models that can be used, which depend on two interlinked facets. Firstly, whether the system state is fully observable, and secondly, whether it is controlled or autonomous. In Table 2.4.3 we summarise the four Markov models, and the parameters under which they apply.

	State Fully Observable	**State Partially Observable**
System is Autonomous	Markov Chain	Hidden Markov Model
System is Controlled	Markov Decision Process	Partially Observable Markov Decision Process

TABLE 2.16: The Markov models that apply under two criteria.

In more detail, the four types of Markov model are as follows:

1. **Markov Chain.** A Markov chain models individual probabilities from each possible state to each possible next state, and is therefore the simplest type of Markov model. A sample Markov chain is illustrated in Figure 2.14. Applicable when the state is fully observable and the system is autonomous;

2. **Hidden Markov Model.** A Hidden Markov model is similar to a Markov chain, but requires an additional algorithm to compute the most-likely corresponding sequences of states. Such algorithms include the Viterbi algorithm (Forney Jr, 1973) and the Baum-Welch algorithm (Baum et al., 1970). Applicable when the state is partially observable and the system is autonomous. A sample Hidden Markov Model is illustrated in Figure 2.15, which demonstrates the probabilities of a building falling down, given unobservable states of the structural integrity of the building;

3. **Markov Decision Process.** A Markov decision process is a Markov chain with the addition of an action vector. The action vector relates to the fact that the system is controlled, and therefore acts typically to maximise some utility function. Applicable when the state is fully observable and the system is controlled. A sample Markov decision process is illustrated in Figure 2.16, which shows the differing probabilities of a building collapse depending on whether the fire was doused with water or not;

4. **Partially Observable Markov Decision Process (POMDP).** A POMDP is a Markov decision process where the state of the system cannot be observed. They therefore cannot be solved entirely, and are known to be an NP-complete problem (Shen et al., 2006), although approximation techniques can be used. Applicable when the state is partially observable and the system is controlled. A sample POMDP is illustrated in Figure 2.17, which shows that the state of the system is not observable, and therefore a belief estimator (such as any domain-specific algorithm that may be generally useful, but does not observe the current state) is used to decide an action to take, leading to an outcome with indeterminate probabilities.

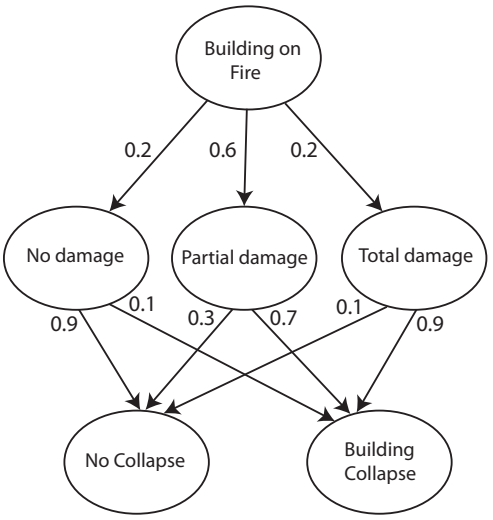


FIGURE 2.14: A Markov chain showing the probabilities of moving from one state to another.

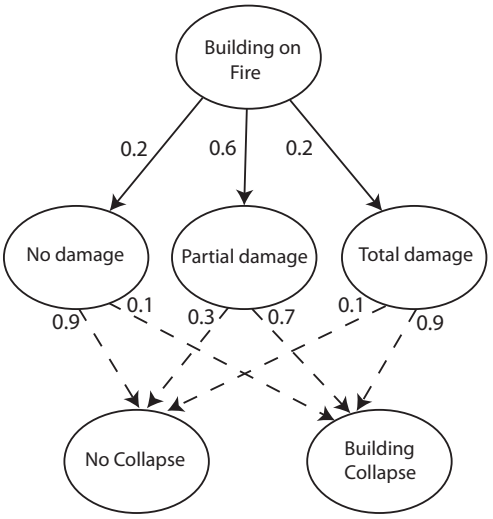


FIGURE 2.15: A Hidden Markov model showing the probabilities of moving from one state to another, where the structural integrity of the building is unobservable, as indicated by dashed lines.

The state of our system is fully observable, because the agents are updated on the state of the world each time step, and use this information as the basis of their decision logic. Our system is also autonomous, since the agents do not require or handle any human input. Thus, a *Markov chain* can be used to model relationships between the present state and the probabilities of future states. Markov chains provide the benefit that only the current state needs to be examined, and as such a Markov chain can be implemented using a finite state machine. A Markov chain allows probabilities to be modelled such that given the present state, a set of probabilities (which total 1.0) are given, one for each possible subsequent step. This chain can be used to model a system, and to make predictions on the next step of a system. Our Requirement 2c (see Section 1.3.2) requires a methodology to predict the concepts that an agent may encounter in the future, and we will therefore consider using Markov chains for this purpose.

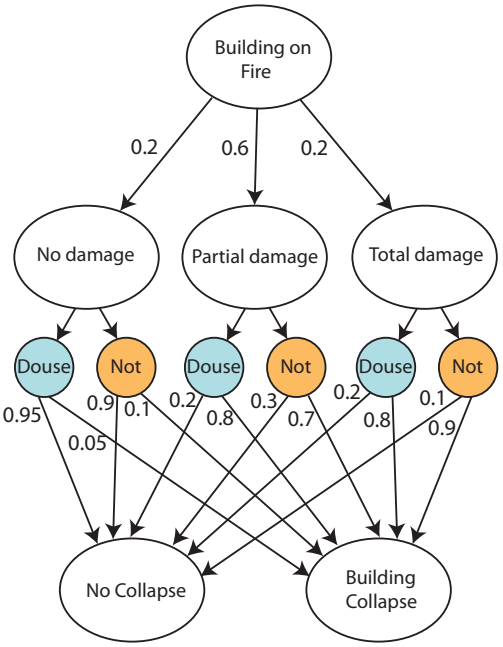


FIGURE 2.16: A Markov decision process showing the probabilities of moving from one state to another, where action vectors of "douse" and "not" indicate the different probabilities of outcomes depending on whether the fire was doused with water or not.

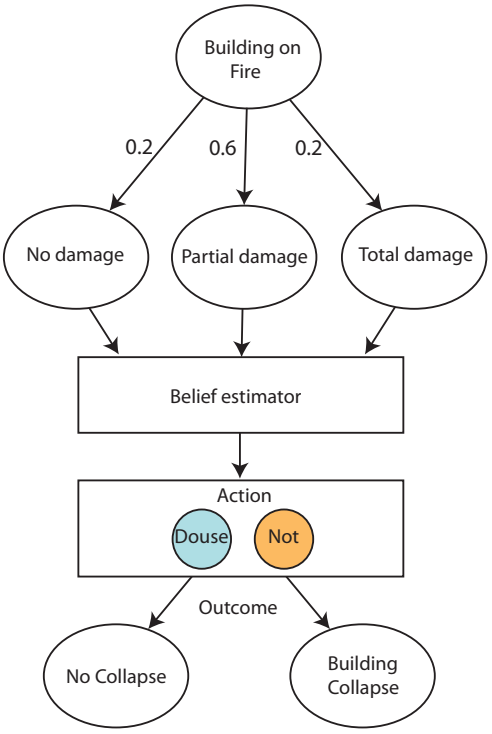


FIGURE 2.17: A Partially-Observable Markov Decision Process, showing that the probabilities of the system are not all available, and that because the state of the system cannot be observed, the model does not handle the decision of which action to take, and hands this task off to some third-party system that does not use the current state in its decision making.

There are also existing probabilistic tools to provide Bayesian and Markov processing. In the next section we will analyse approaches that use probabilistic ontologies to formally express a system in order to use existing processing tools with ontologies.

2.4.4 Using Probabilistic Ontologies for Prediction

Costa and Laskey (2006) define probabilistic ontologies as an explicit, formal representation that expresses knowledge about a domain application. This includes:

1. Types of entities that exist in the domain;
2. Properties of those entities;
3. Relationships among entities;
4. Processes and events that happen with those entities;
5. Statistical regularities that characterise a domain;
6. Inconclusive, ambiguous, incomplete, unreliable, and dissonant knowledge related to entities of the domain;
7. Uncertainty about all of the above forms of knowledge.

In this definition, items 1-3, encompass the basic ontology definition presented in Section 2.1.1. Items 4-7 build upon the definition of an ontology so that the ontology can model uncertainty over the axioms it contains. Such probabilistic ontologies are used to comprehensively describe a domain and the uncertainty associated with the knowledge in that domain, and provides a standard structure that is machine readable.

In particular, Costa and Laskey present the PR-OWL probabilistic extension to OWL ontologies. This extension enables an OWL ontology to represent complex Bayesian probabilistic models by marking up the concepts so that they extend the PR-OWL Upper Ontology. The OWL ontologies marked up with this upper level ontology can be used in conjunction with Bayesian probabilistic tools, such as Netica, Hugin, Quiddity∗Suite, and JavaBayes. These types of tools enable analysis on the concepts within the ontology so that a user can reason about the uncertainty of the ontology's knowledge. Figure 2.18 illustrates the layers required to use PR-OWL ontologies and benefit from the use of probabilistic tools. In more detail, Figure 2.18 shows that the domain OWL ontologies extend the PR-OWL Upper Ontology, both a domain ontology and the PR-OWL Upper Ontologies are required to use the Bayesian probabilistic tools.

While there are benefits to using PR-OWL, namely the ability to use specialised tools to analyse the uncertainty of the knowledge contained within a domain ontology, there are a

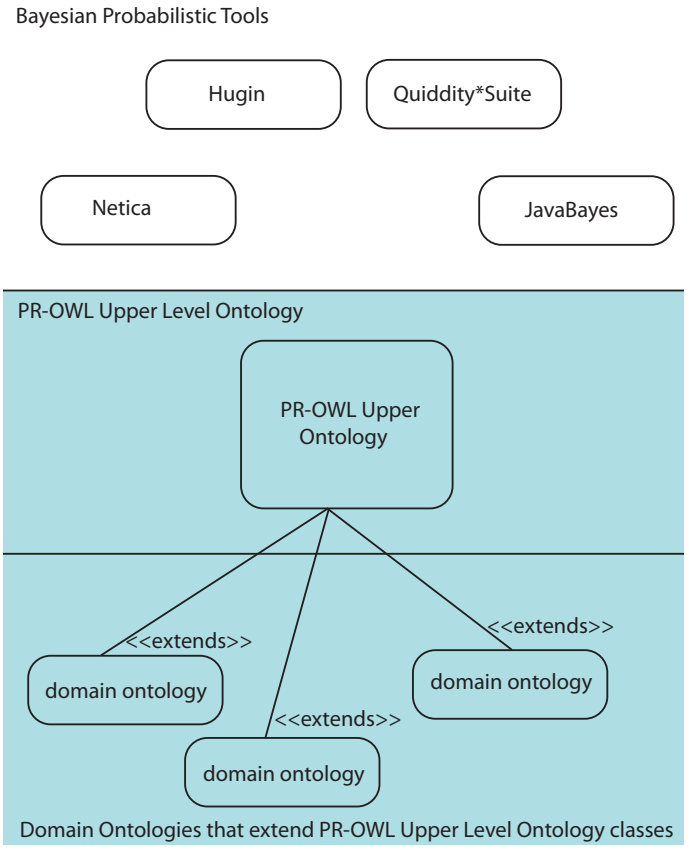


FIGURE 2.18: The required components of using PR-OWL ontologies.

lack of domain ontologies that adhere to this requirement. This may be due to PR-OWL being an under construction system which has not seen any major work since 2008, and has had no commercial uptake[9]. There is a high overhead cost of time to implement the required mark-up for new and existing ontologies. The addition of PR-OWL markup to an ontology increases its complexity, whereas in contrast, our requirements (Requirement 2b, in Section 1.3.2) are to use the optimal number of concepts to complete tasks, so that we maintain a low ontology complexity to promote fast response times. We also note that this approach provides a user with more features than we require and a simpler model would provide the same benefits without the identified overhead costs. Thus, we will not consider using PR-OWL for prediction, instead we will use a Markov chain (see Section 2.4.3).

2.5 Evaluation Techniques for Evolving Ontologies

In order to evaluate online learning and forgetting algorithms, we require measures that determine the performance of the agent, a success measure that shows the benefits of using an algorithm, and a measure that shows the complexity of an ontology. We outline

[9]PR-OWL website: `http://www.pr-owl.org/` accessed 20 November 2010.

these requirements in Sections 1.3.1 and 1.3.3. These measures enable the comparison of different learning and forgetting algorithms, and can determine the complexity of an ontology so that we can evaluate which agents' ontologies are more efficient in generating logical entailments.

We proceed by describing measures that can be used to describe the cost of evolving ontologies and ontology complexity measures. Following that, we consider the standard search and rescue simulation RoboCup Rescue (Kitano and Tadokoro, 2001) to provide a framework for agents to use our learning and forgetting algorithms, so that we can show the effects of different algorithms and the outcome using the standard RoboCup success measure.

2.5.1 Evaluating Costs of Evolving Ontologies

In order to measure the performance of the agent, we require a measure that can relate to the messages that are sent through a network between agents, to the processing costs for retrieving fragments of ontologies. The standard measure of the available or consumed data communication resource is network traffic utilisation, which is the total size of all the messages, measured in bytes. We therefore will record the size of all of the messages the agents send. In order to measure the performance of our agent we consider real time software performance measures: performance profiling, which is measured by the time it takes for each function to complete; A–B timing, which is the time it takes for a set of instructions to complete from program line A to B; and response to external events, for instance the time it takes to acquire the necessary fragment to augment an agent's ontology. In our case, we will require the measure to capture the actions of an agent which may be performed over several functions, and these actions may not relate to external events. Hence, we will record the abstract measure A–B timing to record the time taken to perform an agent's actions.

2.5.2 Measuring the Complexity of an Evolving Ontology

We now consider how to measure the complexity of a T-Box, so that we can evaluate ontologies and the time required to generate logical entailments. In particular, such a measure is dependent on which DL-Reasoner is implemented and the agent's resources. In the current literature related to the measure of complexity for an ontology, there have been several benchmarks for ontologies:

- The LUBM (Guo et al., 2005) and UOBM (Ma et al., 2006) benchmarks, which are the *de facto* standard for reasoning with large ontologies;
- Semintec, the benchmark ontology originally created by the Semintec project[10];

[10]Semintec: `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`

- List (Weithöner et al., 2007), a synthetic benchmark ontology modelling a head|tail list in OWL, which shows that the amount of implicit knowledge rises exponentially with the number of list elements;

- Exquant (Weithöner et al., 2006), a synthetic benchmark ontology heavily using transitive property instances;

- Unions (Weithöner et al., 2006), a benchmark that increases an ontology's A-Box size as well as T-Box complexity to evaluate the ontology's complexity.

These benchmarks, however, focus on the time it takes to infer knowledge from an ontology's A-Box from a single ontology using a specific DL-reasoner. Though their adoption would limit our approach in terms of the types of ontology and reasoners we could use in our investigation. Given this, we want to consider a more universal measure. While these benchmarks are useful in determining which DL-reasoner to use for a specific ontology and other ontologies with similar structures, they do not indicate the complexity of an ontology's T-Box, but instead focus on proving that the DL-Reasoner is more capable of handling a specific feature in an ontology (such as handling A-Boxes that have a high number of entities or the number of relationships). The value of these benchmarks has also been questioned by Weithöner et al. (2006), who argues that these benchmarks use the measure of time against either the number of triples in an ontology or, the number of concepts and instances, to measure the performance of reasoners over queries. These identified flaws would only be able to indicate which reasoner is best suited to our ontology, and does not indicate which ontology is the most complex. Despite the fact the benchmarks have been questioned, in general they indicate that the larger the ontology the greater the amount of time is required to answer queries. Their argument focuses upon the high worst-case complexity of the reasoning complexity of OWL Lite and DL ontologies, and that the desired complexity of an ontology can be intentionally or incidentally created to support the benchmark. In particular, the Unions and UOBM benchmark consider the influence of a T-Box complexity on an A-Box reasoning. Unfortunately, these benchmarks still focus on an ontology's A-Box. In this thesis we focus on evolving an ontology's T-Box, and therefore these benchmarks do not fully meet our requirements (see Requirement 3 in Section 1.3.3).

In addition to benchmarks for the performance of reasoners over specific ontologies, there has been research into measures which can be used to measure the complexity of an ontology. The basic features in an ontology can be used as a measure: Corcho and Gómez-Pérez (2000) list such features that can describe an ontology, these features include concepts and their taxonomies, relations, axioms, and instances. To this end, both Yao et al. (2005) and Kang et al. (2004) propose measures for evaluating the complexity of the subsumption and relationships contained in an ontology, which can be represented in a UML graph. The UML graph representation considers the nodes and

edges, which represent classes and their relationships. In particular, Yao et al. record three measures:

1. Number of root classes (NORC);
2. Number of leaves (NOL);
3. Average Depth of Inheritance Tree of all Leaf Nodes (ADIT-LN).

Each of these measures were used to evaluate the complexity of an ontology separately, and the effectiveness of each measure was compared during an empirical evaluation by a team of humans, where domain experts rated the complexity of a set of ontologies manually. This evaluation showed that the NOL had the best correlation to the experts' value of an ontology's complexity, then the ADIT-LN, and then NORC. While the ADIT-LN and NORC does correspond to the domain experts' evaluation of an ontology's complexity, their evaluation concentrated on ranking a number of ontologies (specifically, into three bounds, of high, medium and low), and comparing this ranking with that of the experts. As such, while this method is useful for direct comparison between ontologies, it has not been proven to produce a measure that can compare evolutions of a single ontology. For example, their measures will produce a comparison of the Gene Ontology and the AKTiveSA Ontology, showing which is more complex, however the measures will not produce a comparable measure for comparing different learning techniques as they evolve an ontology, and are unsuitable for use in our work.

Kang et al. (2004) present a probability-based complexity estimation method whereby an ontology is reduced to a Weighted Class Dependence Graph (WDCG) in order to abstractly calculate its complexity. Each node in the WDCG is classified and weighted according to the type of relationships represented by its edges in the WDCG. The weighting is applied as a probability distribution, based on a ranked list of relationship types, as show in Table 2.17, where *dependency* is the most common and least complex relationship, and *realize* has the highest complexity of the relationship types. These weightings are applied to each node in the graph, and are used to calculate the entropy distance (Burbea and Rao, 1980) between the two nodes, so that the likelihood of the nodes being traversed in the graph can be determined. The combination of the node and relationships' probability distributions are then used to determine an approximate measure of the complexity of the ontology. This approach is dependent on the weightings of relations (in Table 2.17), and as such is unsuitable for all types of ontologies, and thus we will not be using it in our work.

In the work of Yang et al. (2006), they propose three complexity measures (μ, ρ and σ), which are used to represent the complexity of evolving ontologies. In order to calculate μ, ρ and σ, Yang et al. consider a total of eight different measures:

No.	Relation	Weight
1	Dependency	H1
2	Common association	H2
3	Qualified association	H3
4	Association class	H4
5	Aggregation association	H5
6	Composition association	H6
7	Generalization (parent class is concrete)	H7
8	Binding	H8
9	Generalization (parent class is abstract)	H9
10	Realize	H10

TABLE 2.17: The dependency weight value of relations, taken from Kang et al. (2004).

1. Total Number of Concepts ($TNOC$): the sum of concepts in the set C;
2. Total Number of Relationships ($TNOR$): the sum of relationships of each concept;
3. Total Number of Paths ($TNOP$): the sum of paths of each concept;
4. λ_i: the longest path length of concept c_i to the root node;
5. $\overline{\lambda_i}$: the average path length of concept c_i to the root node;
6. μ: the average number of relations per concept, a ratio of TNOR to TNOC, indicating the average degree of connectivity between concepts;
7. ρ: the average number of paths per concept, a ratio of TNOP to TNOC;
8. σ: the ratio of maximum path length to average path length of the ontology, $\sigma = \Lambda \ / \ \overline{\Lambda}$, where Λ is the maximum path length of an ontology, and $\overline{\Lambda}$ is the average path length of an ontology.

The complexity measures, μ, ρ and σ, were used to describe the evolution of the Gene Ontology (GO) (Ashburner et al., 2000), over a period of approximately 2.5 years. The values of the measures were charted and Yang et al. were able to analyse the complexity of the changing ontology over time. By comparing the values of the different complexity measures, it was found that the path-related measures (μ and $TNOP$) varied with relation-related measures (ρ and $TNOR$) synchronously. This shows that due to the increase in relations, every concept has more shortcuts or choices to form paths to the general concept. By using the number of paths in an ontology, this method follows a measure that affects the time that a reasoner takes to traverse an ontology, and thus we will investigate using this technique to evaluate the evolution of our ontologies.

The abstract measures and measures from Yang et al. (2006), Kang et al. (2004), and Yao et al. (2005) determine the complexity of an ontology as a whole, and complexities within an ontology (such as relational complexities between classes). While these measures can

indicate the complexity of an ontology, there is no accepted benchmark used by the community, and no approach has been adopted as a de facto standard. The measures used by Yang et al. and Yao et al. are used to compare a set of ontologies separately, and result in a set of different comparisons. There is no unification of these measures, and there is also no indication of which measure has stronger implications or which ontology is more complex, in terms of the time required to reason entailments.

Therefore, in addition to the research described above, we also consider concrete measures of the complexity of an ontology, with the use of functions available from DL-Reasoners because the measures from DL-Reasoners will provide accurate timings for specific reasoners. Specifically, DL-Reasoners that conform to the DL Implementation Group (DIG)[11] protocol are able to load and consistency check an ontology. These two functions can be used to indicate the time required to infer logical consequences from the T-Box for the worst case scenario, where each path is expanded for inference over a specific ontology. This is similar to the proposed approach of Ma et al. (2006) who perform loading and consistency checking of an ontology to benchmark ontology systems, as it requires a reasoner to fully process an ontology. Based on their experience that this measure is useful for comparing reasoners, we will use it to investigate the comparison of evolving ontologies.

To summarise, we consider investigating both abstract and concrete measures to evaluate the complexity of an ontology. In more detail, we will use DL-Reasoners and the approaches presented in Yao et al. (2005), to evaluate ontologies' complexity for a concrete and abstract measure, respectively. In order to compare these learning and forgetting techniques, we require a system that will enable us to run simulations to which we can apply the techniques individually, and compare the results. In the next section we discuss RoboCup Rescue, which we will use as our simulation environment.

2.5.3 RoboCup Rescue

The RoboCup Rescue (RCR) framework is a standard environment for search and rescue, which provides a platform for multi-agent systems to research strategies for task allocation, agent co-ordination, and path planning, using incomplete information (Kitano and Tadokoro, 2001). We consider RCR for evaluating our approach because it is a standard platform that enables the testing of strategies, and has a standard evaluation measure (Sarika et al., 2009), known as the "Score Vector," which is used in the official RoboCup Rescue competitions. Moreover, the search and rescue paradigm is applicable to our approach because it requires fast decision making given the available information, the faster the decision the more of the city and civilians can be rescued. In more detail, RCR models the effects of an earthquake on a virtual city, by modelling the state of buildings, civilians, and roads. At the beginning of a simulation, buildings may

[11] DIG: `http://dig.sourceforge.net/`

have: collapsed, possibly with civilians buried inside; caused road blockages; and, ignited. There are three types of RCR agents with specific capabilities: ambulance teams recover buried civilians, and transfer them to refuges; fire teams extinguish fires, and police teams clear blocked roads (as shown in Table 2.5.3). Each agent, building and civilian has 10,000 health points which degrade according to simulated physical damage. Through co-ordination, agents can minimise the impact of negative events such as fires and blockades, and increase the number of positive actions such as rescuing civilians.

In RCR, a team of agents must complete its tasks within five seconds, which represents one time step in the simulation. Specifically, a time step is the amount of time that each agent has to decide on its next action before the targets in the world are updated, either with new targets or changes to existing targets. Thus an agent must spend its time efficiently performing actions.

Agent Type	**Task**	**Target**
Ambulance	Rescue buried civilians	Civilians
Fire	Extinguish fires	Buildings
Police	Remove road blockages	Blockades

TABLE 2.18: The types of agents, tasks and abilities, in RoboCup Rescue.

The challenge for a RCR team is to save the lives of as many civilians as possible, and to minimise the area of the city which is burnt. From 2002 to 2008 RCR teams were evaluated using a single formula which factors in the percentage of live civilians, the state of live civilians, and the average building damage, shown in Equation 2.4, taken from Sarika et al. (2009).

$$RCR\ Score = (P + \frac{H}{H_{int}}) * \sqrt{\frac{B}{B_{max}}} \tag{2.4}$$

where P is the number of persons alive, H is the amount of health points (HP) of all the agents and the ratio to the number of persons alive initially, H/H_{int}, shows the efficiency of operations, B is the area of buildings that are not burnt, and B_{max} is the area of all buildings. Scaling factors are used in this formula to adjust the relative importance of each of these factors. As of 2009 a new scoring methodology was introduced, this was introduced because the previous score (shown in Equation 2.4) does not capture the true picture of an agent team's performance during the game resulting the difference scores of two teams being insignificant. In more detail, the new scoring method known as the score vector consists of a set of parameters that can provide analysis of the game at a microscopic level. These parameters are shown in Table 2.19. These parameters can be used to graphically compare the agents' scores over time enabling the organisers and the competitors to evaluate the performance of agents over time. We will use this framework and compare the vectors **D**, **E** and **F**, which compare the ratio of civilians in the refuge, civilians rescued and the percentage of buildings destroyed, respectively. We

will use these measures because they represent the state of the environment, and thus at any point in time they can be directly compared in order to assess which technique is performing best. They are also the most appropriate to compare because they are directly affected by the core objectives of the agents, specifically that when the fire brigade agents are effective, score vector **F** will be high, and when the ambulance agents are effective, score vectors **D** and **E** will be high.

Factor	**Influence on Score**	**Objective**
Agent State:		
A.1. Dead (0≤HP≤10)	Negative	Minimum
A.2. Critical (11≤HP≤40)	Negative	Minimum
A.3. Average (41≤HP≤70)	Positive	Maximum
A.4. Healthy (71≤HP≤100)	Positive	Maximum
B. Time spent by a rescue agent travelling	Negative	Minimum
C. Average number of messages passed	Negative	Optimise
D. Ratio of civilians in refuge	Positive	Maximum
E. Ratio of civilians rescued	Positive	Maximum
F. Percentage of building area destroyed	Negative	Minimum
G. Ratio of fires extinguished	Positive	Maximum
Average time taken to:		
H.1. Rescue a civilian	Negative	Minimum
H.2. Extinguish a fire	Negative	Minimum
H.3. Transport a civilian to a refuge	Negative	Minimum

TABLE 2.19: The factors that influence the performance of a rescue team and the type of influence on the score, taken from Sarika et al. (2009).

In summary, we have identified that RCR provides a framework that can distribute tasks to agents, enable agents to co-ordinate the rescue of a target, and evaluate agent strategies with a standard success measure. Despite these benefits, RCR does not enable agents to have their own ontologies, or the agents the ability to exchange their ontologies or fragments of their ontologies. RCR is also limited in the information about the targets an agent rescues, thus only enabling agents to make simple rescue commands. We require that RCR is more realistic so that instead of using a single rescue command the agent can make the choice between the actions it can take. For example, in RCR an agent calls the extinguish method to put water on the fire. We require that an agent can decide whether to use different suppressants to put out a fire given the materials contained within the building. Thus, in order to meet Requirement 1 (see Section 1.3.1) we will extend RoboCup Rescue in order to enable it to use domain ontologies to model its environment and the agent's knowledge.

2.6 Summary

In summary, the following list describes how the discussed research addresses our research requirements (defined in Section 1.3) and its shortfalls.

Develop a Framework to Enable Agents to Evolve Their Ontology. In order to create a framework for agents to augment their ontologies, we will use RoboCup Rescue (RCR) (see Section 2.5.3) because it distributes tasks to agents, has a well defined success measure, and provides a suitable paradigm to situate our approach. However, RCR does not enable agents to have their own ontologies, or the agents the ability to exchange their ontologies or fragments of their ontologies. We also require that when an agent is assigned a task that it can deliberate which is the best course of action using its ontology, RCR agents use only one type of action to rescue their targets and there is no need for deliberation. For example, in RCR an agent calls the extinguish method to put water on the fire. We require that an agent can decide whether to use different suppressants to put out a fire given the materials contained within the building. Therefore, we will develop an extension for RCR to enable these desired capabilities, see Section 3.2;

Develop Efficient Algorithms to Evolve an Agent's Ontology. In order to evolve an agent's ontology, we consider Seidenberg and Rector's (2006) segmentation technique that selects a set of axioms which relate to a specific concept. This segmentation algorithm enables an agent to choose from a set of related axioms that it is required to learn, and it also provides context for a concept so that an agent can verify the semantics of the acquired concept, thus we will be using this work to enable our agents to: send fragments to describe requested concepts; and determine the semantic context of a fragment to determine its domain. We now summarise the related work for each of the individual subparts of Requirement 2:

Part 1: Currently, most existing work has augmented an agent's ontology with one concept at a time. While this technique increases the vocabulary of an agent, it does not reduce the cost of communicating with the collaborating agent if it requires more vocabulary in the future. To rectify this, we believe that learning concepts that are closely related to the required knowledge can support future queries when an agent answers queries about a specific domain. Therefore we consider augmenting an agent's ontology with a set of concepts closely related to the queried concept, in order to improve the efficiency of our approach (see Requirement 2, in Section 1.3.2);

Part 2: Currently, there is no standard that can be used to predict which concepts will be required to complete future tasks. In order to fulfil this requirement, we will use Markov chains (presented in Section 2.4.3) to store the concepts an agent learns about so that we can predict what concepts are required in

the future. Ontologies which present probabilistic models require an overhead of mark-up which will in turn increase the time necessary to use the ontology (discussed in Section 2.4.4). Thus, we will not use probabilistic ontologies, and instead model our system using Markov chains alone (see Section 5.2), so that Requirement 2 is met, which requires our algorithms to optimise the overhead resources used by their learning and forgetting algorithms;

Part 3: Currently, existing work focuses mainly on augmenting an agent's ontology, but not on pruning it. The Semantic Web community, however, have produced methods that prune ontologies. In particular, we have chosen to use the technique presented by Eiter et al. (2006), which removes one concept and maintains the consistency of the ontology. However, while such research focuses on removing a concept from the ontology, we require removal of more than one concept at a time, because we aim (in Requirement 2, in Section 1.3.2) to reduce the cost of acquiring concepts. Thus, in order to meet this requirement we will apply their heuristics to removing a set of concepts because we aim to maximise the reduction of an ontology's complexity for one removal action (see Section 6.2). In particular, the removal of the concepts is based on the value of knowledge, described by Markovitch and Scott (1988). However, their knowledge valuation does not consider the cost of the acquisition of the knowledge. Thus, we propose that if concepts, cost wise, are cheap to obtain and remain to do so then it is better to forget this concept compared with a concept that was and still is expensive to obtain (see Section 6.2.2). These issues have not been considered in conjunction with respect to an agent evolving its ontology.

Evaluate the Effectiveness of the Ontology Evolution Algorithms. As presented in Sections 2.3.1 and 2.2.2, there are no presented methodologies to compare different agent ontology evolution techniques. Therefore, we consider software measures in Section 2.5 and propose to use the unit of clock ticks to measure the acquisition process and the complexity of an ontology. In more detail, our comparative approaches have been developed with the approaches presented in Sections 2.3.1 and 2.2.2, where an agent can learn or forget one concept and its relationships, learn and forget nothing, learn everything it retrieves, or forget an unnecessary subtree hierarchy from its ontology. We argue that in the literature presented in Sections 2.1 and 2.5 that there is not a conclusive method to estimate the complexity of an ontology, while there are identified factors that influence the time and space complexity of an ontology there is no benchmark, standard measure or measure that determines its complexity. Therefore, we will investigate the appropriate consistency checking measures presented by (Yao et al., 2005) and the DL Implementation Group, in order to evaluate the complexity of the evolving ontologies (see Section 3.4).

To this end, the next four Chapters describe how we implement the identified approaches and our strategies to address the issues that are left unresolved. These issues include: investigating how to evaluate the complexity of an ontology and develop an extension to RCR (in Chapter 3); develop an online ontology learning algorithm (in Chapter 4); develop a proactive online learning algorithm (in Chapter 5); and develop an online forgetting algorithm which considers how to reduce associated costs (in Chapter 6).

Chapter 3

Experimental Design

In this chapter, we describe the design of an agent framework which we use to evaluate learning and forgetting algorithms. In order to perform an evaluation, we require a framework that provides tasks to the agents and additional knowledge which can aid them in completing these tasks. We also require measures that can evaluate the evolution of an ontology. To this end, this chapter addresses all of the requirements detailed in Section 1.3.1 which refers to developing a framework for evaluating our algorithms.

In more detail, in Section 3.1 we define a formal model that describes the requirements for our framework, which we use to extend RoboCup Rescue. Then, in Section 3.2, we describe the framework of RoboCup OWLRescue, RoboCup OWLRescue agents, and the ontologies contained within the framework. Following that, in Section 3.3 we describe benchmark approaches which provide alternative algorithms to augment and prune an ontology so that we can evaluate our algorithms against state-of-the-art and comparative approaches. Then, in Section 3.4 we investigate how to measure the complexity of an evolving ontology using measures specified in Section 2.5.2. In Section 3.5 we detail the evaluation measures that will be recorded and used to evaluate learning and forgetting algorithms. Finally, in Section 3.6 we summarise our framework and contributions.

3.1 Formal Model

In order to adapt the RoboCup Rescue framework, we first introduce and define the components we require in a framework to evaluate online learning and forgetting algorithms. In order to define the components in the system, we use the following global sets:

1. A_t is the set of all task agents, where $A_t = \{a_{t,1}, \ldots, a_{t,n}\}$;
2. A_s is the set of all specialist agents, where $A_s = \{a_{s,1}, \ldots, a_{s,n'}\}$;

3. A is the set of all agents where $A = A_t \cup A_s$;
4. and T is a set of tasks, where $T = \{t_1, \ldots, t''_n\}$.

We also use the following sets to describe an agent's ontology:

1. O is a set of ontologies, where $O = \{o_1, \ldots, o'_m\}$;
2. C is a set of concepts $C = \{c_1, \ldots, c''_m\}$;
3. and K is a set of axioms $K = \{k_1, \ldots, k'''_m\}$.

Each ontology contains a set of axioms, where $o \subset K$. An ontology can only be accessed directly by one agent, and an agent can share axioms from its ontology through collaboration. Each axiom in an ontology makes use of a number of concepts in its definition. All axioms from K refer to a subset of concepts from C. We represent this relationship between an axiom and the concepts it uses with the function $\mathit{refers\_to} : K \to \mathbb{P}C$, we write the set of concepts referred to in axiom k_i as $\mathit{refers\_to}(k_i)$.

Definition 3.1. The concepts contained in ontology o may be identified by taking the union of the sets of concepts referred to by each axiom in the ontology. We write the set of concepts contained in an ontology o_i as $\mathit{concepts}(o_i)$, where:

$$\mathit{concepts}(o_i) = \bigcup_{k \in o_i} \mathit{refers\_to}(k) \tag{3.1}$$

An axiom k defines the relationship between the concepts it refers to, there is a finite set of the types of relationships R all ontologies O can hold (Gruber, 1995) (see Section 2.1.1). We write the set of relationships that are contained in o as R_o. These relationships can be used to define branching factors (Maynard et al., 2006) and paths contained within an ontology:

1. The branching factor bf of a concept c (Edelkamp and Korf, 1998), is referred to as bf_c, and is the number of concepts which relate to c with the subclass relationship, where $|\{c_n \in \mathit{concepts}(o) : \mathit{subclass}(c_n, c)\}|$. The average branching factor, abf, in an ontology, o, is the average number of bf_{c_n} where $abf = \sum_{c_n \in \mathit{concepts}(o_i)} \frac{bf_{c_n}}{|\mathit{concepts}(o_i)|}$ and $|\mathit{concepts}(o_i)|$ is the number of concepts in o_i, as illustrated in Figure 3.1 and defined by Maynard et al. (2006);
2. A path P_c is a set of concepts which are related to concept c by subclass relationships where the last node is the root concept, c_r, in o_i, where $P = c, \ldots, c_r$ (Maynard et al., 2006). The average path length in o_i, P_o, is the average number of P_{c_n}, where:

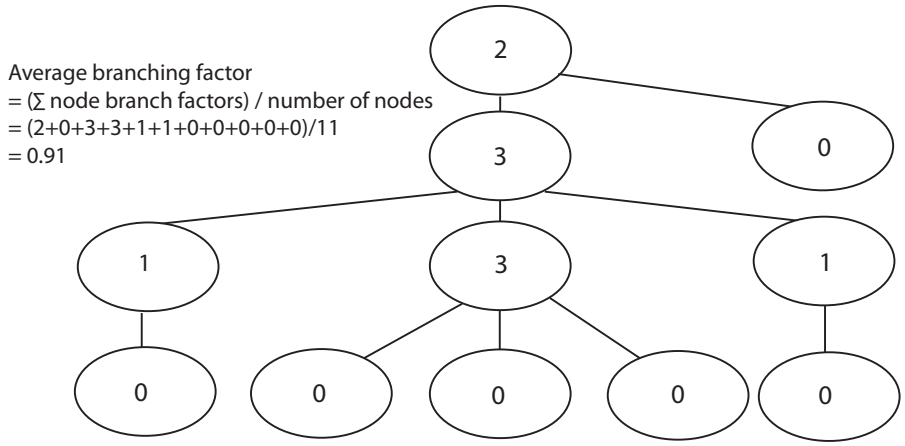


FIGURE 3.1: An example ontology showing the branch factors for each node, and the average branch factor of the ontology.

$$\sum_{c_n \in concepts(o_i)} \frac{P_{c_n}}{|concepts(o_i)|} \tag{3.2}$$

and $|concepts(o_i)|$ is the number of concepts in o_i, as illustrated in Figure 3.2.

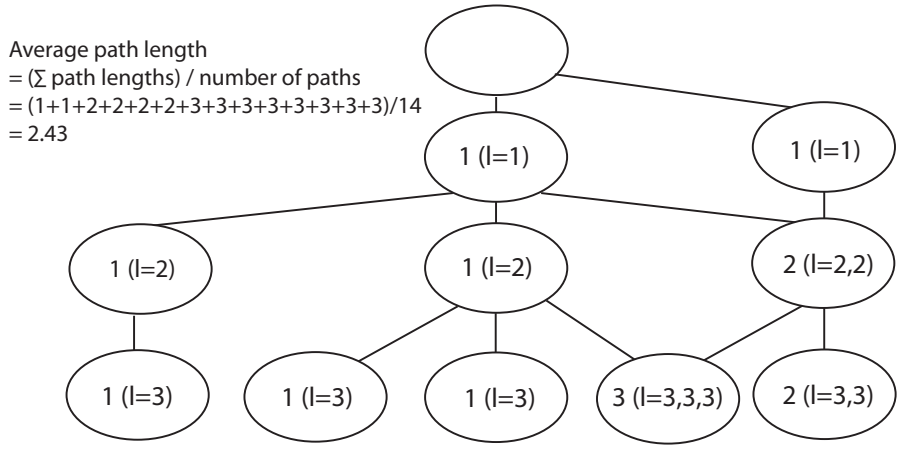


FIGURE 3.2: An example ontology showing the number of paths to root of each node, the lengths of each, and the average path length.

With this notation we introduce the agents in our model. An agent is either:

1. A task agent, which has two distinct ontologies, a Domain Ontology (DO) and an Evolving Ontology (EO). The DO contains the axioms with which the agent is instantiated and does not change. The EO contains acquired axioms and allows the agent to augment its knowledge base with concepts, without affecting its core expertise. The DO imports the EO, when the task agent queries its ontology it uses both the DO and EO. We express these as a tuple of the form $\langle do_i,\ eo_i \rangle$, such that $\forall a_{t,i} \in A_t : a_{t,i} = \langle do_i,\ eo_i \rangle, do_i \in O$ and $eo_i \in O$. The Domain Ontology

is static, whereas the Evolving Ontology is dynamic because it evolves during the lifetime of the task agent. The task agent $a_{t,i}$ uses both of these ontologies to complete tasks, we denote the union of a task agent's ontologies as $o_i = do_i \cup eo_i$, where $o_i \in O$. The do_i and the eo_i can contain axioms about the same concepts, where $concepts(do_i)$ and $concepts(eo_i)$ are not necessarily disjoint, however, they do not contain the same axioms, where $do_i \cap eo_i = \emptyset$;

2. A specialist agent, which has one distinct ontology, a Domain Ontology. We express its ontology as a tuple of the form $\langle do_i \rangle$, such that $\forall a_{s,i} \in A_s : a_{s,i} = \langle do_i \rangle, do_i \in O$. The Domain Ontology is static and does not evolve, because specialist agents do not learn information or augment their ontologies.

Specifically, a task agent desires to complete a subset of tasks $\{t_x, \ldots, t_y\}$, from the set T, where $\{t_x, \ldots, t_y\} \subset T$, and an agent can receive one or more tasks at a time. The task agent cannot access the set T and thus does not have perfect foresight. In order to complete a set of tasks the agent requires a subset of axioms from K, thus it does not require that its ontology contains all the axioms in the framework. In order to complete a task, an agent $a_{t,i}$ can learn about the concept c by augmenting its eo_i with a fragment of an ontology. An ontology fragment, which is denoted by $f_{o_j,c}$, is a subset of an agent's ontology o_j, generated with respect to concept c. The task agent augments its eo_i with concepts from fragments provided from specialist agents $f_{o_j,c}$. Thus, the result of augmenting eo_i with $f_{o_j,c}$ is $eo_i^+ = eo_i \cup f_{o_j,c}$, where eo_i^+ denotes an agent's EO after it has been augmented. When an agent receives more than one fragment to learn from which describes c, more than one ontology contains definitions for a concept. When more than one ontology contains the definition of a concept, we describe them as having common concepts as defined in Definition 3.2.

Definition 3.2. Two ontologies o_i and o_j have common concepts when at least one concept is contained in both ontologies. We denote this using the predicate commonConcepts, where:

$$\forall o_i, o_j \in O \; commonConcepts(o_i, o_j) \Leftrightarrow concepts(o_i) \cap concepts(o_j) \neq \emptyset \tag{3.3}$$

In addition to learning, a task agent can also forget a subset of concepts from its ontology. In order to forget a subset of concepts from its ontology, we require that an agent $a_{t,i}$ can create a fragment of a certain concept $f_{o_i,c}$ from its ontology, then specify which concepts to remove and remove them. The agent prunes its eo_i with concepts that it selects from the fragment $f_{o_i,c}$ relating to c. If agent $a_{t,i}$ prunes all of the concepts contained in the fragment $f_{o_j,c}$, it would result in $eo_i^- = eo_i \cup f_{o_j,c}$, where eo_i^- denotes an agent's EO after it has been pruned.

In summary, we require that our evaluation framework has:

1. Task agents that have:

   (a) A Domain Ontology;

   (b) An Evolving Ontology.

2. Specialised agents that have an environment ontology;

3. Tasks that require an agent's ontology to contain subset of K to complete.

We extend our formal model in Section 3.2.2, so that we can describe how our RoboCup Rescue extension fulfils the above requirements. The formal model is also used to describe the measures defined in Section 3.5, and our evolution algorithms in Chapters 4, 5, and 6. The next section provides an overview of the required changes to RoboCup Rescue to support the above model, and details the RoboCup OWLRescue extension.

3.2 A Framework for Evolving Ontologies: RoboCup OWLRescue

The RoboCup Rescue simulation (introduced in Section 2.5.3) provides an environment for the development of strategies to enable agents to allocate tasks, co-ordinate agents and plan their path to their target which is either a building, civilian or road. Once the agent has reached its target it can call the functions *extinguish*, *rescue* and *clear*, respectively. The state of the target is affected by environment variables, such as the wind direction and the fabrication of the buildings, however, these environment variables provide a limited model and are intended to be used for generating fires by the simulator. This framework partially supports our requirements in Section 3.1, where the fire brigade, ambulance, and police agents are task agents A_t (Requirement 1), and the tasks by the framework T (Requirement 3). However, these required components are not modelled with the degree of detail that would enable a task agent to make decisions about its actions according to its current task. For example, the injuries that civilians sustain and the equipment required to treat them are not modelled.

Thus, our extension aims to provide additional information to the environment so that agents can allocate specialist equipment to handle tasks that require such equipment, supporting our third requirement (see Section 1.3). For example, an agent is required to extinguish a fire that has engulfed a store of chemicals, and these chemicals react to water. With the knowledge that these chemicals react to water, the agent can then use a dry agent to put out the fire. Without this knowledge, the agent may use a water jet to put out the fire causing the fire to become uncontrollable. This reflects real life scenarios where tasks can be unpredictable without the right knowledge because of various interrelated factors. Dissemination of knowledge is vital because it is unrealistic

that rescue workers are aware of all the likely effects of different variable combinations because it is impossible to have perfect foresight. It is noted by the community that this is a real problem, thus there are various standards that provide reference material to rescue workers so that they can overcome a lack of knowledge; for example the Emergency Action Code (EAC) list (National Chemical Emergency Centre, 2009) and Hazard Identification Number (HIN) [1] are reference guides where there are over 1000 chemicals listed with their recommendations and these are annually updated. In order to provide the detail we require to enable agents to make decisions about how to rescue their target, we extend RoboCup in the following ways:

1. Additional variables for: the buildings, which include a set of chemicals contained in the building; blockages, which have a height and weight; and civilians which have symptoms;

2. A set of specialist agents (A_s) that can disseminate knowledge from the set of environmental ontologies (O) which contain knowledge about: chemicals and how to handle them; vehicles and their equipment; and symptoms and the equipment needed to treat them. This provides the RoboCup OWLRescue agents with a common knowledge base which can be accessed by all agents;

3. A personal ontology for each RoboCup OWLRescue agent (o_i) which they can augment with knowledge from the environmental ontologies;

4. Specialist equipment for RoboCup OWLRescue agents such as jet and foam nozzles for putting out fires, and breathing apparatus for fire fighters.

Our approach thus enables agents to extend their actions, and provides a new practical problem which can be proposed as an AI problem. In summary, we make the following contributions to simulating a disaster scenario:

1. We extend the RoboCup Rescue targets, buildings, civilians, and blockades with additional variables, such as chemicals which are contained in buildings, symptoms which civilians can suffer from, and the height and weight of blockades;

2. We provide RoboCup Rescue with specialist agents (A_s) that can disseminate knowledge from environment ontologies (O) and task agents (A_t) that have private ontologies (o_i). This enables the task agents to acquire the knowledge that can enable the agent to make appropriate decisions in the selection of equipment when rescuing a target thus maximising their success;

[1] The HIN is used in accordance with the 1957 European Agreement concerning the International Carriage of Dangerous Goods by Road (ADR), see UN document ECE/TRANS/202.

3. Together the above provide a general framework which furthers RoboCup Rescue's contribution to enable the development of strategies to allocate and co-ordinate resources.

To this end, we now introduce the basic structure of RoboCup OWLRescue (RCOR).

3.2.1 Architecture of RCOR and Simulators

The original RCR architecture is built up of five components: agents which are of three types: fire brigade, ambulance team, and police; a Geographic Information System (GIS) which stores and manages the co-ordinates of the roads and the buildings in virtual cities; sub-simulators which generate fires, blockades, and civilians' health contained in a virtual city; the kernel, which manages the simulation by handling actions from agents which affect the world's buildings, blockades and civilians; and the viewer, which provides a visual representation of the virtual city's state (shown in Figure 3.3). These components are shown in Figure 3.4 and are grey hatched. In our RCOR extension, agents have their own ontologies which they can use to save their targets, and can request additional information about concepts from the environment ontologies. The simulators generate chemicals for buildings, symptoms for civilians, and height and widths for road blockades. In order to generate these additional variables the simulators use the information contained in the environment ontologies.

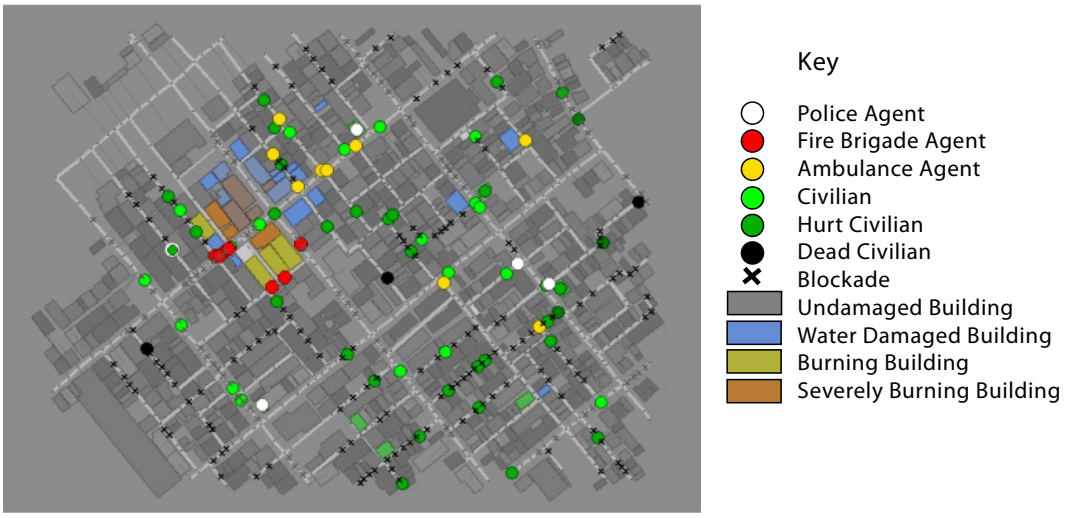


FIGURE 3.3: The Tokyo Map from RoboCup Rescue.

In more detail, the RCR virtual city (shown in Figure 3.3) has buildings represented with rectangles that are filled with colour to indicate the state of the building, blue indicates water damage, and yellow through to orange, red and then black indicates the effect of a burning building from ignition to burning out. The roads are represented with light grey lines connecting different parts of the city, and the blockades are black

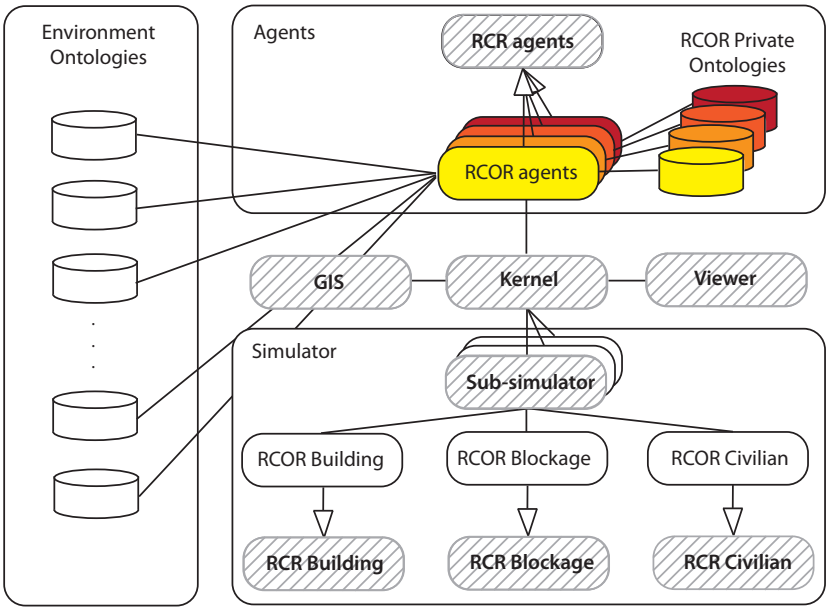


FIGURE 3.4: The RCOR Framework.

crosses. The agents and civilians are represented with circles: red, yellow, white, green and black represent fire brigades, ambulance teams, police, alive civilians, and dead civilians, respectively.

Currently RCR implements no additional information about the city's buildings aside from their location and size. Our extension aims to model the city with greater detail, therefore we zone our city map into different segments. Zoning is a commonly used technique by local governments to plan the use of land (Lefcoe, 2005). Often these zones can be categorised as open space, residential, agricultural, commercial or industrial. In our case, we zone the RCR maps into three zones residential, commercial and industrial because these zones have the highest use of chemicals and is best suited to the city maps in RCR. Each zone in the city is selected by clustering a random set of buildings, we randomise the buildings' zones because we use the same map in every simulation and it enables each simulation to be different. For example, the building with the ID 52 in scenario 1 is in a commercial zone and will have a different possible set of chemicals than scenario 2 where it is in an industrial zone. These random zone allocations allow the city to have different buildings for each type of area, for each simulation. These zones are represented in Figure 3.5 where orange, green and purple represent industrial, residential, and commercial, respectively. While the zoning of the city is random, the clustering technique selects approximately a third of buildings for each of the three areas so that the scenario is not biased towards chemicals from one type of zone. This generation of the zones can be used to generate many different random areas from one map. The zoning we use to segregate our map is not a faithful representation of zoning in the real world, this is not our aim. Rather, our aim is a more detailed virtual city compared with the existing description in RCR and because of the zones,

the map contains more information about types of chemicals and equipment that would be available in an area. Each area has a set of building types that it can contain, and each building has a possible set of chemicals that can be contained in each building type. For example, a `bakery` is a building type which is in `commercial` area, and a `bakery` may contain a subset of chemicals from the set {`gms_powder`, `sodium_cyclamate`, `pectin`, `sorbitol`, `sorbic_acid`, `ascorbic_acid`}. The RCOR Building sub-simulator generates the chemicals contained within the city's buildings. First it generates the city zones, second it assigns each building with a building type, finally selects a set of chemicals for the building.

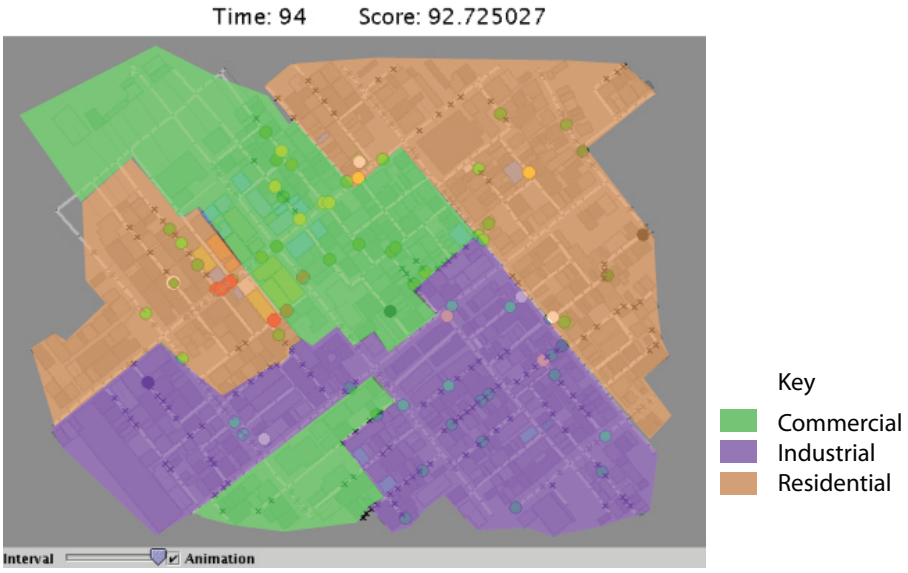


FIGURE 3.5: A zoned city map.

Formally, let: B be the set of all buildings; COM, IND, RES be the sets containing all the buildings belonging to commercial, industrial, and residential areas respectively; $getBuildingTypes(area)$ be a function that returns all the types of buildings given the parameter of an area; and, $getPossibleChemicals(buildingType)$ be the function that returns all the possible chemicals that can be contained in a building type. Given these sets and functions, we define the methodology for generating zones and assigning building types in Algorithm 2 and generates the chemicals for each building in the city in Algorithm 3.

A civilian's symptoms are generated by the RCOR Civilian sub-simulator. At the beginning of the scenario all civilians are located inside buildings. When an ambulance rescues a buried civilian, the civilian will suffer with two types of injuries: first, injuries that relate to being buried under rubble or from a burning building (i.e. broken bones, bruising, burns, etc.); and second, injuries related to chemicals that they were exposed to from the building that they were trapped in (i.e. chemical burns, chemical poisoning, dizziness, confusion etc.). The second type of chemical induced injury is generated by a selecting a subset of possible symptoms given the exposure to a specific chemical. Formally, let:

Algorithm 2 Algorithm to generate zones of the city and assign building types to buildings.

Require: $B \leftarrow \{all\ buildings\}$

Function: $getBuildingTypes(area)$: `return set of types of buildings given a zone`
Function: $buildingsOnSameBlock(building)$: `return set of all buildings on the same city block as building`
Function: $neighbourBlock(building)$: `return a set of buildings on the neighbouring city block to building`
Function: $random(set)$: `return random element from a set`

1: $Zones = \{COM, IND, RES\}$
2: `/* loop through all buildings */`
3: **for** $building \in B$ **do**
4: `/* check if building is assigned a zone */`
5: **if** $building.zone = unassigned$ **then**
6: $newZone \leftarrow random(zones)$
7: `/* get all other buildings on the same block */`
8: $WholeBlock \leftarrow buildingsOnSameBlock(building)$
9: $Neighbours \leftarrow WholeBlock$
10: `/* recursively select 10 neighbouring blocks */`
11: **for** $i = 1$ to 10 **do**
12: $randomBuilding \leftarrow random(Neighbours)$
13: $Neighbours \leftarrow Neighbours \cup neighbourBlock(randomBuilding)$
14: **end for**
15: `/* set zone and building type of each selected building */`
16: **for** $blockBuilding \in Neighbours$ **do**
17: $blockBuilding.zone = newZone$
18: $PossibleTypes = getBuildingTypes(newZone)$
19: $chosenType = random(PossibleTypes)$
20: $blockBuilding.type = chosenType$
21: **end for**
22: **end if**
23: **end for**
24: **return** B

$C = \{c_1, \ldots, c_n\}$ be the set of civilians in a scenario; the function $getBuilding(c)$ returns the building that the civilian is in; and, the function $getPossibleSymptoms(building)$ return the possible set of symptoms given the chemicals contained in the building. Using these definitions we define Algorithm 4 for generating symptoms for injured civilians.

The RCOR Blockade sub-simulator generates a weight and width for each road blockade. When the simulator generates a blockade, using the GIS it finds the width of the road and generates a random width for the blockade. The simulator then generates the weight based on the damage of the surrounding buildings and their volume. For example, a wooden oak building has a volume of 12 meters cubed and has sustained 50% damage and on average per cubic meter the building is 330kg, the blockade next to the building

Algorithm 3 Algorithm to generate chemicals for each building in the city.

Require: $B \leftarrow \{all\ buildings\ \}$

Function: $buildingType(building)$: `return the building type of a building`
Function: $random(set)$: `return random element from a set`
Function: $getPossibleChemicals(building)$: `return the possible chemicals that can that be contained in the building`

```
1: /* loop through all buildings */
2: for building ∈ B do
3:    buildingType ← buildingType(building)
4:    PossibleChemicals ← getPossibleChemicals(buildingType)
5:    /* assign 3 random chemicals to the building */
6:    for i = 1 to 3 do
7:       selectedChemical ← random(PossibleChemicals)
8:       building.chemicals ← building.chemicals ∪ selectedChemical
9:    end for
10: end for
11: return B
```

Algorithm 4 Algorithm to generate symptoms for each civilian in the city.

Require: $C \leftarrow \{c_1, \ldots, c_n\}$ `/* all civilians */`

Function: $getBuilding(civilian)$: `return the building that the civilian is in`
Function: $getPossibleSymptoms(chemicals)$: `return the possible symptoms that can be caused by these chemicals`
Function: $random(set)$: `return random element from a set`

```
1: /* loop through all civilians */
2: for civilian ∈ C do
3:    building ← getBuilding(civilian)
4:    /* get the possible symptoms of the building's chemicals */
5:    PossibleSymptoms ← getPossibleSymptoms(building.chemicals)
6:    /* assign 3 random symptoms to this civilian */
7:    for i = 1 to 3 do
8:       symptom = random(PossibleSymptoms)
9:       civilian.symptoms ← civilian.symptoms ∪ symptom
10:   end for
11: end for
12: return C
```

weighs 2280kg. There are three types of building material in RCR, wood, concrete and steel, the average weight of these materials per cubic meter is shown in Table 3.1.

During the simulation the sub-simulators - building, blockade, and civilian simulators - generate changes to the simulation every five seconds. Each five seconds is called a timestep, and in a timestep agents evolve their ontologies, deliberate their actions and act. The length of a timestep in RoboCup Rescue is standardised to five seconds, because

Material	Weight (kg)
Wood	330
Steel	550
Concrete	740

TABLE 3.1: A table showing the building material and its average weight per cubic meter.

it models the time constraints in the real world (see Section 2.5.3). If the length of a timestep is increased, then agents would be able to spend more time deliberating their actions and the simulation would progress slower. The opposite is true if the length of a timestep decreased, agents would have to reduce the amount of time they deliberate in order to decide on an effective action. Therefore, an agent's strategy should be optimised so that it is able to complete an action within a given timestep. In more detail, our extension can perform in two modes:

1. **Discrete**, where all agents must perform all of their actions within the specified five second timeframe. This means that agents must be able to prepare and act on their plan within this timeframe, otherwise they will not be able to rescue any of their targets. The agent cannot remember the planned actions and has to make a new plan each timestep. This is the same technique used in the RoboCup Rescue scenario which mandates that each agent must perform all of its actions within the five second timeframe;

2. **Continuous**, where an agent can continue to perform its planned action over as many timesteps as necessary. The simulators still update the world with new fires, blockades, and injured civilians every five seconds, thus it is beneficial for an agent to perform its actions in the smallest amount of time so that it can make plans on up-to-date information.

At the end of the simulation the performance of the agents is evaluated using the RoboCup Rescue score vector (introduced in Section 2.5.3).

3.2.2 RCOR Agents

The agents in RCOR have three specialised teams: ambulance teams, fire brigades, and police agents and this partially fulfils the second part of our specification in Section 3.1 by providing *specialised task agents*. Thus, we extend our formal definition in Section 3.1 by specifying that task agents may belong to one of three teams: ambulance, fire brigade, and fire, in sets A_{amb}, A_{fire}, and A_{police}, respectively where $A_t = A_{amb} \cup A_{fire} \cup A_{police}$. Each of these teams consist of a set of platoon agents which rescue targets and a station agent which is stationary and located in a building, let $A_{team} = \{a_s, a_1, \ldots a_n\}$ where

A_{team} is a specialised team, a_s is the station agent, and $a_1, \ldots, a_n$ is the set of platoon agents. Each agent (a) has its own private ontology o_a consisting of two ontologies, its initialised domain ontology DO and an evolving ontology EO. The agents can add concepts from ontologies contained by other agents in the environment, these ontologies are called environment ontologies and are represented by the set O_E. In addition to an agent having its own ontology, it also uses a vehicle to rescue targets where $V = \{v_1 \ldots v_n\}$ is the set of vehicles. Each vehicle has its own set of equipment, where all pieces of equipment are contained in the set $E = \{e_1 \ldots e_n\}$. The set of equipment belonging to a vehicle is not always unique, it is dependant on the equipment available on different vehicles. Throughout the simulation an agent can change its vehicle by moving to its station and selecting a vehicle contained within that station.

The framework is instantiated with two resources: first, a set of specialist agents A_s that disseminate information from the environmental ontologies O_E which provide information about variables or concepts $concepts(O_E)$ within the scenario; and second, with a finite set of vehicles V that are stationed at a station s, where the function $containsVehicles(s)$ returns a set of vehicles $v = \{v_1 \ldots v_n\}$ that are stationed in that agent's building. In the next two sections we describe the ontologies which our agents use, in Section 3.2.3 we detail the task agent's ontologies and in Section 3.2.4 we detail the specialist agent's ontologies.

3.2.3 RCOR Task Agents' Ontologies

The RCOR agents are initiated with an ontology that contains a set of basic concepts which enables agents to use a vehicle to achieve their tasks. The agents can augment their ontology with concepts from the environment ontologies so that they can perform their task more efficiently. These ontologies are depicted in Figure 3.6, these ontologies have common concepts `thing` and `vehicle`; `thing` is the common root class, and `vehicle` is the common to all agents because they all traverse roads to the location of their target. The types of `vehicles` in each type of agent's ontology differs because each agent is required to fulfil different tasks, and thus needs a vehicle with specific capabilities. For example, a fire engine is required to put out fires, an ambulance is required to transport civilians to refuges, and the police are required to remove blockages from the road.

A RCOR agent can augment its ontology with concepts from a set of ontologies from specialist agents. When an agent receives a task then it uses its personal ontology to plan how to carry out its response. In contrast to our extension, all types of RCR agent use the same sequence to respond to a list of target ID's (shown in Figure 3.7). With our extension, each agent has a different sequence and requires different information to respond to their tasks. The sequence diagrams in Figures 3.8, 3.9, and 3.10 shows the sequence of events for agents which put out fires, rescue civilians and remove blockages, respectively.

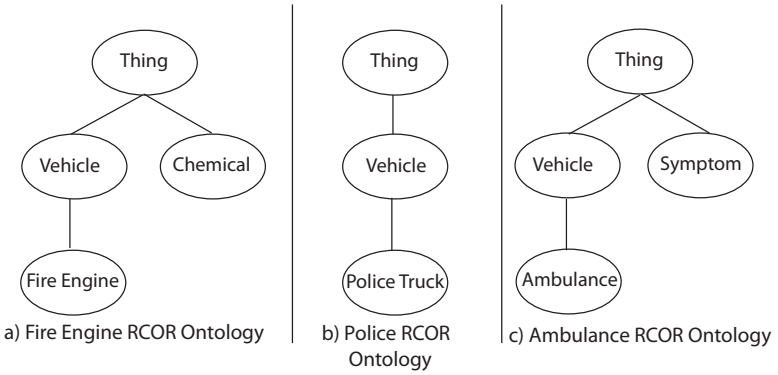


FIGURE 3.6: The subsumption hierarchy of the ontologies for fire brigade, police, and ambulance agents.

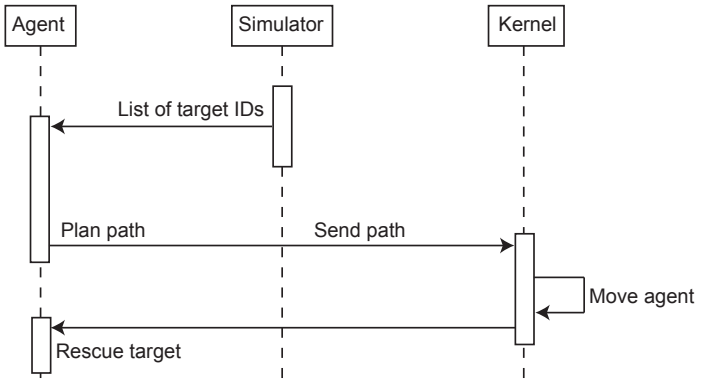


FIGURE 3.7: A sequence diagram of the behaviour of a RoboCup Rescue agent.

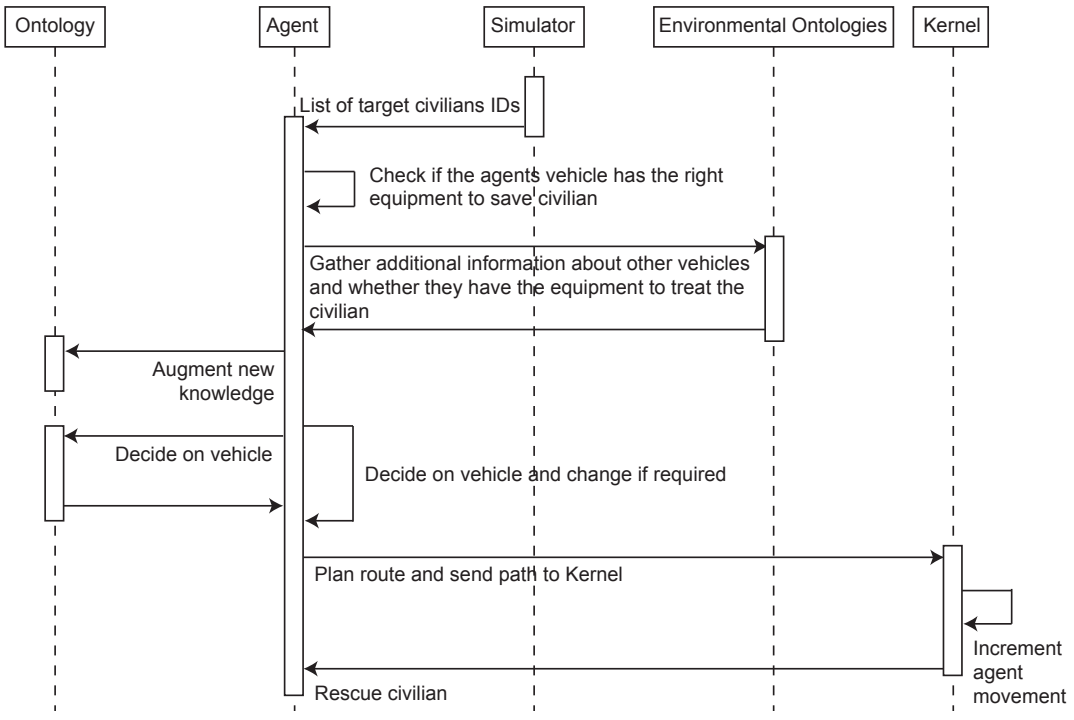


FIGURE 3.8: A sequence diagram of a RoboCup OWLRescue ambulance agent.

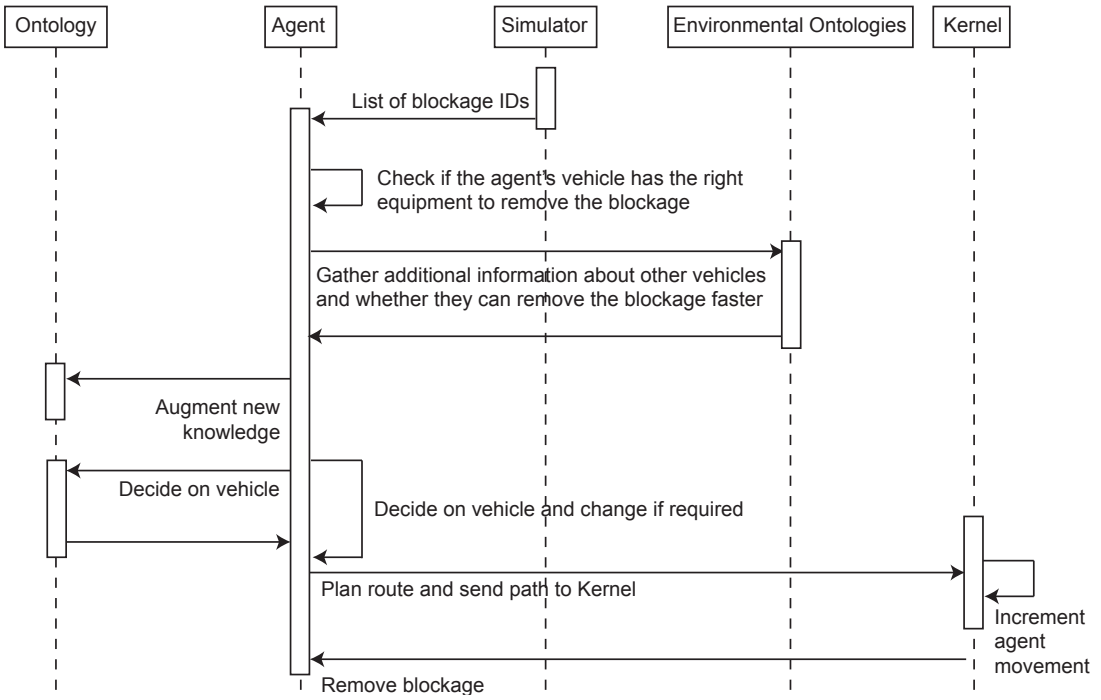


FIGURE 3.9: A sequence diagram of a RoboCup OWLRescue police agent.

We now proceed to describe the ontologies which describe the framework's resources, which are used by the specialist agents to disseminate information to the task agents so that they can complete tasks.

3.2.4 Specialist Agents' Environment Ontologies

In more detail, we use OWL ontologies to enable agents to define entities with a formal standardised structure, and constitute a vocabulary. In our setting, RoboCup OWLRescue, an agent can use the knowledge contained in the ontologies to make the following decisions on which equipment to use, to:

1. Extinguish fires which contain chemicals;
2. Rescue buried civilians with symptoms;
3. Remove rubble from roads which have a weight and height.

In the real world there are many sources of information from government bodies to industry standards, and in order to find a particular piece of information it is common to consult more than one source. Despite an abundance of information sources, not all of these sources will contain the required information and searching these sources may provide extraneous information.

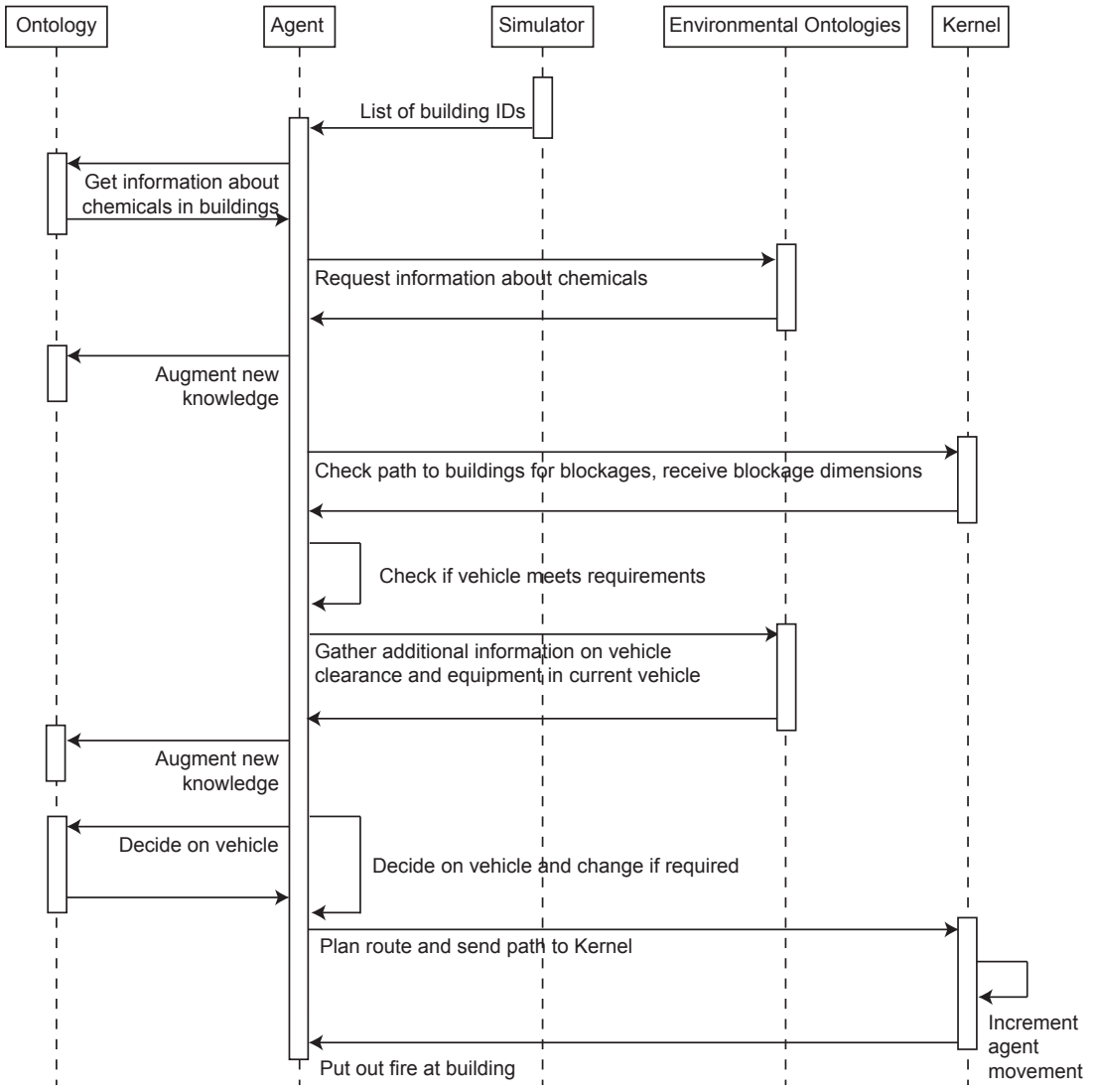


FIGURE 3.10: A sequence diagram of a RoboCup OWLRescue fire brigade agent.

In order to model the different sources of information we use ontologies, each ontology provides information modelled from a single body and is either published in the form of an ontology, or derived from the provided hierarchies and standards. The environment ontologies provide specialist information on a particular domain, in particular we focus on chemicals, vehicles, and treatments for injuries. These ontologies contain common concepts, thus enabling the agent to benefit from identifying the same concept in different ontologies. Our framework includes the following ontologies, which have been chosen because they are representative of standard industry vocabularies for the domains of interest of the RCR agents. This combination of ontologies covers the areas required by the RCOR extension and represent a realistic set of information that rescue agents would need to consult in real conditions:

1. **EAC Ontology**, this ontology describes the Emergency Action Code (EAC), which is a three character code displayed on all dangerous goods classed carriers,

and provides a quick assessment to first and emergency responders. The three characters (one number and two letters) classify the category of the dangerous goods. The number assigned to a good specifies how to extinguish the good if caught alight (see Table 3.2).

Number	**Action**
1	Jets
2	Fog
3	Foam
4	Dry Agent

TABLE 3.2: The EAC Number Categories.

The letters assigned determine what actions are required when handling, containing and disposing of the chemical in question (see Table 3.3). Eight 'major categories' exist which are commonly denoted by a black letter on a white background. Four subcategories exist which specifically deal with what type of personal protective equipment responders must wear when handling the emergency, denoted by a white letter on a black background. If a category is classed as violent, this means that the chemical can be violently or explosively reactive, either with the atmosphere or water, or both. Finally, the letter 'E' may be added to the number and letter code to signify that evacuation should be considered.

<table>
<tr><th>Category</th><th>Violence</th><th>Protection</th><th>Substance Control</th></tr>
<tr><td>P</td><td>V</td><td rowspan="2">Full</td><td rowspan="5">Dilute</td></tr>
<tr><td>R</td><td></td></tr>
<tr><td>S</td><td rowspan="2">V</td><td>BA</td></tr>
<tr><td>S</td><td>BA for fire only</td></tr>
<tr><td>T</td><td rowspan="2"></td><td>BA</td></tr>
<tr><td>T</td><td>BA for fire only</td></tr>
<tr><td>W</td><td>V</td><td rowspan="2">Full</td><td rowspan="6">Contain</td></tr>
<tr><td>X</td><td></td></tr>
<tr><td>Y</td><td rowspan="2">V</td><td>BA</td></tr>
<tr><td>Y</td><td>BA for fire only</td></tr>
<tr><td>Z</td><td rowspan="2"></td><td>BA</td></tr>
<tr><td>Z</td><td>BA for fire only</td></tr>
<tr><td>E</td><td colspan="3">Consider evacuation</td></tr>
</table>

TABLE 3.3: The EAC Letter Categories.

This ontology can provide RCOR fire brigade agents with information about which capabilities their vehicle must have. For example, when a chemical is present in a burning building it requires the use of a dry agent so that the fire can be put out safely, thus the vehicle is required to have a dry agent suppressant. The emergency action code list is published by the National Chemical Emergency Centre

(NCEC)[2].;

2. **HazChem Ontology**, is a Hazardous Chemical (HazChem) ontology which classifies chemicals using Hazardous Identification (ID) Numbers. A HID number consists of two or three numbers, which indicate the hazards shown in Table 3.4.

Number	Hazard
2	Emission of gas due to pressure or to chemical reaction
3	Flammability of Liquids
4	Flammability of solid or seal-heating liquid
5	Oxidising (fire-intensifying) effect
6	Toxic or risk of infection
7	Radioactivity
8	Corrosivity
9	Risk of spontaneous violent reaction

Table 3.4: The HazChem HID numbers.

These numbers are combined to form the HID numbers, in particular the doubling of a number indicates intensification of that particular hazard. If the HIN is prefixed with an "X" this indicates the substance will react dangerously with water. For such substances, water may only be used by approval of experts. Where the hazard associated with a substance can be adequately indicated by a single figure, this is followed by a zero. The combinations shown in Table 3.5 have special meanings and refer to the Hazchem List or control.

Similar to the EAC Ontology, this ontology can also provide RCOR fire brigade agents with information about which capabilities their vehicle must have. For example, a chemical which is reactive with water is present in a burning building, an agent requires that its vehicle is equipped with a dry agent to extinguish the fire safely. The hazardous chemical classifications are published by the United Nations Economic Commission for Europe (UNECE)[3].;

3. **Chemical Ontology**, contains chemicals and their EAC and HID classification. This ontology allows agents to use either standard provided by the EAC and HID. For example, an agent may only wish to use one classification technique thus it is required to translate any chemicals which are described in the other. This ontology contains information found in the emergency action code list, published by the National Chemical Emergency Centre (NCEC)[4], and the hazardous chemical classifications are published by the United Nations Economic Commission for Europe (UNECE);

[2]The NCEC: `http://the-ncec.com/ncec`

[3]The United Nations Economic Commission for Europe (UNECE) publish the HazChem recommendations and are available online: `http://www.unece.org/trans/danger/publi/adr/pubdet.htm`

[4]The NCEC: `http://the-ncec.com/ncec`

Number	Hazard
323	Flammable liquid which reacts with water, emitting flammable gases
362	Flammable liquid, toxic, which reacts with water, emitting flammable gases
382	Flammable liquid corrosive, which reacts with water, emitting flammable gases
423	Solid, flammable solid or self-heating solid which reacts with water, emitting flammable gases
432	Spontaneously flammable (pyrophoric) solid which reacts with water, emitting flammable gases
44	Flammable solid, in the molten state at an elevated temperature
446	Flammable solid, toxic, in the molten state at an elevated temperature
462	Toxic solid which reacts with water, emitting flammable gases
482	Corrosive solid which reacts with water, emitting flammable gases
623	Toxic liquid which reacts with water, emitting flammable gases
642	Toxic solid, which reacts with water, emitting flammable gases
823	Corrosive liquid which reacts with water, emitting flammable gases
842	Corrosive solid which reacts with water, emitting flammable gases

TABLE 3.5: The HazChem HID numbers classification.

4. **Vehicle Ontology**, describes land vehicles, their attributes, purpose, and manufacturer. In particular, this ontology provides information about the track type of the vehicle, and the capabilities allowing them to perform certain tasks[5]. For example, a fire truck has the purpose of putting out fires and can put out fires with a water jet and foam;

5. **HantsFireEngineFleet Ontology**, this ontology contains information about the fleet of fire engines in the county of Hampshire (UK). This information is derived from the Hampshire fire service website[6], which details vehicle types, their model, manufacturer, and registration numbers. Each vehicle of the same type and different manufacturer has different capacities;

[5]This information is taken from John Dennis Coach Builders who specialise in making custom built fire engines: `http://www.johndennisfire.co.uk/`

[6]Hampshire Fire and Rescue Service: `http://www.hantsfire.gov.uk/theservice/sp-and-sr/fleetmanagement.htm`

6. **Ambulance Ontology**, this ontology contains information about different types of ambulance, their attributes, and equipment. This information is derived from the website of the American College of Surgeons[7];

7. **ConstructionVehicles Ontology**, this ontology contains information about construction vehicles, their capacity and equipment. The ontology also details the height clearance and drive (i.e. 4-wheel drive and front-wheel drive). This information is taken from the Construction Equipment Guide website which details information about sales of construction vehicles and news[8];

8. **Triage Ontology**, is an ontology that describes the 5-Category Triage System and identifies symptoms for each category excluding 'Black' (see Table 3.6). Table 3.6 describes the 5 categories[9];

Priority	Colour	Symbol	Casualty Condition
First	Red	R	Critical: Likely to survive if simple care given within minutes.
Second	Blue	B	Catastrophic: Unlikely to survive and/or extensive or complicated care needed within minutes.
Third	Yellow	Y	Urgent: Likely to survive if simple care given within hours.
Fourth	Green	G	Minor: Likely to survive even if care delayed hours to days. May be walking or stretcher cases.
None	Black	X	Dead.

Table 3.6: The 5-Category Triage System.

9. **CSI Ontology**, this ontology contains information from the Chemical Sampling Information (CSI) of the United States Department of Labor Occupational Safety and Health Administration (OSHA). The CSI contains details about chemicals, its Chemical Abstract Service (CAS) number, their health effects on humans, symptoms of exposure to the chemical, and the organs affected. The OSHA have developed a standard code, a health code, to classify each type of chemical (see Table 3.7);

10. **Treatment Ontology**, this ontology contains information about types of injuries. In particular, it details information about burns and broken bones, their symptoms,

[7] American College of Surgeons: `www.facs.org/trauma/publications/ambulance.pdf`.
[8] Construction Equipment Guide website: `http://www.constructionequipmentguide.com/`.
[9] Taken from the government website of the Agency for Healthcare Research and Quality, US department of Health and Human Services: `http://www.ahrq.gov/research/esi/`.

Code	Health Effects
HE1	Cancer—Currently regulated by OSHA as carcinogen
HE2	Chronic (Cumulative) Toxicity—Known or Suspected animal or human carcinogen, mutagen (except Code HE1 chemicals)
HE3	Chronic (Cumulative) Toxicity–Long-term organ toxicity other than nervous, respiratory, hematologic or reproductive
HE4	Acute Toxicity—Short-term high risk effects
HE5	Reproductive Hazards—Teratogenesis or other reproductive impairment
HE6	Nervous System Disturbances– Cholinesterase inhibition
HE7	Nervous System Disturbances– Nervous system effects other than narcosis
HE8	Nervous System Disturbances–Narcosis
HE9	Respiratory Effects Other Than Irritation—Respiratory sensitization (asthma or other)
HE10	Respiratory Effects Other Than Irritation—Cumulative lung damage
HE11	Respiratory Effects—Acute lung damage/edema
HE12	Hematologic (Blood) Disturbances–Anemias
HE13	Hematologic (Blood) Disturbances–Methemoglobinemia
HE14	Irritation-Eyes, Nose, Throat, Skin–Marked
HE15	Irritation-Eyes, Nose, Throat, Skin–Moderate
HE16	Irritation-Eyes, Nose, Throat, Skin–Mild
HE17	Asphyxiants, Anoxiants
HE18	Explosive, Flammable, Safety (No adverse effects encountered when good housekeeping practices are followed)
HE19	Generally Low Risk Health Effects—Nuisance particulates, vapors or gases
HE20	Generally Low Risk Health Effects—Odor

Table 3.7: The OSHA Health Codes.

and their treatment. We have serialised 60 out of 1854 concepts from the treatment ontology in N3 format (see Appendix D), so that we provide an example of the contents of an ontology and indicate the scale of an environment ontology. This information contained in this ontology has been taken from the NHS website[10].

To summarise, the number of concepts in each of the ontologies is given in Table 3.8.

[10]NHS Health Information: `http://www.nhs.uk/chq/pages/Category.aspx?CategoryID=72`

Ontology	No. of Concepts
EAC Ontology	1906
Chemical Ontology	1800
HantsFireEngineFleet Ontology	745
ConstructionVehicles Ontology	114
CSI Ontology	3841
HazChem Ontology	1167
Vehicle Ontology	70
Ambulance Ontology	407
Triage Ontology	74
Treatment Ontology	481

TABLE 3.8: The number of concepts in each of the environment ontologies.

In order to share concepts, we identified in Section 2.2.3 that we would use Seidenberg and Rector's basic fragmentation algorithm to generate a fragment of a concept. In more detail, we describe this fragmentation process in Algorithm 5.

We now proceed by describing a use case which steps through how a Fire Brigade agent uses its ontology to deliberate its actions.

3.2.5 Fire Engine Use Case

In order to illustrate an agent's behaviour we give the following example. At the beginning of a RoboCup OWLRescue simulation, a fire brigade agent a is instantiated with an ontology o, where $o = \langle do, eo \rangle$, see Figure 2.1.1 part (a), and a vehicle v. The vehicle is a fire engine and has the equipment {`foam_nozzle`, `jet_nozzle`, `breathing_apparatus`, `body_suit`}. The agent is allocated two target burning buildings b_1 and b_2. The agent's primary goal is to minimise the area of the city which is burnt, and a secondary goal to save as many lives as possible. In order to achieve these goals, the agent prioritises the order to which the buildings are to be attended, and ranks the buildings according to:

1. Whether the chemicals within the building are explosive or volatile. Specifically, if chemicals explode, the fire spreads more rapidly and increases the area of the city which is burnt;

2. The number of people in the building, and their 5 category triage classification. For example, if the building contains more red patients than any of the another buildings, then it may choose to extinguish this fire earlier than buildings with fewer civilians inside.

Given these goals, the agent first analyses the information received about the buildings (B), in particular, the agent receives the ID for each building (b). Using a building ID,

Algorithm 5 Algorithm to create a fragment of an ontology based on a concept, from Seidenberg and Rector (2006).

Require: $c \leftarrow$ `target concept`

Function: *getParents(concept):* `return set of parents of a concept`
Function: *getLinks(concept):* `return set of links across an ontology from a concept`
Function: *getChildren(concept):* `return set of children of a concept`

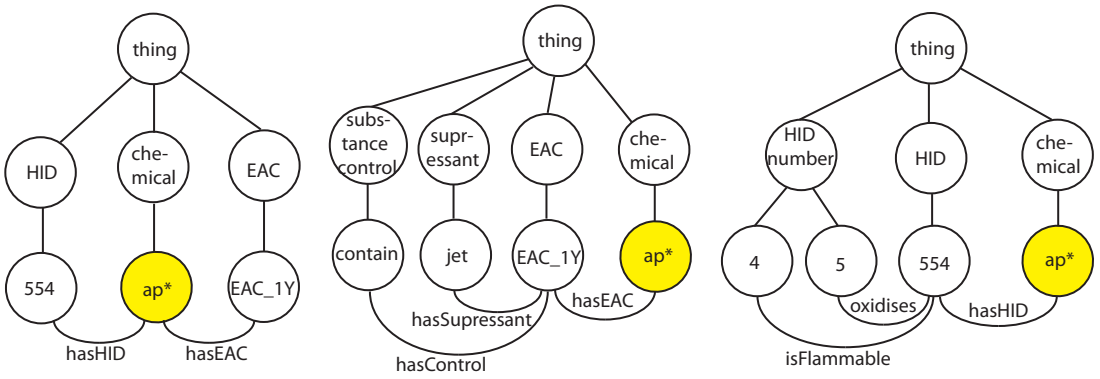
```
1:  Visited ← ∅
2:  ToVisit ← {c}
3:  /* traverse from concept up to the root by */
4:  /* looping through all concepts in ToVisit */
5:  for concept ∈ ToVisit do
6:    Visited ← Visited ∪ { concept }
7:    /* loop through all parents of this concept */
8:    for parent ∈ getParents(concept) do
9:      /* visit the parent if it hasn't already been visited */
10:     if parent ∉ Visited then
11:       ToVisit ← ToVisit ∪ { parent }
12:     end if
13:   end for
14:   /* traverse links across the ontology */
15:   for link ∈ getLinks(concept) do
16:     /* visit the linked concept if it hasn't already been visited */
17:     if link ∉ Visited then
18:       ToVisit ← ToVisit ∪ { link }
19:     end if
20:   end for
21: end for
22: /* traverse from concept down to the leaves */
23: ToVisit ← getChildren(c)
24: /* loop through all concepts in ToVisit */
25: for concept ∈ ToVisit do
26:   Visited ← Visited ∪ { concept }
27:   /* loop through all children of this concept */
28:   for child ∈ getChildren(concept) do
29:     /* visit the child if it hasn't already been visited */
30:     if child ∉ Visited then
31:       ToVisit ← ToVisit ∪ { child }
32:     end if
33:   end for
34: end for
35: return Visited
```

the agent can observe the set of chemicals contained within the building, and find any trapped civilians (referred to with an ID). In order for the fire brigade agent to plan its actions it takes the following steps:

1. Retrieve the chemicals contained in b_1 and b_2, which contain the sets of chemicals: {`ammonium_perchlorate`} and {`calcium_cyanide`}, respectively;

2. Then a checks whether its ontology contains the chemicals in the set {`ammonium_perchlorate`, `sodium_cyclamate`}, using the *refersto* predicate. In this example, o does not contain either of these chemicals;

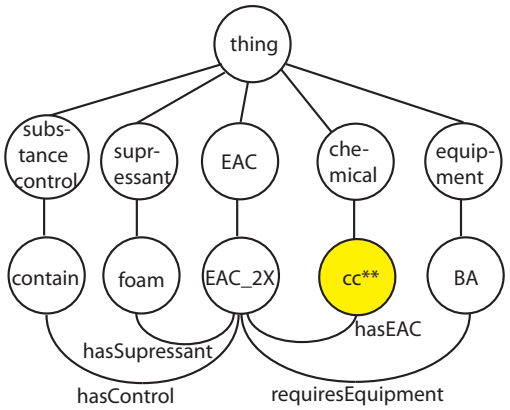
3. The agent then requests for fragments from the environment ontologies which contain {`ammonium_perchlorate`, `sodium_cyclamate`}. The environment ontologies which contain these concepts return a fragment of their ontology, Figure 3.11 shows the fragments from three ontologies (HazChem, EAC and Chemical ontologies) representing `ammonium_perchlorate`, and Figure 3.12 shows the fragment returned from the EAC ontology which represents `calcium_cyanide`. In Figures 3.11 and 3.12 the shaded nodes indicates the concept which the fragment represents;



* for illustration purposes ammonium perchlorate has been abbreviated to ap, this does not reflect the ontology where the concept label is ammonium perchlorate.

FIGURE 3.11: Three fragments representing the concept `ammonium_perchlorate` from the CSI ontology, EAC ontology and Hazchem ontology, respectively.

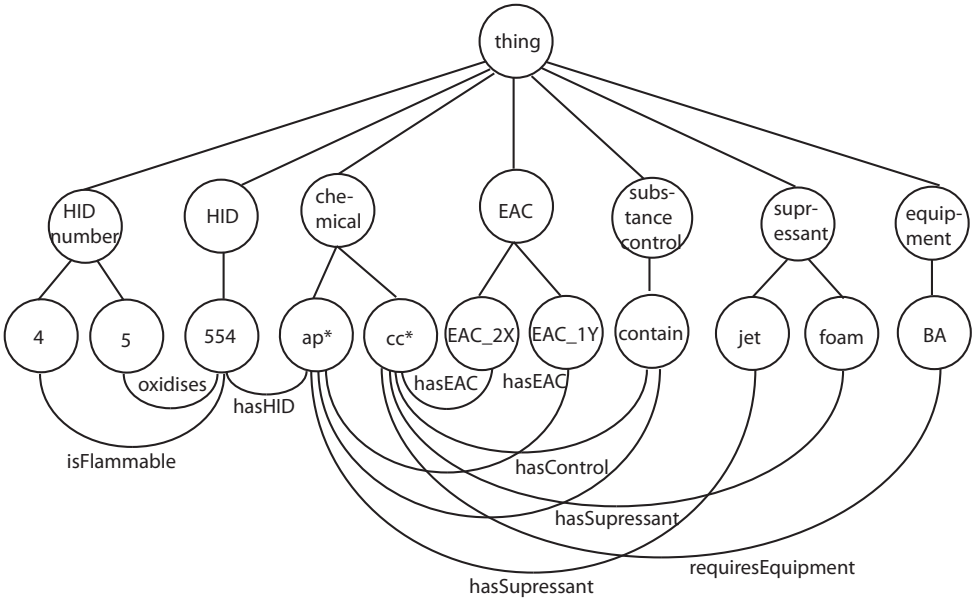
4. The received fragments are then augmented into the agent's ontology (eo), this enables the agent to reuse this information for future tasks if required without sending another request to the environment ontologies. It is possible that merging a fragment with the agent's ontology will cause it to contain inconsistent axioms about concepts and thus provide incorrect information about instances. When merging ontologies, it is therefore important to verify that this does not occur. However, this is not the focus of our work; we focus instead on selecting concepts to augment into an ontology. Therefore, our experiments use a closed world where we have *a priori* verified that the union of all of our ontologies are consistent, and



** for illustration purposes calcium cyanide has been abbreviated to cc, this does not reflect the ontology where the concept label is calcium cyanide.

FIGURE 3.12: A fragment representing the concept `calcium_cyanide`.

therefore no post-merging verification is required. In order to ensure that a merged ontology is consistent in an open world where the content of received fragments cannot be controlled or predetermined, a post-merging verification technique is required (Haase and Stojanovic, 2005a). However, Haase and Stojanovic do not recommend a specific algorithm for selecting this subset, nor is there a standard algorithm. In Figure 3.13 we show a's augmented ontology;



* for illustration purposes ammonium perchlorate has been abbreviated to ap, this does not reflect the ontology where the concept label is ammonium perchlorate.

** for illustration purposes calcium cyanide has been abbreviated to cc, this does not reflect the ontology where the concept label is calcium cyanide.

FIGURE 3.13: The agent's ontology after augmenting the fragments.

5. From its ontology the agent finds that it requires a vehicle that can suppress fire with water for b_1 and foam b_2, and breathing apparatus for b_2. In addition to

this information, the agent's ontology contains information from the HazChem ontology which indicates that the chemical `ammonium_perchlorate` is a strong oxidiser and thus is potentially explosive. The agent plans to suppress the fire in b_1 first, then b_2;

6. The agent checks its vehicle whether v has the required equipment to handle both fires. In this case, the agent's vehicle has the required equipment. In the counter case, the agent will respond in a similar way to steps 3 and 4, the agent sends a request to the environment ontologies for fragments about fire engines with the required equipment. Once it has learnt which vehicles contain the required equipment and if that vehicle is present in the fire station it can change its vehicle;

7. The agent then plans the route and uses the required equipment to extinguish the fires in b_1 and b_2.

The following section introduces state-of-the-art and control, learning and forgetting approaches. We use these approaches to provide comparison benchmarks approaches for learning and forgetting algorithms.

3.3 Benchmark Learning and Forgetting Algorithms

In this section, we describe algorithms we use to benchmark our learning and forgetting algorithms. These benchmarks are required so that we can evaluate our algorithms against state-of-the-art and basic approaches which augment and prune ontologies. To this end, the following two sections detail learning and forgetting algorithms which we use to benchmark our approach, respectively.

3.3.1 Learning Benchmark Algorithms

In order to evaluate our learning algorithms, we use the following techniques as a comparison. We have chosen these algorithms because they are: the state-of-the-art learning approaches (see Section 2.3); and control approaches.

1. *Learn-Repeated*: This algorithm incorporates all concepts and their relationships into its ontology which are required more than once. This aims to offset the cost of learning a concept compared with its use. It demonstrates how an agent would learn if it had perfect foresight, and is used in this evaluation as in Winoto et al. (2002) (see Section 2.3);

2. *Learn-Concept*: This algorithm incorporates all axioms into its ontology that are used to describe the concept being queried. This is a comparable technique with

the agent approaches of Bailin and Truszkowski (2002), Afsharchi et al. (2006), and Soh (2002), which learn a single concept at a time (see Section 2.3);

3. *Learn-Everything*: This algorithm incorporates all axioms which it receives in its ontology. It is designed to show an agent's potential ontology size and complexity and is a comparable technique with the agent approach of Yu and Singh (2002) (see Section 2.3.1);

4. *No-Collaboration*: This is a control approach that does not collaborate with the environment ontologies and highlights the need for learning new concepts.

We put forward these four approaches: *learn-repeated*, *learn-concept*, *learn-everything*, *no-collaboration*, as benchmarks for the performance of our learning approaches, *learn-fragment* and *learn-Markov*.

3.3.2 Forgetting Benchmark Algorithms

In order to evaluate our forgetting algorithm, we use the following techniques as a comparison. We have chosen these algorithms because they are: the state-of-the-art forgetting approaches (see Section 2.2.2); and control approaches.

1. *Forget-Concept*: This approach removes all concepts and relationships related to the selected concept. This technique removes one concept at a time, and is comparable to the techniques presented by Eiter et al. (2006) and Wang et al. (2008) (see Section 2.2.2);

2. *Forget-Tree*: This approach extends the above approach by selecting a subtree from the hierarchy of concepts in the agent's EO. The subtree is selected by comparing the weighting used for each concept (see Algorithm 6). Removing a connected subtree can result in removing a subtree, branch, or extraction of a subtree (Shasha and Zhang, 1997), as shown in Figure 3.14 (see Section 2.2.2);

Algorithm 6 Algorithm used to select the concepts which are connected by their concept weighting, to form a subtree, to be pruned from an agent's EO.

Function: *parents(concept):* `return a set of the parents of the concept`
Function: *children(concept):* `return a set of the children of the concept`
Function: *getLowestWeightedConcept(concepts):* `return the concept with the lowest weighting`
Function: *getWeight(concept):* `return the weight of the concept`

Require: $t \leftarrow 10$
Require: $ConceptsToRemove \leftarrow \emptyset$
Require: $CH \leftarrow \emptyset$
Require: $P \leftarrow \emptyset$

1: `/* find the concept with the lowest concept weighting */`
2: $c_t \leftarrow getLowestWeightedConcept(concepts(EO))$
3: $w_{c_t} \leftarrow getWeight(c_t)$
4: $CH \leftarrow children(c_t)$
5: `/* loop through the children of` c_t `*/`
6: **for** $ch \in CH$ **do**
7: `/* add to the concepts selected to be removed if this concept's weighting is within the threshold` t `*/`
8: **if** $|w_{ch} - w_{c_t}| \leq t$ **then**
9: $ConceptsToRemove \leftarrow ConceptsToRemove \cup \{ch\}$
10: `/* recurse to the grandchildren */`
11: $CH \leftarrow CH \cup \{children(ch)\}$
12: **end if**
13: **end for**
14: $P \leftarrow parents(c_t)$
15: `/* loop through the parents of` c_t `*/`
16: **for** $p \in P$ **do**
17: `/* add to the concepts selected to be removed if this concept's weighting is within the threshold` t `*/`
18: **if** $|w_p - w_{c_t}| \leq t$ **then**
19: $ConceptsToRemove \leftarrow ConceptsToRemove \cup \{p\}$
20: `/* recurse to the grandparents */`
21: $P \leftarrow P \cup \{parents(p)\}$
22: **end if**
23: **end for**
24: **return** $ConceptsToRemove$

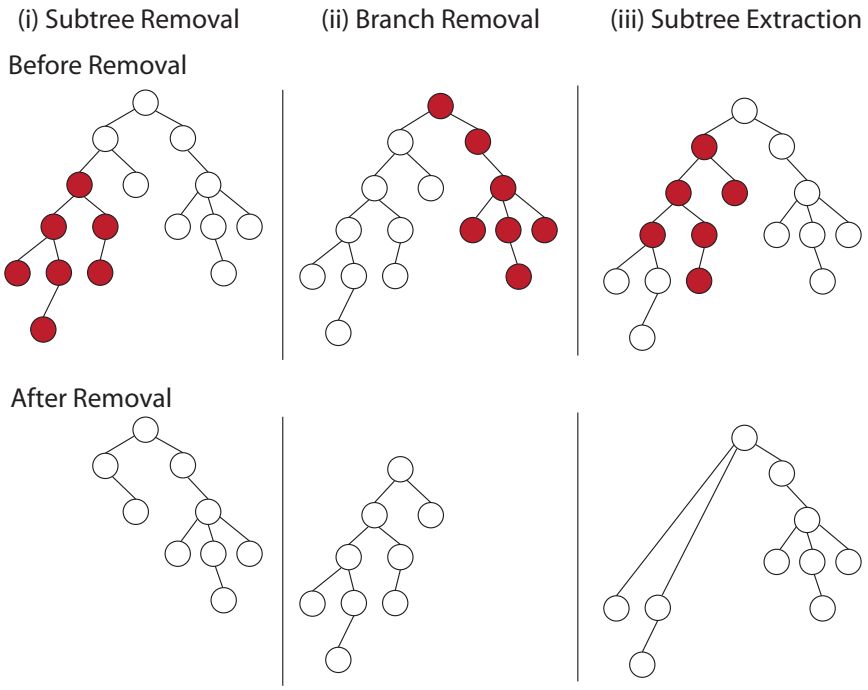


FIGURE 3.14: Subtree Removal, Branch Removal and Subtree Extraction, where the highlighted nodes are removed from the graph.

3. *Forget-Redundant*: This approach removes all concepts that are not used in future queries. We provide a list of the future queries to the agent at the start of the simulation, which have been recorded on a dummy run using the same random seed. This is the only agent that requires a complete list of future queries. This agent does not evaluate whether the time taken to assess its ontology is stopping it from making its actions in a timestep, because it checks at the end of each timestep whether it can forget any of its concepts. It demonstrates how an agent could forget if it had perfect foresight, and is used in this evaluation as in Winoto et al. (2002) (see Section 2.3);

4. *Forget-Nothing*: This is a control approach that does not implement a forgetting algorithm and highlights whether forgetting concepts is beneficial to an agent.

We put forward these four approaches: *forget-concept*, *forget-tree*, *forget-redundant*, *forget-nothing*, as benchmarks for the performance of our forgetting approach, *forget-fragment*.

So far, we have presented our agent framework, RCOR, and approaches that we can benchmark our learning and forgetting algorithms against. We now need to consider how to compare these approaches. Thus, in the following section, we start by presenting an investigation into measures which can be used to evaluate the complexity of an evolving ontology.

3.4 Evaluating the Complexity of Evolving Ontologies

This section details an investigation into measures for the complexity of an evolving ontology. We require a measure for measuring the complexity of an evolving ontology in order to compare the effect of different techniques that aim to augment and to prune the number of concepts and axioms in an ontology. Specifically, our aims (see Requirement 2, in Section 1.3.2) require that learning and forgetting algorithms result in ontologies that optimise the overhead costs of agents. One of the ways that overhead resource costs can rise is through a high level of complexity in an ontology, because it raises the costs to host, manage, and use the ontology. Similarly, if an agent wishes to use a reasoner with its ontology, the costs of doing so will also rise with the level of complexity. Thus, being able to measure the complexity of an ontology allows us to compare this aspect of our approach against the benchmark approaches.

The complexity of an ontology may be referred to in terms of space or time complexity. In our case, space complexity is the worst case computer memory space required to infer knowledge, and time complexity is the worst case time required to compute inferences from an ontology. A measure that indicates the complexity of an ontology enables us to determine which ontology costs the least to use in terms of memory and time, and can be used to evaluate the effectiveness of using an ontology.

The investigation used the RoboCup OWLRescue framework to provide a learning environment (see Section 3.2), and compared the *learn-fragment*, *learn-concept*, *learn-everything*, *learn-repeated*, and *no-collaboration* approaches (presented in Section 3.3) so that we could evaluate the complexity measures discussed in Section 2.5.2. We ran the RCOR simulation for 2000 timesteps and recorded the concrete measure, the time required to load and consistency check an ontology using the Pellet reasoner (as described in Section 2.1.3), and three abstract complexity measures: the number of concepts in the ontology; the number of paths contained in the ontology; and the μ, ρ and σ measures defined by Yang et al. (2006) (see Section 2.5).

We conclude that the concrete measure of the time to load and consistency check an ontology, and the abstract measure of the number of concepts in an ontology both give equivalent results which estimate the complexity of an ontology. Out of these two measures, the total number of concepts is cheap to calculate, whereas loading and consistency checking an ontology is a process that becomes extremely slow when ontologies are large, and thus calculating this every timestep for every agent in our simulations is intractable. Therefore, we will use the number of concepts in an ontology as a measure, which we will hereafter refer to as the *estimated complexity*.

3.4.1 Ontology Complexity Measures

In order to evaluate the complexity of an evolving ontology, we consider two types of measures, abstract and concrete:

Abstract Costs: In order to investigate which measures can be used to estimate the time and space complexity of an ontology, we consider eight different measures that affect the complexity of an ontology. These eight different measures have been suggested by the literature in Chapter 2 (see Sections 2.1 and 2.5) and while these measures were effective for evaluating the complexity of specific ontologies, they have not been evaluated with ontologies with different measures (such as ontologies with different numbers of relationships, average branching factors, and ontology depths). We require a measure that can capture the complexity of an evolving ontology so that we can compare the complexity of different approaches.

The class structure of an ontology is similar to the structure of a type system, which may contain similar hierarchies. In addition, a type system associates types with values and aims to ensure that functions return values of expected types through the use of type inference and type checking. Both type inference and type checking in programming languages have similarities with classification and consistency checking in ontology inferencing. We require a measure to evaluate the complexity of an ontology, and considered existing *ad hoc* measures for ontology complexity (discussed in Sections 2.1.3 and 2.5.2) and whether the type systems community had contributed complexity measures. We considered type systems' measures because they are comparable to ontologies and the type checking tests they run are similar to those used in ontology inference. However, we were unable to find a general-case measure of complexity for type inference or type checking from within the type system community, and thus we were not able to use type systems' measures to generate complexity measures for ontology inferencing.

It is desirable for an agent to have an ontology with a low ontology complexity so that it can derive logical entailments from its ontology quickly. Thus, during our empirical evaluation we will record the following abstract costs:

1. The average path length of an ontology o, P_o (Yang et al., 2006);
2. Total number of relationships in ontology o, R_o (Yang et al., 2006);
3. Average number of relationship per concept (μ), in ontology o, calculated by: $\frac{|concepts(o)|}{R_o}$ (Yang et al., 2006);
4. Average paths per concept (ρ), in ontology o, calculated by: $\frac{|concepts(o)|}{P_o}$ (Yang et al., 2006);
5. Ratio of maximum path length to average path length (σ), of an ontology o, calculated by: $\frac{maximum(P_c)}{P_o}$ (Yang et al., 2006);

6. The average branching factor of ontology o, abf (Bao and Honavar, 2005);
7. The number of concepts contained in the ontology o, where $|concepts(o)|$ (Corcho and Gómez-Pérez, 2000);
8. Number of axioms, $|o|$, contained in the ontology o (Corcho and Gómez-Pérez, 2000).

Specifically, we consider the depth and branching factor because these factors affect the time required to perform a breadth-first search over a graph (see Section 2.1). The number of concepts and the types of constructs used to express an ontology also affect the time required to infer logical entailments from the ontology (see Section 2.1). Abstract measures 3–7 correspond with those used in Section 2.5 to compare the complexities of an evolving ontology;

Concrete Costs: We also evaluate concrete costs which are real-time measures: the time taken (in CPU time) to load the ontology into a reasoner and evaluate it by checking that it is consistent. These two factors are used because they model two functions that can be repeatedly performed on an ontology to provide an estimate of the complexity without the need for the lengthy process of complete inference extraction from an ontology. The loading time of an ontology increases monotonically with the size of an ontology; additionally, the consistency check of an ontology requires the inspection of each axiom in an ontology, and the time taken increases with the complexity of an ontology. In particular, loading an ontology into a reasoner is a prerequisite for its use, and while this is not normally required for every inference operation, it occurs whenever the ontology changes. In our case we are evolving an ontology, and as such, the ontology is reloaded into the reasoner more regularly than in typical applications that utilise static ontologies. Ontology loading time is also measured using CPU time in milliseconds as the unit of measure. We ran our experiments on Iridis 3 compute nodes with 2 x 4-core 2.27 GHz Intel Nehalem processors and 22GB of RAM running Linux.

In the next section, we use the costs described above to evaluate the complexity of an evolving ontology.

3.4.2 Results

We start by first analysing the time to load and consistency check an ontology, shown in Figure 3.15. This graph shows that the *learn-repeated* approach requires the least amount of time (out of the agents that augment their ontologies), for a reasoner to load and check the consistency of our ontology. The agents using the *learn-concept* and *learn-fragment* approaches are similar in the amount of time to check the consistency of their ontologies. The agents using the *learn-repeated* approach require the least amount of time to load

and check the consistency, because it only learns concepts that are repeated and therefore augments fewer concepts then other approaches (this is supported in Figure 3.15). The *learn-concept* and *learn-fragment* approaches require the next least amount of time to load and check the consistency of the EO. This is because they both augment concepts based on the encountered concepts. They contain more concepts than the previous approaches, and therefore there are more paths through the ontology (see Figure 3.17). The *learn-everything* approach required the most amount of time to load and check the consistency of the EO (see Figure 3.15), and this ontology contains the most concepts and has the most number of paths compared with the other approaches (see Figures 3.16 and 3.17).

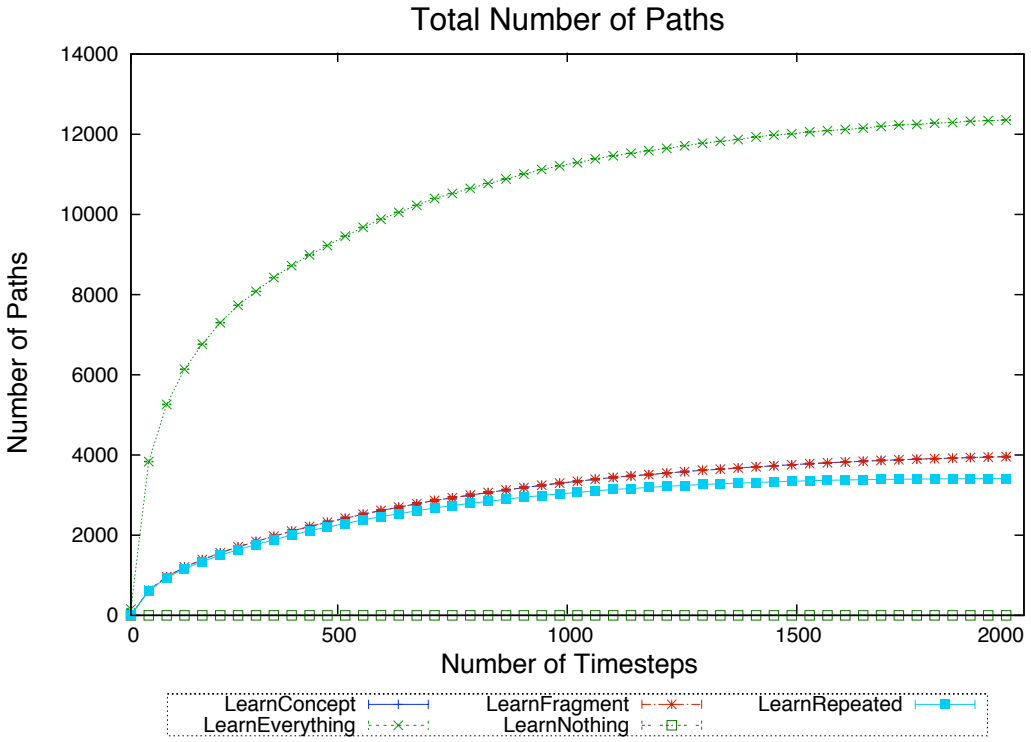


FIGURE 3.17: Graph showing total number of paths in the agents' ontologies.

We now finish by analysing the three abstract measures: the number of concepts in the ontology, shown in Figure 3.16; the number of paths contained in the ontology, shown in Figure 3.17 ; and the μ ρ and σ measures, shown in Figures 3.18, 3.19 and 3.20.

The total number of concepts, as shown in Figure 3.16, illustrates the number of concepts in the task agents' ontologies, which correlate to the number of concepts learned by that agent's approach. The *learn-nothing* approach has the fewest concepts, because it learns nothing. The next approach with the least number of concepts is the *learn-repeated* approach, which has an optimal learning strategy of learning only those concepts which will be used again, resulting in the agent skipping concepts that it will not need to use again. The next lowest approaches are *learn-fragment* and *learn-concept* which

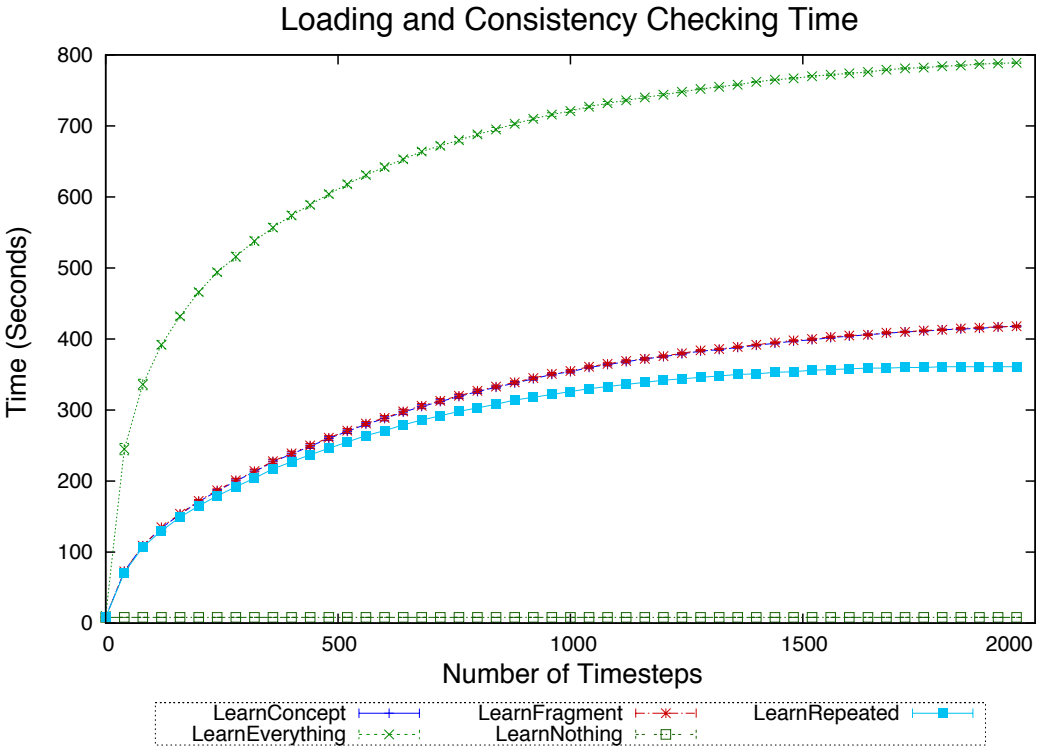


FIGURE 3.15: Graph showing the time taken to load and consistency check of all the task agents' ontologies.

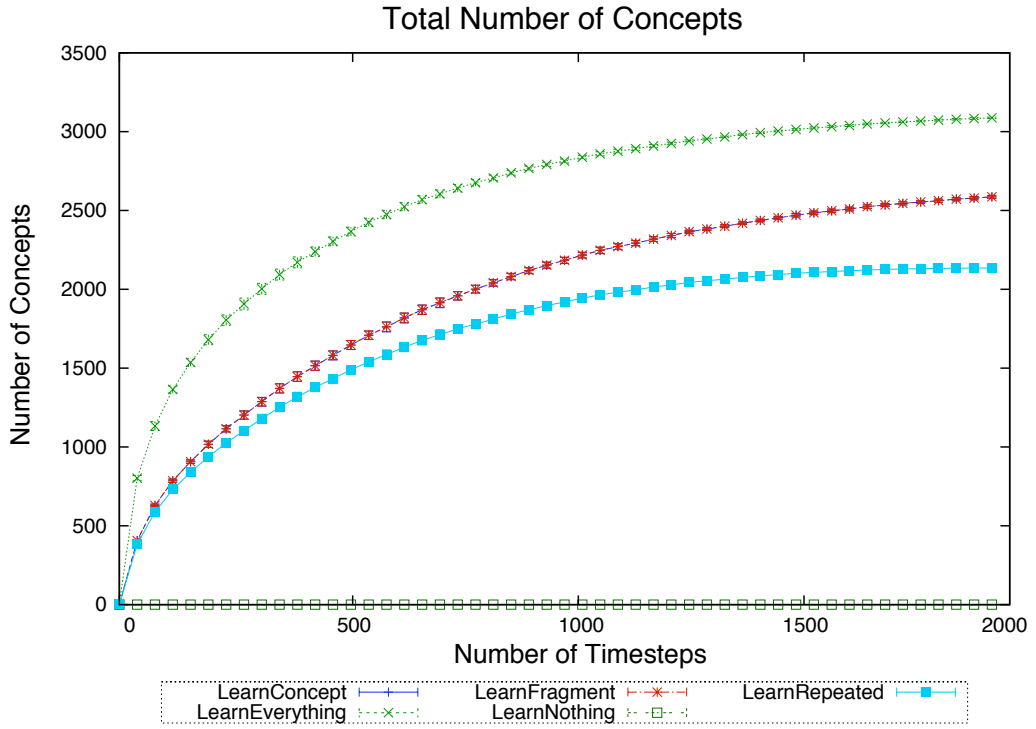


FIGURE 3.16: Graph showing the average total number of concepts in the agents' ontologies.

have a similar number of concepts. Specifically, the number of concepts in the *learn-fragment*'s approach is higher than the *learn-concept* approach, this is because the *learn-fragment* learns more than one concept compared to the *learn-concept* approach. The *learn-everything* approach is the least efficient learning strategy (in terms of number of concepts), since it learns all concepts it receives.

We illustrate the total number of paths in Figure 3.17, where a path is defined as a unique route, through superclasses, from a concept to the root node. The total number of paths is a measure of estimated ontology complexity that takes into account the structure of the class hierarchy, as an additional estimation to the number of concepts. We note that of the learning approaches, the *learn-repeated* approach maintains the lowest number of paths overall, with *learn-fragment* and *learn-concept* approaches containing the next highest number of paths. These three approaches have the same number of paths up to approximately 2,000 timesteps, whereby the *learn-repeated* approach starts to encounter concepts that are not repeated in the future, and hence does not learn them, resulting in a lower number of paths.

Figures 3.18, 3.19 and 3.20 present the values of μ, ρ, and σ, respectively, for each approach, as shown by Yang et al. (2006). These results indicate that learning everything has the highest values for σ and ρ compared with the other approaches, thus indicating that it has the most complex ontology. We can also identify that the *learn-nothing* approach has the least complex ontology because σ, μ, and ρ are all 0, whereas the other approaches have varied values. The *learn-fragment*, *learn-concept*, and *learn-repeated* approaches have the same trends, ρ is 0, σ is 1, and μ increases in the first 100 timesteps from 0 to 1. This increase corresponds with the increased number of relationships that are added to the agent's evolving ontology. We note that after 100 timesteps the complexity of all the approaches does not change, with the exception of the *learn-everything* approach which does not change after 200 timesteps, despite the fact the ontologies are still increasing in size and require more time to load and check the consistency. We can determine from this finding that these three measures do not distinguish which ontology is more complex from the *learn-fragment*, *learn-concept*, and *learn-repeated* approaches. At this time, there are also no rules or guidelines (as discussed in Section 2.5) on how to interpret these measures, and because of these issues we do not use them to compare and evaluate our approach.

3.4.3 Investigation Summary

In this section, we have investigated methods to determine the complexity of an ontology, in order to compare different learning and forgetting approaches. We conclude that the concrete measure of the time to load and consistency check the ontologies, and the abstract measures of the total number of paths and total number of concepts in the ontologies, indicate the complexity of evolving ontologies compared with different

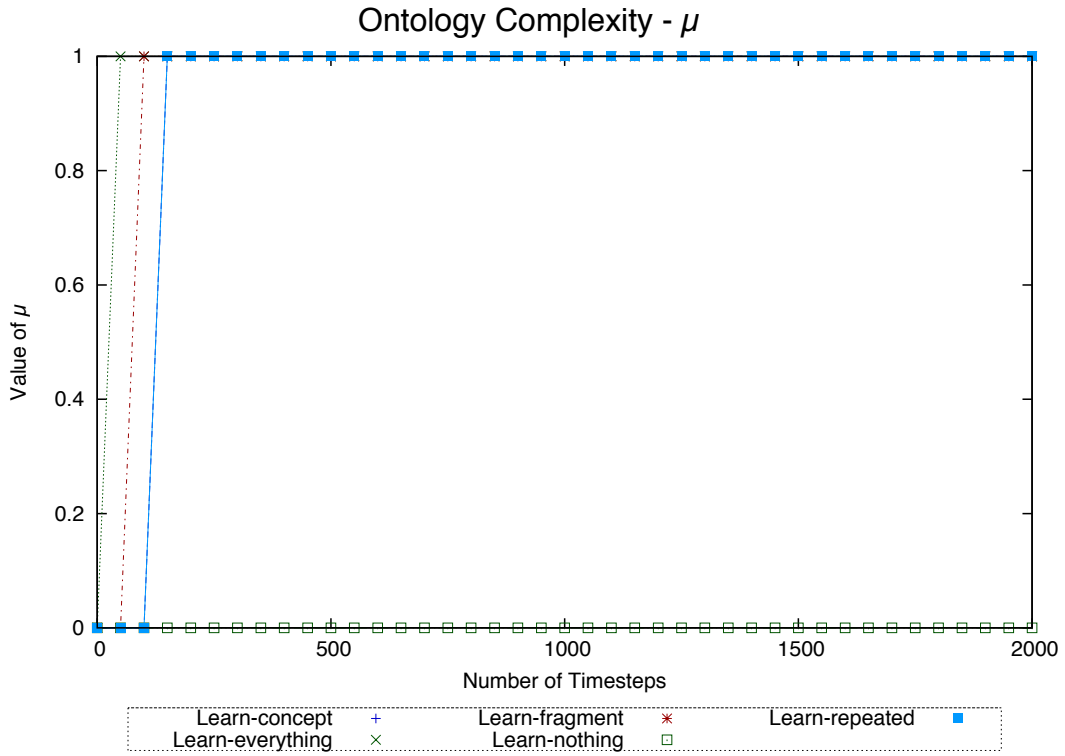


FIGURE 3.18: A graph showing the average μ value from the agents' ontologies.

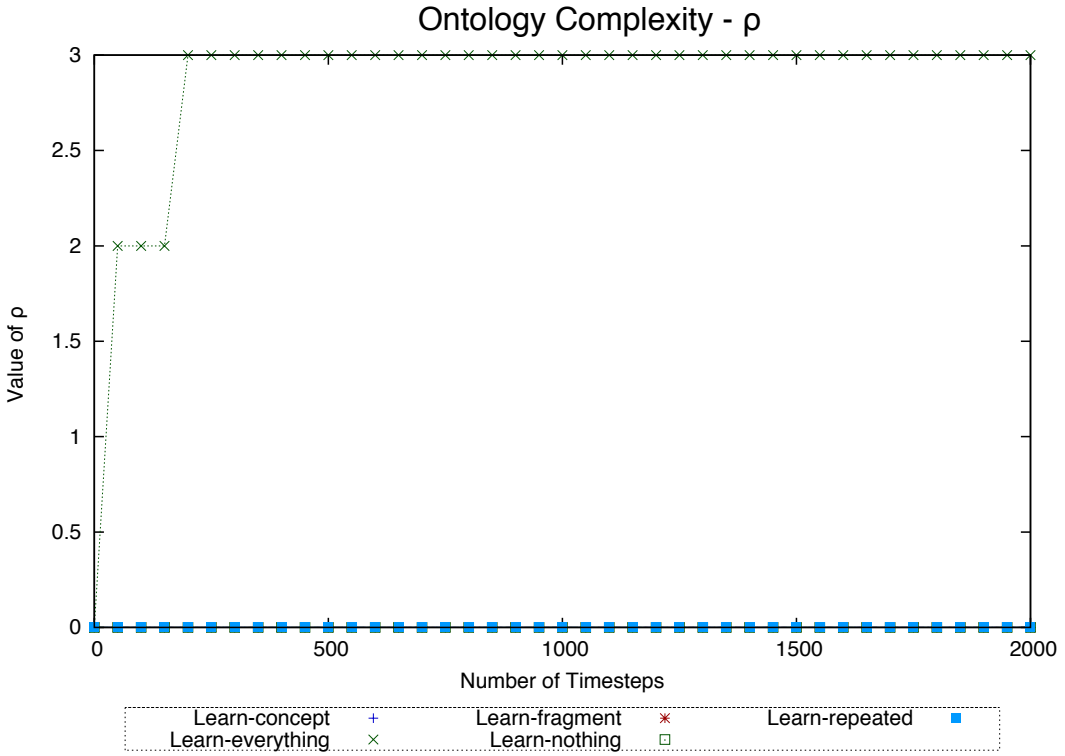


FIGURE 3.19: A graph showing the average ρ value from the agents' ontologies.

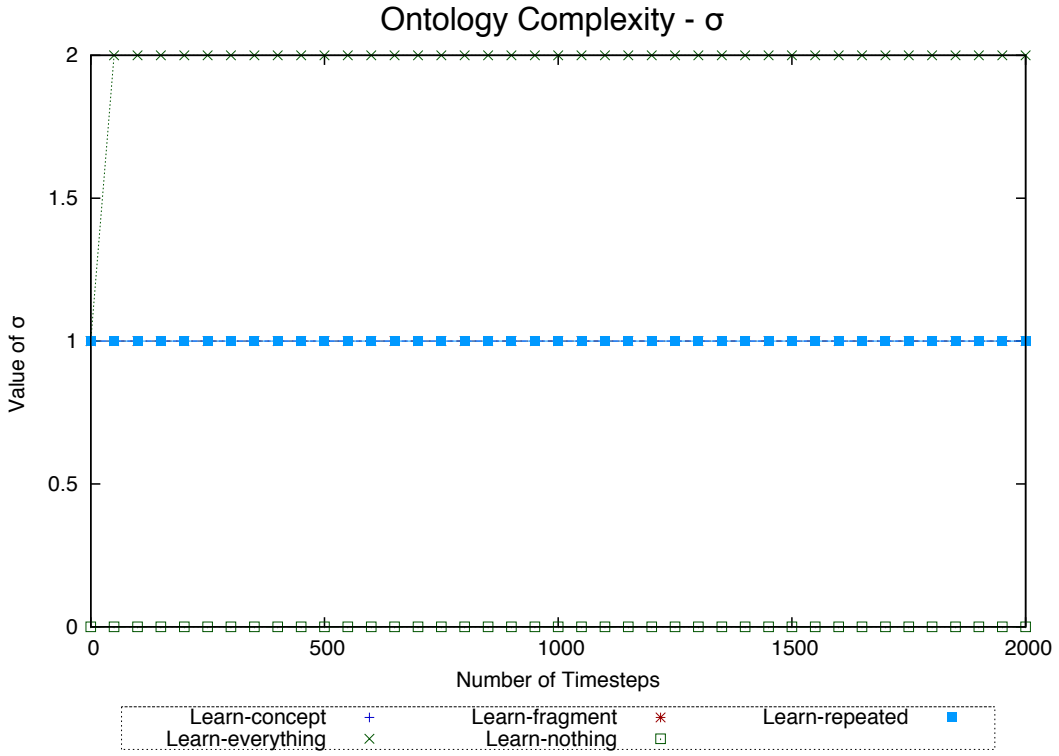


FIGURE 3.20: A graph showing the average σ value from the agents' ontologies.

evolution approaches. These measures provide a rank order of the ontologies evolved using different approaches, and derive the same ranking for the RCOR agent's ontologies. We therefore look to the overhead of each of these approaches to decide which to use in our experiments. The number of concepts in the ontology has the least overhead, because it is the simplest operation, since we need to only count the number of concepts in the ontology. Calculating the number of paths in an ontology is more complex, since it requires traversing the concept tree multiple times. The largest overhead cost is experienced with loading and consistency checking an ontology, which requires serialising the ontology, saving it to disk, reloading it into memory and running multiple passes over it to consistency check it. Given that we have found that these three measures give the same ordering of ontologies, we will therefore measure the ontology complexity using the number of concepts in the ontology throughout our experiments in order to compare the effect on the resources required to host, manage and use agents' ontologies that have been augmented and pruned using our approaches and the benchmark approaches.

3.5 Evaluation Measures

In this section, we present the measures which we will use to evaluate online learning and forgetting algorithms. We categorise these measures as one of two types, first the standard measures to evaluate an agent's performance in terms of the RoboCup Rescue

simulation, and second measures which evaluate an evolving ontology in terms of time, operations which are performed on the ontology, and an ontology's complexity. To this end, the following two sections detail which measures we will use to evaluate agents' performances using the RoboCup Rescue score vector, and measures which evaluate evolving ontologies, respectively.

3.5.1 RoboCup Rescue Score Vector

In order to evaluate the performance of an agent with respect to RoboCup Rescue we use RoboCup Rescue's score vector (presented in Table 2.19, in Section 2.5). Specifically, we will focus on score vectors A, D, E, F, and G which indicate an agent's state, ratio of civilians in a refuge, ratio of civilian rescued, proportion of the city that is not destroyed, and the ratio of fires extinguished, respectively. We have chosen to omit from our evaluation score vectors B, C, and H, which indicate the time an agent spends travelling, the number of messages passed, and the time taken for an agent to complete its actions. This is because our framework and algorithms are designed to enable an agent to use appropriate equipment to rescue their targets in order to achieve the best outcome in the scenario, and we do not focus on routing agents to their targets or the number of messages sent to co-ordinate the rescue of the targets. In more detail, we use the score vectors shown in Table 3.9 which provides the algorithms used to calculate them.

Score Vector	Equation
A1	$\sum c : c \in civilians\ and\ 0 \leq c.HP() \leq 10$
A2	$\sum c : c \in civilians\ and\ 11 \leq c.HP() \leq 40$
A3	$\sum c : c \in civilians\ and\ 41 \leq c.HP() \leq 70$
A4	$\sum c : c \in civilians\ and\ 71 \leq c.HP() \leq 100$
D	$C_{ref} = \frac{CiviliansInRefuge}{TotalCivilians}$
E	$C_{res} = \frac{CiviliansRescued}{CiviliansRescued+CiviliansBuried}$
F	$A_{undestroyed} = \frac{TotalUnburntArea}{TotalBuildingArea}$
G	$F_{er} = \frac{FiresExtinguished}{FiresExtinguished+BuildingsOnFire}$

TABLE 3.9: The score vectors and the equations used to calculate them, where $HP()$ is a function that returns the health of a civilian.

3.5.2 Measures to Evaluate an Evolving Ontology

In order to evaluate our technique and the benchmark algorithms, we consider measures that are abstract (for example, the number of message pairs, the number of classes in an ontology, and the size of messages), and concrete (for example, the actual time taken

to complete processes). We measure abstract and concrete costs separately in order to ensure that differences between the performance of different algorithms are not down to implementation-level differences in parts of the system that we cannot control. For example, if there were specific optimisations on certain types of axiom operations that exist in a library that our framework uses, that may skew the concrete cost, but would not affect the abstract cost. Thus, we measure both the abstract and concrete forms of each measurement to ensure that improvements are based on measurable differences, and not merely runtime differences.

Given these types of measurements, we identify the cost involved with each step of our approach and these costs are identified in Figure 3.21. This figure illustrates the interaction between two actors: a task agent $a_{t,i}$, where $a_{t,i}$'s ontology does not contain c, and collaborates with $a_{s,j}$ with the aim of retrieving a fragment about c.

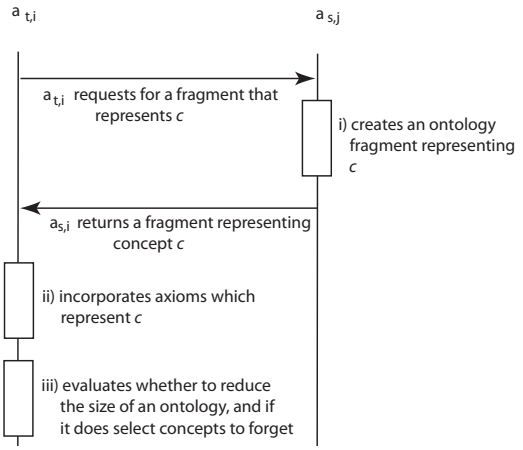


FIGURE 3.21: A sequence diagram showing the messages and costs associated with learning and forgetting

For each message and process indicated by arrows and a box in Figure 3.21 there are associated abstract and concrete costs, which are explained in more detail in the subsequent sections. In addition to the measures described below, we use the number of concepts to rank a set of ontologies according to estimated complexity (as discussed in Section 3.4).

3.5.2.1 Message Sending

Agents send messages to request knowledge, and receive messages containing fragments that represent that knowledge (see Figure 3.21 and Section 1.3). Specifically, messages are only received in response to requests and there is no broadcasting of other information. Therefore, our system assumes that messages are received in sequence (shown in Figure 3.21), and in response to requests. In our experiments, the agents and environment ontologies are contained on the same computer and therefore are not hindered by network costs that may cause slow response times (see Section 1.3)). It is more likely

that network costs associated with sending messages (i.e. bandwidth) will affect agents that operate on an open system compared to a closed one. An agent will receive a message after it has been created and therefore will not need to adopt any strategies for receiving partial information. Similarly, the agents trust all of the responses they receive, and do not have to adopt any strategies that compare responses from different sources in order to verify the quality of information received.

In our scenario, we assume that minimising the number of messages is beneficial to the agent. This is because we focus on reducing the costs associated with augmenting ontologies and there are costs associated with sending messages (identified in Figure 3.21). However, minimising the number of message may not always provide the greatest utility for an agent. In our case our utility is cost, and where cost does not factor into the utility we do not recommend attempting to minimise messages. For example, an agent negotiation framework requires messages to acquire resources at the right price without a limited timeframe, and their utility focuses purely on acquiring resources for the lowest price. Sending more messages to find the lowest price point is more beneficial than limiting the number of messages sent.

Therefore, in our case an agent desires to send the smallest possible number of messages over its lifetime. When an agent augments its ontology with new concepts because of a task, it does not need to communicate with other agents when completing the same task. However, if an agent does not evolve its ontology then it will be required to send more messages to acquire the necessary knowledge. Agents that implement different learning and forgetting strategies may send different numbers of messages over a network. As such, we measure abstract and concrete measures to record the costs of message sending:

Abstract Costs: We consider the following three abstract measures:

1. Number of message pairs;
2. Size of a message in bytes, b (serialised as OWL/XML);
3. Cost to send a byte of data over network, n_b.

The cost of sending a message through a network can vary according to different factors; these include the size of the message, and environment restrictions such as bandwidth and routing protocols. In the case that an environment containing a set of agents is located on the same machine, there is no related network traffic cost. However, in contrast to this case, an environment with agents that are distributed across a network are dependent on the bandwidth available to the agents, thus there is a cost related to sending messages across a network. Additionally, agents may be hosted on a mobile or remote platform where they have limited resources, such as battery power. Such distributed agents may require the use of their battery power for powering wireless communication hardware and for the generation of

message content. Given this background, we have modelled the real-time cost of sending a message in Equation 3.4;

$$Message\ Cost\ =\ b * n_b \tag{3.4}$$

Concrete Costs: We consider the following two concrete costs which capture the actual CPU time taken to perform the sending of a message:

1. Time taken to compose a message;
2. Time taken to send a message.

These measures can be combined to form the overall cost of a message.

3.5.2.2 Generation of Fragments

Fragments are sent by specialist agents in messages, so that a task agent has a choice of concepts to learn. Generating a fragment incurs costs because of the computational time required to generate it, which includes loading and performing a fragmentation algorithm. The deeper the concepts hierarchy of the agent's ontology, and the more relationships contained within that ontology, the longer it takes to generate a fragment. The cost of fragment generation depends on the number of axioms in the ontology and its complexity. In order to determine the cost of generating a fragment, we consider the following abstract and concrete costs:

Abstract Costs: The fragment generation process generates a fragment from an ontology which represents a concept. The following measures are used to generate a fragment and can therefore be used to describe and compare the generation of fragments:

1. Number of concepts that relate to c_t;
2. Number of relationships in c_i;
3. Number of axioms contained in the fragment, f, where the number of axioms is $|f|$.

Concrete Costs: The concrete cost recorded for generating a fragment is the CPU time taken generate a fragment, in seconds.

3.5.2.3 Learning Cost

Once a task agent receives a set of fragments, it can then start the process of augmenting its ontology. The agent can then selectively choose a set of concepts to learn from the

received fragments. The cost of processing fragments and augmenting an ontology is dependent on the number of concepts and axioms contained in the fragments. In order to determine the cost of processing the fragments, we consider the following abstract and concrete costs:

Abstract Costs: We consider the following abstract costs to represent the learning costs for each timestep:

1. Number of fragments received, $|F|$ where F is the set of all fragments received in a timestep;
2. Number of concepts contained in all of the received fragments f_i, in a timestep $\sum_{i=0}^{n} |concepts(f_i)|$;
3. Number of axioms contained in all received fragments f_i, in a timestep $\sum_{i=0}^{n} |f_i|$;

Concrete Costs: We consider the following three concrete costs which capture the actual CPU time taken to perform the agent learning process:

1. Time taken process the fragments;
2. Time taken to select concepts to augment into an ontology;
3. Time taken to augment an agent's ontology.

3.5.2.4 Forgetting Cost

The process of forgetting incurs costs which include: evaluating which concepts to remove, and removing the selected concepts. This cost will differ according to the algorithm used to remove the concepts, thus we consider the following abstract and concrete costs:

Abstract Costs: The abstract cost recorded for forgetting a fragment is measured by the number of concepts removed, where eo is the ontology before forgetting a set of concepts about c, and eo^{-} is the ontology after the concepts have been forgotten:

$$abstract\ cost\ =\ |eo| - |eo^{-}| \tag{3.5}$$

Concrete Costs: We consider the following two concrete costs which capture the actual CPU time taken to perform the agent forgetting process:

1. Time taken to select a set of concepts to remove;
2. Time taken to remove the set of concepts from an ontology.

3.6 Summary

In this chapter, we fulfilled Requirement 1.3.1 which requires a framework that enables agents to evolve their ontology by learning and forgetting concepts. Specifically, we fulfil this requirement with the RoboCup OWLRescue framework which enables agents to collaborate and perform tasks, and enable a standard success measure so that we can evaluate the effectiveness of different learning and forgetting algorithms. We have published our framework in the following two papers: Packer et al. (2010a); and Packer et al. (2010b). In more detail, we identify the four parts of Requirement 1.3.1 and how we have addressed them:

1. **Task Environment:** We require that our framework's environment simulates tasks for agents to complete, and the agents' actions directly affect the outcome of the task. We identified in Section 2.5.3 that RoboCup Rescue fulfils this objective, by using simulators to generate fires, injured civilians, and blockades, for fire brigade, ambulance and police agents to rescue. When agents rescue targets from this simulation it affects the outcome of future events within the environment. However, RCR does not describe in detail the attributes of a situation. For example, an injured civilian has a number of health points which decreases over time, but it does not contain information about the civilians' injuries. Thus, we have developed an extension, RoboCup OWLRescue (see Section 3.2), which describes additional attributes about agents' targets. This provides agents with information that they can deliberate over, in our example this enables an agent to decide what equipment to use to treat an injured civilian.

2. **Agent Model:** We require that our framework represents agents that have their own ontologies to use to decide their actions when encountering tasks, and enable them to evolve their ontologies during their lifetime. We fulfil this through enabling agents to have their own ontologies (see Section 3.2.3) and providing a set of environment ontologies (see Section 3.2.4) to learn from. These environment ontologies contain information which enables agents to save more targets. The agents in the environment can therefore use online learning and forgetting algorithms to evolve their ontologies.

3. **Domain Ontologies contained within the Environment:** We require that our framework contains domain ontologies to provide agents with fragments of ontologies so that they can complete tasks that they could not before. We fulfil this by providing a set of environment ontologies that contain real information about recommendations and guidelines for extinguishing fires and treating civilians (see Section 3.2.4).

4. **Performance Evaluation Measures:** We also require that our framework enables us to evaluate the performance of the agents using learning and forgetting

algorithms, in terms of the outcome of the scenario and comparison measures. We fulfil this by using the RoboCup Rescue score vector (see Section 3.5.1) and using concrete and abstract measures (see Section 3.5.2). We also investigate how to measure the complexity of an ontology, however our investigation showed that state-of-the-art ontology complexity measures were not effective, therefore in order to estimate an ontology's complexity we use the number of concepts contained in the ontology (see Section 3.4). We use this estimated ontology complexity measure to evaluate an ontology's complexity. It is desirable for an agent to have a low ontology complexity because it can access information faster than an ontology with a higher complexity.

This framework enables us to fulfil our first requirement (see Section 1.3.1), however for future work we plan to extend our framework so that it can enable agents to reason over their ontologies. It is necessary to develop and evaluate an agent's approach that can balance the trade off between using a reasoner, which can be costly (see Section 2.1.3), and the cost of instead acquiring the inferred information directly from specialist agents (see Section 7.3).

While the RoboCup OWLRescue framework is currently sufficient for evaluating our proposed ontology evolution algorithms, we also plan to extend it so that it models an open environment (see Section 7.3). In order to model an open environment, we require that the environment ontologies are dynamic. For example, we could use live ontologies from the Web instead of the fixed set of environment ontologies. Also for future work, we plan to evaluate our approach using different modularisation techniques to disseminate information from the specialist agents, so that we can recommend which modularisation technique is the most effective for evolving an ontology.

Given that we have fulfilled Requirement 1.3.1 by developing an agent framework that supports the evaluation of online learning and forgetting algorithms, we proceed in the following chapters to present online learning and forgetting algorithms. In more detail, Chapter 4 presents our reactive learning algorithm for evolving an ontology. We then present our proactive learning algorithm in Chapter 5 for evolving an ontology using prediction, followed by our forgetting algorithm, presented in Chapter 6.

Chapter 4

A Reactive Learning Algorithm for Evolving Ontologies

This chapter describes our reactive learning algorithm which augments an agent's ontology with concepts from a set of fragments representing a target concept. It is a reactive algorithm because it is used when an agent encounters a task which requires the definition of a concept. Our algorithm focuses on reducing the costs incurred by the acquisition of knowledge and the evolution of an agent's ontology. To this end, we aim to minimise the costs associated with: acquiring regularly required knowledge; and hosting, managing and using the ontology. Thus, we develop an approach that enables an agent to learn, therefore minimising the costs of acquisition. In this chapter, we also evaluate this algorithm against control approaches and other state-of-the-art online learning algorithms to provide a comparison. Thus, this chapter fulfils part *a*) of our second and third requirements (see Section 1.3) which requires the use of an online learning algorithm to select which concepts to augment into an agent's ontology, and an evaluation of this approach, respectively.

To this end, the next section provides motivation for using a reactive online algorithm to select concepts to augment into an ontology. Then in the following section, we detail our online reactive learning algorithm. Following that, we evaluate our algorithm against other benchmark algorithms and then in the next section we conclude. A glossary of the terms used and those introduced in this chapter are given in Appendix F.

4.1 The Motivation for Learning

In order to motivate the need for augmenting an ontology (as introduced in Section 1.1.1), we consider the following:

1. Increasing an agent's vocabulary will enable it to complete tasks that it could not before, thus increasing the number of tasks it can complete and its utility;

2. Augmenting an agent's ontology removes the need for re-acquisition of regularly required concepts. Thus reducing the acquisition cost, compared to acquiring the knowledge that is regularly required each time;

3. Augmenting an agent's ontology selectively will decrease the ontology's potential size because an agent does not require all of the concepts contained in the environment to complete a task. Therefore, an agent will encounter fewer costs associated with hosting, managing, and using its ontology compared to approaches that do not learn selectively;

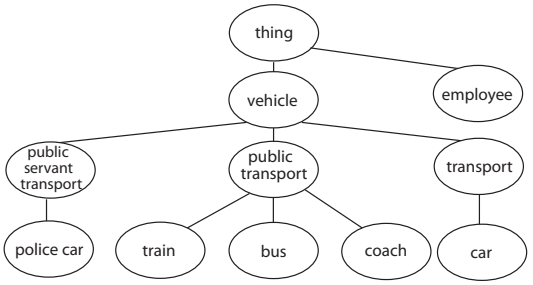
4. An agent will be able to make new inferences over implicit knowledge contained in its ontology.

While an agent can benefit from augmenting its ontology, it can also be hindered by hosting, managing and using large ontologies (see Section 2.2.2). Therefore, an agent may wish to be selective in the concepts that it augments into its ontology. We hypothesise that selectively augmenting an agent's ontology with concepts counteracts the disadvantages of having a large ontology, and enables the agent to complete new domain tasks. Thus, we proceed to detail our reactive learning algorithm.

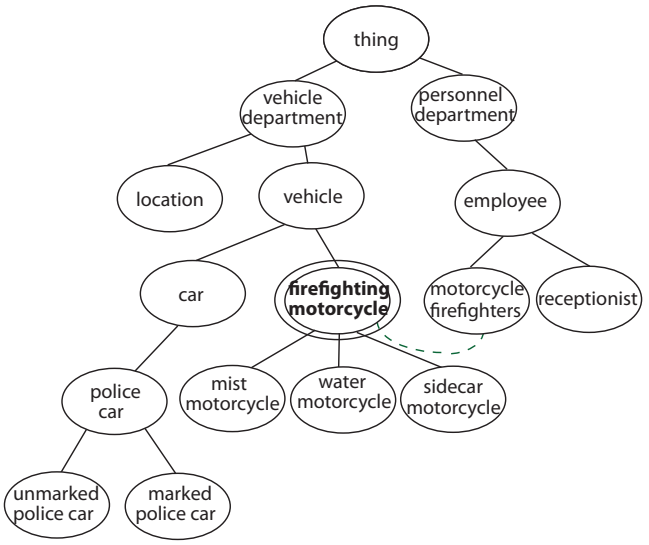
4.2 The Reactive Learning Algorithm

A rescue agent aims to save a target and requires specific knowledge to do so. This knowledge is contained in a private ontology, which is composed of two distinct ontologies: a Domain Ontology (DO) and an Evolving Ontology (EO) (see Section 3.1). The DO contains the axioms with which the agent is instantiated and does not change. The EO contains acquired axioms and allows the agent to augment its knowledge base with concepts, without affecting its core expertise. The DO imports the EO, so that it uses the information contained in both the DO and EO to make decisions.

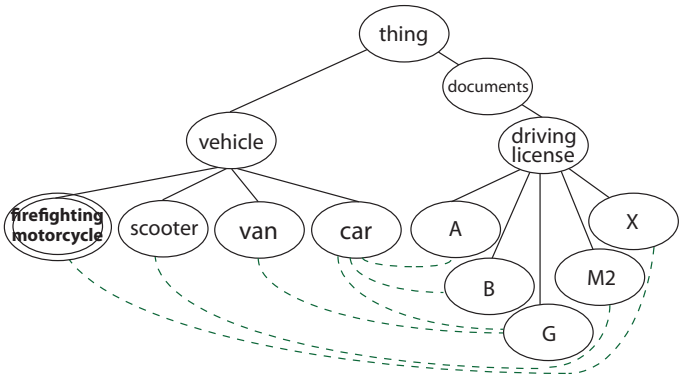
In order to explain our approach, we use the following example: a rescue agent a_t has a private ontology o_t that contains the concepts shown in Figure 4.1(a). It desires to learn the concept `firefighting_motorcycle` as used in Japan, because it currently does not use fire fighting motorcycles but would like to be able to rush to the scene of a fire disaster, without concern for such problems as traffic jams and narrow residential roads. Thus, it has received two fragments from the environment ontologies, which represent `firefighting_motorcycle`, as shown in Figures 4.1(b) and 4.1(c). We proceed to describe our merging process.



(a) An example rescue agent's ontology.



(b) First fragment, f_1, which represents the concept `firefighting motorcycle`.



(c) Second fragment, f_2, which represents the concept `firefighting motorcycle`.

FIGURE 4.1: The rescue agent's ontology, and three fragments, received from specialist agents in response to a query of `firefighting_motorcycle`. The dashed lines represent relationships between two concepts.

4.2.1 Merging Fragments

An agent a_t receives a set of fragments F from the agents that own the environment ontologies, which represent the queried concept c. The agent a_t merges the fragments contained in F to a new fragment f_m, where $concepts(f_m) = concepts(F)$. The fragments are merged into a new fragment by renaming the namespaces for each concept and relationship, so that they have the same namespace as a_t's ontology. The namespace specifies the source ontology of the concepts, however for now the new fragment's concepts are not augmented into the agent's ontology because they are not saved in the same file as the agent's ontology. In our fire fighting motorcycle example, the concept `firefighting_motorcycle` is contained in two ontologies, the Driver and Vehicle Licensing Agency (DVLA) ontology and the japaneseFireFighting ontology. These ontologies have the namespaces `http://www.dvla.co.uk/ont#` and `http://www.japaneserescue.org/japaneseFireFighting#` respectively, and the URIs for the concept `firefighting_motorcycle` are `http://www.dvla.co.uk/ont#firefighting_motorcycle` and `http://www.japaneserescue.org/japaneseFireFighting#firefighting_motorcycle`. We merge the fragments into a new fragment where all concepts are changed to have the same namespace as a_t's ontology. So that, in our example, `firefighting_motorcycle` from both fragments in now a single resource which shares all of the properties and attributes defined in both the DVLA and the japaneseFireFighting ontologies.

During this merging it is possible that a fragment might become inconsistent because the merged ontologies may define different aspects or attributes about a concept. For example, two ontologies that represent fire fighting motorcycles, from different counties, define that their motorcycles are always red, and the other blue; these attributes contradict each other. Therefore, when merging fragments it is advised by McGuinness et al. (2000) (see Section 2.2.1) to check whether they are consistent, which can be facilitated with a reasoner (see Section 2.1.3). When a fragment is inconsistent it is possible to select the largest consistent subset of axioms using an algorithm in the manner described by Haase and Stojanovic. Haase and Stojanovic do not recommend a specific algorithm for selecting this subset, nor is there a standard algorithm. Therefore, we present Algorithm 7 which can be used to select the largest subset of consistent axioms using a reasoner. The algorithm creates the powerset of all received fragments, and orders them by size. Beginning with the largest of the set it evaluates all of the fragments in the set for consistency. When a set's fragments are all consistent, it is selected. Thus, this will be the largest consistent set.

For our evaluation experiments we *a priori* verified that the union of the agents' and environment ontologies are consistent. Therefore, we negate the need to check whether the evolving ontology is consistent using Algorithm 7. In an open world system it is unlikely that the union of all the available ontologies contains no inconsistencies,

Algorithm 7 Algorithm which selects the largest consistent fragment.

Require: $P' \leftarrow$ powerset of received fragments
Require: o_t is the ontology of a_t which is the agent performing this algorithm

Function: *consistent*(): using a reasoner, this function returns true if the parameter's axioms are consistent and false if they are not.

```
1: /* order the powerset by the number of sets of fragments, largest
   first */
2: P'_o ← { order P' by |P'|}
3: /* order the subsets of fragments by the number of concepts they con-
   tain, largest first */
4: P'_os ← { suborder P'_o by |concepts(P'_o)|}
5: /* loop through each set of fragments (Set) in the powerset */
6: for Set ∈ P'_os do
7:   /* return the largest set of fragments that is consistent when
     merged with the agent's ontology */
8:   if consistent(Set ∪ o_t) then
9:     return Set
10:  end if
11: end for
```

however the use of the above algorithm after each augmentation of the agent's ontology will ensure consistency of the agent's ontology. Thus, our algorithm is suitable for use in environments where inconsistencies can occur. In the next section we describe our learning approach for selecting the concepts to augment into an agent's ontology.

4.2.2 Concept Selection

Our concept selection approach, selects concepts to augment into an agent's ontology. We adopt this approach because we want to balance the trade-off between learning everything, which can be costly in terms of accessing the knowledge from the ontology, against sending requests for fragments, which describe concepts required to complete a task which can also incur costs of time and resources. In order to balance this trade-off, we select concepts closely related to the domain of the agent's EO by first selecting a set of concepts with a similar depth, through **Hierarchical Selection**, followed by reducing the number of relations to the agent's ontology, through **Relational Selection**.

The following learning algorithm can be used with OWL Lite and OWL DL ontologies, and we have verified using a reasoner that the ontologies created in those OWL sublanguages were valid. However, it is not possible to verify that these created ontologies are valid OWL Full, because no reasoners support complete OWL Full reasoning. Reasoning over an ontology with the expressiveness of OWL Full is not possible because its lack of restrictions on the use of transitive properties mean that is not fully decidable (Horrocks et al., 2000). Thus, we believe that our algorithms are applicable to all expressivities of

OWL because our selection process uses structural components that are present in all OWL sublanguages, even though we have been unable to demonstrate that they apply to OWL Full. We describe the hierarchical and relational selection algorithms in the next two sections.

4.2.3 The Hierarchical Selection Algorithm

The hierarchical selection technique returns a list of concepts C^h, where h denotes that these concepts are selected in the hierarchical selection process. The hierarchical selection technique aims to reduce the number of superclasses that are used to represent the target concept c, when f_m has a larger hierarchical depth than o_t, to reduce the amount of non-domain-related information the agent learns.

The hierarchical selection algorithm aims to reduce the number of concepts by analysing the superclass hierarchy of the target concept, and prioritising the selection of concepts which are related to the learning agent's ontology. This method is used because the parent classes of a concept may not be relevant outside of the context of their original hierarchy, and this is particularly true if the received fragments contain a deep branch of parents which do not link to any concepts in the agent's ontology. Thus, the hierarchical selection algorithm rates concepts based on the number of axioms that refer to them, so that agents are less likely to learn about concepts that do not integrate with their ontology.

The hierarchical selection technique rates concepts in f_m with a *concept rating* using the following equation which rates concepts according to how they relate to the agent's domain of interest. The equation calculates the number of axioms that refer to the concepts in both the DO and EO, and we weight more highly those concepts which are referred to by more axioms. The concept rating provides an agent with the ability to rate how relevant a concept is to its ontology, thus increasing the likelihood that it will be augmented into its ontology.

$$concept\ rating_c = w * (nDO + nDOR) + nEO + nEOR \tag{4.1}$$

where nDO and nEO are the number of axioms which refer to the concept c in the DO and EO, respectively, and $nDOR$ and $nEOR$ are the number of axioms which contain a relationship between the concept c in f_m and a concept in the agent's DO and EO, respectively. The concepts in the DO are core concepts that are important to the basic functionality of the agent. Thus, we use the concepts in the DO and EO as a basis for learning new concepts, we weight the concepts from the DO more highly, to ensure that concepts related to the agent's domain are learned. A weighting w is used to increase the rating of concepts that refer to items in the DO, which are domain related. We aim

to ensure that an agent learns information about its domain of expertise so that it can support unforeseen tasks. It is possible that evolving an ontology can alter an ontology's domain, therefore we prioritise learning concepts related to the those that are contained in an agent's DO. The concepts in the DO are core concepts that are important to the basic functionality of the agent. We use both the concepts in the DO and EO to select which concepts to learn. In order to prioritise learning concepts related to the DO, we use a weighting for concepts in the DO. Specifically, we use the weighting w to increase the rating of concepts that refer to items in the DO. This weighting can be adjusted to specific requirements: the lower the weighting, the more likely an agent can evolve its ontology to represent a different domain to its intended domain; the higher the weighting is the less likely it is to deviate from its intended domain. The weighting also affects the number of concepts that the agent learns; with higher weightings it is more likely to incorporate axioms about concepts it already contains in its ontology. In contrast, if the weighing is lower, the agent will incorporate more axioms about new concepts because it prioritises learning concepts that are not necessarily directly connected to those in its DO. Currently, we use the weighting $w = 2$, which we chose because concepts in the DO will then have twice the influence on learning compared to concepts from the EO. We considered other values of w (see Figure 4.2) and selected $w = 2$ because it balanced the trade-off between learning the highest and lowest number of concepts because:

- We want an ontology to learn new concepts that will support future tasks, and therefore selecting a higher w will reduce the number of concepts that will support future tasks.
- We also want an ontology to contain a limited amount of concepts because the larger the ontology the greater number of resources it requires, and therefore a lower w will increase the memory resources required.

In order to show this trade-off Figure 4.2 shows the average number of concepts in an agent's ontology given a weighting. In summary, when w equals 1, 2 and 3, on average agents learn 2904, 2540 and 2385 concepts, respectively.

The hierarchical selection then performs one of two different actions, depending on the depth of the fragment, specifically, the following cases:

1) $depth(concepts(f_m)) > depth(concepts(o_t))$. In this case, the agent reduces the depth of the fragment. The agent calculates the average depth d of f_m and o_t, in our example 4, and selects d levels. It does this by selecting $d - 1$ levels with the highest mean average concept value shown in Figure 4.3. In our example, the levels selected are: level 3, because it contains c (the level containing c is always selected first); followed by level 2, level 5 and level 4, based on the order of the level's average concept rating.

2) $depth(concepts(f_m)) \leq depth(concepts(o_t))$. In this case, the agent selects all of the levels in f_m.

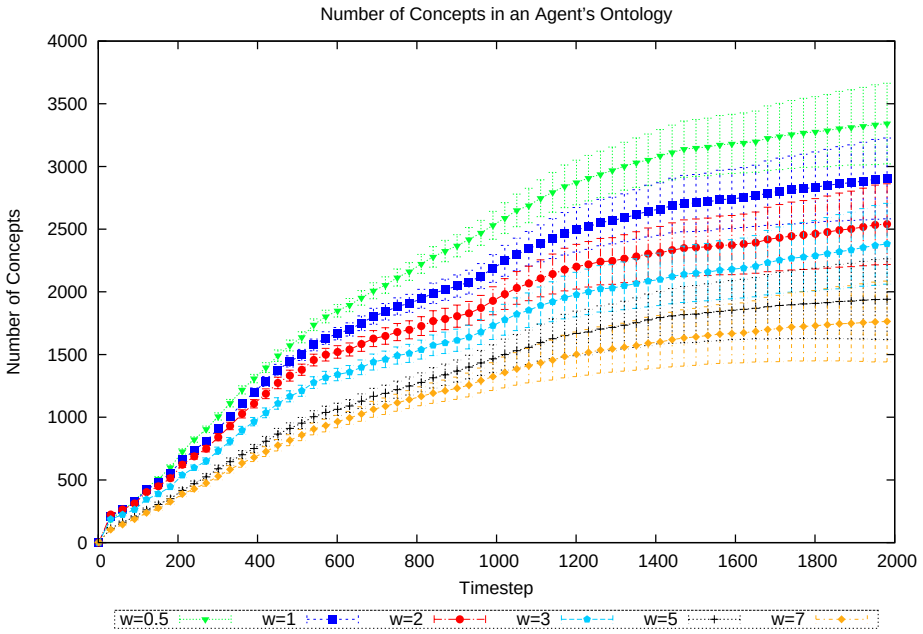


FIGURE 4.2: The number of concepts augmented into an agent's ontology with different values of w.

Both of these cases create a set of concepts, C^h, that represent the target concept, c. After the hierarchical selection process, the resulting set of concepts is then used by the relational selection technique to select the concepts to augment into the rescue agent's ontology. This selection process is described in Algorithm 8.

4.2.4 The Relational Selection Algorithm

The relational selection algorithm is designed to select concepts that are related through relationships to the target concept. The purpose of selecting concepts via semantic relationships is they are closely connected to the target concept, and we assume that if you require the target concept then it is likely that you will require closely connected concepts for future tasks. For example, a `vehicle` has the `engine_type` relationship to `petrol` and `diesel`, and by selecting these concepts at the point when `vehicle` is selected, then any query about the vehicles' engines will not require additional learning. Therefore, by selecting related concepts it is likely that future needs have already been met, thus removing the need to individually learn those concepts in future.

The relational selection technique returns a set of concepts C^r that will be incorporated into an agent's EO, where r denotes that these concepts were selected by the relational technique. The relational selection technique is used on the concepts selected in the

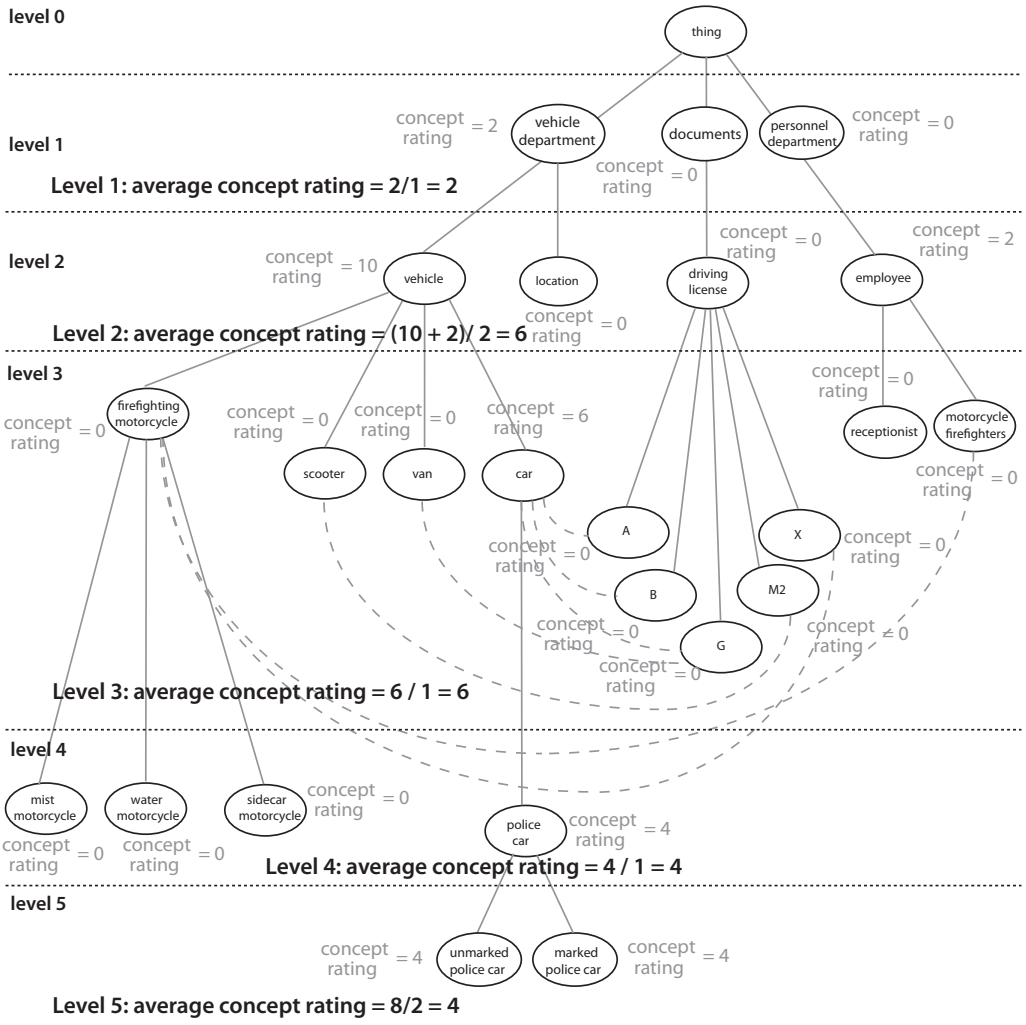


FIGURE 4.3: The merged fragment showing the average concept ratings per hierarchical level.

hierarchical selection process. The first stage of our relational selection technique selects concepts by traversing relationships, such as the subsumption relationship *subClassOf*, and object properties, for example *requiresLicense*, from our target concept. The distance of traversal is determined by a threshold t. To calculate t, we determine if an agent has an ontology that contains no relationships, in which case $t = 1$, otherwise $t = round(\frac{\sum r_r}{r_c})$, where r_r is the number of concepts that have more than one relationship, and r_c is the number of concepts that are linked together by relationships. The second stage of our relational selection technique recursively selects the parents of the concepts selected in the first stage, up to the root node. In our example, $t = 1$, and eight concepts are related at this distance from the target concept `firefighting motorcycle`. This process is described in Algorithm 9.

In our fire fighting motorcycle example, the agent first selects concepts that are directly related through relationships to the `fire_fighting_motorcycle` concept. Then, the agent selects concepts that are directly connected by relationships to the concepts selected in the previous stage. This is shown in Figure 4.4 which depicts the concepts

Algorithm 8 Algorithm showing the Hierarchical Selection technique.

Function: $depth(x)$ returns the depth of an object x that contains classes
Function: $getHighestLevels(n)$ returns the concepts from the highest n number of levels in the merged fragments F_d
Function: $getLevelContaining(c)$ returns the concepts in the level containing class c

Input: $F_d \leftarrow$ fragment received from merging process
Input: $o_t \leftarrow$ agent's ontology
Input: $n \leftarrow$ number of depths of classes to incorporate
Input: $c_t \leftarrow$ concept to incorporate

```
/* if the fragment has a greater depth than the agent's ontology */
if depth(F_d) > depth(o_t) then
    C^h ← getHighestLevels(n)
    /* check that the selected concepts contain the target concept */
    if C^h ∉ c_t then
        C^h ← getLevelContaining(c_t)
        C^h ← C^h ∪ getHighestLevels(n − 1)
    end if
else
    /* return the whole fragments if the depth is lower or equal to
    that of the agent's ontology */
    C^h ← concepts(F_d)
end if
return C^h
```

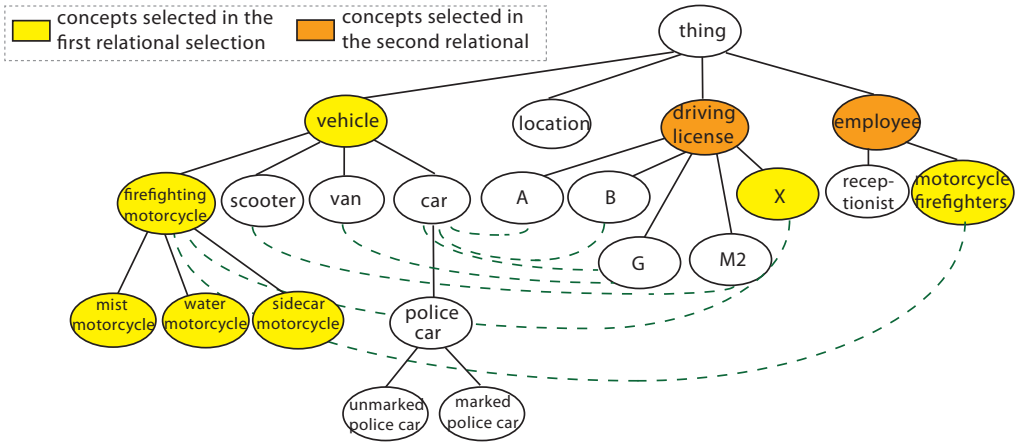


FIGURE 4.4: A selection of concepts which are related to c by subsumption, where $c =$`firefighting_motorcycle`.

selected in the first stage in yellow, and the concepts selected in the second stage in orange. This selection results in a set of concepts to be augmented into the agent's ontology. In our example, the selected concepts (shown in Figure 4.5) are augmented into the agent's ontology which result in the ontology represented in Figure 4.6.

Algorithm 9 Algorithm showing the Relational Selection technique.

Require: t is the average number of relationships that connect classes in an agent's ontology

Function: $path(c)$ returns a set of concepts from the root node to the parameter concept c to the leaf nodes, from C^h.

Function: $relations(c)$ returns a set of concepts that are related by relationships to c_t where the relationships are defined in the merged fragment and the concepts are from C^h.

Input: C^h is the concepts from the hierarchical selection technique

Input: c_t is the target concept

Require: $related \leftarrow \emptyset$

1: $C^r \leftarrow path(c_t)$
2: `/* loop` t `number of times */`
3: **for** $i \in \{1 \ldots t\}$ **do**
4: $ToVisit \leftarrow \{c_t\}$
5: `/* loop through the concepts in ToVisit */`
6: **for** $class \in ToVisit$ **do**
7: $Related \leftarrow Related \cup relations(class)$
8: `/* select relationships of the concept */`
9: **for** $c \in Related$ **do**
10: $C^r \leftarrow C^r \cup path(c)$
11: `/* traverse related classes */`
12: $ToVisit \leftarrow ToVisit \cup c$
13: **end for**
14: **end for**
15: **end for**
16: **return** C^r

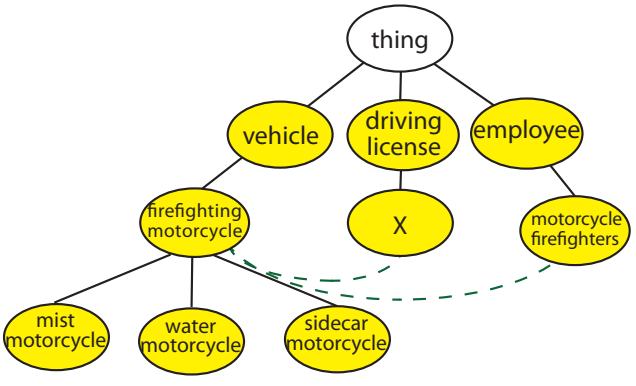


FIGURE 4.5: The chosen set of concepts which are augmented into a_t's EO.

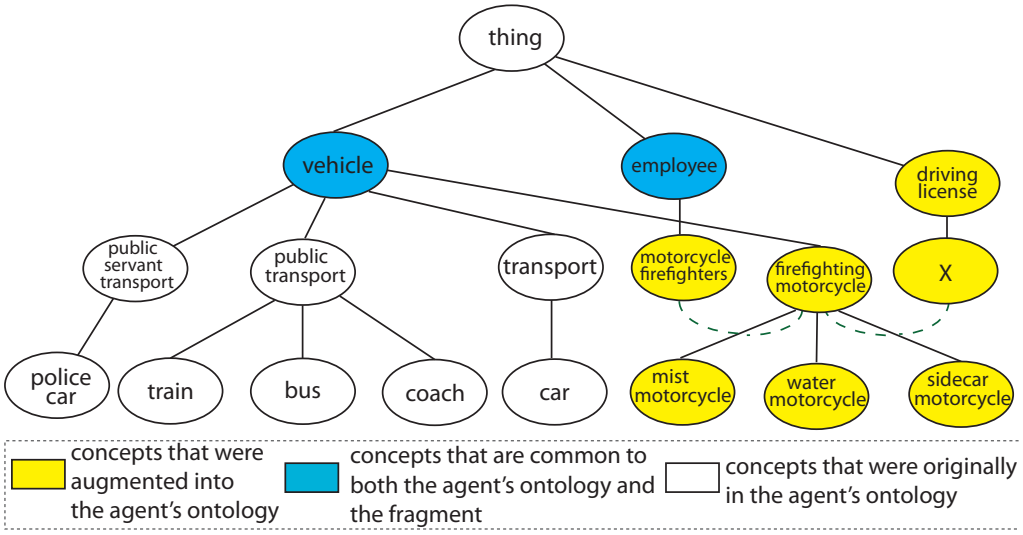


FIGURE 4.6: Our example ontology augmented with the selected concepts.

This approach can be compared to the state-of-the-art ontology modularisation techniques, which have been categorised by Palmisano et al. (see Section 2.2.3) by the way they extract modules. These categorise include: "traversal-based extraction", where modules are partitioned based on their class structure; or "logical-based extraction" where semantics of the ontology are considered. Our approach uses traversal and therefore can be compared to Noy and Musen (2003), Doran et al. (2007) and Seidenberg and Rector (2006) (see Section 2.2.3). Our approach, however, is the most similar to Seidenberg and Rector's, because unlike the other approaches it does generate huge fragments or require the translation of an ontology to a graph model. Therefore, we contrast our approach to Seidenberg and Rector (2006) so that we can highlight the difference between the concepts selected by: our learning algorithm which selects based on how relevant they are to an agent's ontology; and Seidenberg and Rector's approach which selects concepts relevant to the target concept:

1. Generates a different fragment representing the same concept for different ontologies. Seidenberg and Rector's approach would generate the same fragment for the same concepts every time. This is because our approach selectively chooses concepts based on a rating generated from another ontology. In contrast, the segmentation technique of Seidenberg and Rector produces the fragment illustrated in Figure 4.7[1]. We note that Seidenberg and Rector's approach does not contain all of the concepts that directly relate to `firefighting_motorcycle`, whereas our approach produces a fragment with a more detailed representation of `firefighting_motorcycle`. Both Seidenberg and Rector's approach and our approach

[1]This fragment was generated using the `Segmentation Application` available from: `http://www.co-ode.org/galen/`, which was developed by Seidenberg and Rector's work (Seidenberg and Rector, 2006).

produce fragments that remove concepts that do not directly relate to `firefighting_motorcycle`, while our approach's fragment automatically contains concepts that are already contained in our agent's ontology because of the use of concept ratings, whereas Seidenberg and Rector's approach does not contain concepts that are in our agent's domain;

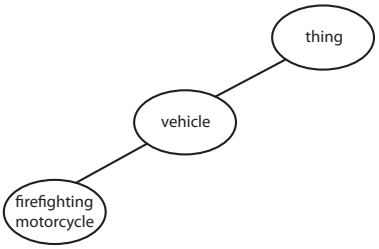


FIGURE 4.7: A fragment generated from the merged fragment illustrated in Figure 4.5 representing the concept `firefighting_motorcycle`, using Seidenberg and Rector's segmentation approach.

2. Constructs a fragment that mimics the same level of detail obtained in another ontology by: generating a fragment with a small or the same hierarchical depth as the ontology; and generating a fragment representing the concepts with the same average number of relationships as the ontology. The depth of the fragment generated by Seidenberg and Rector's approach produces a fragment with the same depth as the concept which the fragment represents from the original ontology, whereas our approach limits which classes it selects for the hierarchy of the fragment, so that the agent's ontology remains focused on its domain. In our example, our selected fragment has a depth of three, whereas Seidenberg and Rector's approach produced a fragment with a depth of two, this is because our agent's ontology has a depth of three and our approach aims to match the same level of detail. As before, Seidenberg and Rector's approach will generate the same fragment given the concept, whereas our approach will differ for agents with different ontologies;

3. Focuses on generating a fragment for the purpose of augmenting it into an agent's ontology, and thus it aims to contain concepts that relate to this ontology so that when the agent augments its ontology it maintains its intended domain. Seidenberg and Rector's approach does not address this scenario. In our example, our approach effectively provides an agent with information about `firefighting_motorcycle` and enables it to augment the fragment into its ontology by situating `firefighting_motorcycle` within the ontology by using concepts which are contained in both the merged fragment and the agent's ontology. This reduces the chances of an agent's ontology containing many branches from the concept `thing` which do not contain relationships linking these branches together, a reduction of the number of branches can reduce the complexity of the ontology (as discussed in Section 2.2.2). Our approach prioritises the selection of concepts that are already

contained in the agent's ontology. Therefore, compared with Seidenberg and Rector's approach, the augmented ontology has improved cohesion, caused by learning concepts that relate to the augmented ontology;

4. Generates a fragment from a comparatively smaller ontology (a merged set of fragments) than Seidenberg and Rector's approach, which is designed to enable a user to load a fragment from a large ontology, so that not all concepts are loaded at one time, thus reducing the loading and reasoning complexity. Both our and Seidenberg and Rector's approach are designed to fulfil different objectives.

Given these differences, we have addressed part *a)* of our first objective (see Section 1.3) and have provided a novel approach that enables an agent to select a set of concepts from a set of fragments to augment into its ontology. This approach is effective because it enables a higher cohesion of concepts than approaches described in Section 2.3.1. Our approach achieves a higher cohesion through the use of our hierarchical selection algorithm that selects concepts using a depth similar to that of the target ontology. Thus, long disconnected branches are avoided, and the cohesion of the ontology is greater. In contrast to the comparative approaches presented in Section 2.3.1 where their approaches require that all agents have the same experiences or all agents have the same domain, our approach compares the domain of the received information which enables a diverse set of agents to learn from, and will only augment its ontology with domain related concepts. Also in contrast to the comparative approaches, our approach augments its ontology with more than one concept at a time, thus improving the cost of acquisition for sets of queries which require information about one section of a domain.

4.3 Empirical Evaluation

This section discusses our empirical evaluation which evaluates the performance of the reactive learning algorithm, against benchmark learning approaches (detailed in Section 3.3.1). To this end, we present our hypotheses and how we intend to show them in Section 4.3.1. The next section provides the specific parameters of our experiments. In the following section we present and discuss the results from the experiments.

4.3.1 Hypotheses

This section presents our hypotheses which show how our reactive approach addresses our contributions detailed in Section 1.4. In more detail, our learning algorithm is required to:

1. Demonstrate a fully automated technique that enables an agent to learn;

2. Augment an agent's ontology, and demonstrate that this is an effective way for an agent to complete continual tasks in a specific domain.

In order to demonstrate that our approach addresses our contributions and meets our first research objective (see Section 1.3), we aim to evaluate the following hypotheses:

Hypothesis 1: Augmenting an agent's ontology will reduce the number of messages required to complete tasks, compared to agents that learn nothing. Specifically, our approach will require the least number of messages compared to all other agents that collaborate. This hypothesis considers the effectiveness of our reactive learning algorithm of reducing the messages required to complete tasks. It addresses Objective *2a* which aims to reduce the costs of re-acquiring regularly required knowledge. We hypothesise that our algorithm will perform better because it benefits from the advantages of augmenting an agent's ontology (see Section 4.1) and selectively augments more than one concept at a time;

Hypothesis 2: Our approach will enable an agent to augment its ontology with relevant concepts, based on its domain, and based on the structure of its ontology. This is because our reactive learning algorithm utilises a hierarchical and relational selection approach that limits the depth of classes augmented by how they relate to each other (see Section 4.2.2). Specifically, our approach will perform better in terms of the number of concepts in the ontology than the *learn-everything* approach, because we reduce the number of concepts that are learned through selection. This entails that our approach will evolve the ontology with the lowest estimated complexity (see Section 3.4). We measure the success of our hypothesis by comparing the number of tasks that are attempted and completed, and the size of an agent's ontology. The greater the number of task attempted and the smaller the ontology is the more domain focused the ontology is;

Hypothesis 3: Our approach will spend the least total time acquiring and learning concepts compared with the other approaches and thus outperforming them. This hypothesis considers the effectiveness of our approach against the benchmark approaches, by comparing the total time spend acquiring concepts. It addresses Objective *2a*, which aims to reduce costs of re-acquiring regularly required knowledge;

Hypothesis 4: Agents using our proactive learning algorithm will perform better than using other approaches in terms of their RoboCup Rescue score. We hypothesise that because of the above Hypotheses 1, 2 and 3 that agents using our approach will be able to save more of the city from fire and rescue more civilians. We will measure this using the RoboCup Rescue's score vector (see Section 3.5.1).

In the next section, we present the experiments used to show these hypotheses and detail their parameters.

4.3.2 Experimental Setup

In order to evaluate the effectiveness of our reactive learning algorithm, we perform the following experiments:

- **Experiment 1:** Evaluates the performance of the fire brigade agents;
- **Experiment 2:** Evaluates the performance of the ambulance agents;
- **Experiment 3:** Evaluates the performance of the fire brigade, ambulance, and police agents.

We isolate the fire brigade and the ambulance agents in Experiments 1 and 2 respectively, because their performance is interdependent. In theory, without the fire brigade, the ambulance agents should save fewer civilians because the fire brigade will not control fires; and without the ambulance agents, the fire brigade would be able save fewer civilians because they cannot save them from fires. Thus, in order to evaluate the effect of our algorithm we use all three types of agent in our third experiment. We omit running experiments with just the police agents because RoboCup Rescue's score vector is biased towards the percentage of rescued civilians and of the city affected by fire, while the role of the police agents affects that of the fire brigade and ambulance agents; they do not save civilians or extinguish buildings. In more detail, Table 4.1 contains the parameters used in our three experiments.

	Experiment 1	**Experiment 2**	**Experiment 3**
Environment ontologies	10	10	10
Number of agents	20	20	30
Number of fire brigades	10	0	10
Number of police	10	10	10
Number of ambulance	0	10	10
Number of time steps	2000	2000	2000
Map	Kobe	Kobe	Kobe
Benchmark approaches	Learn-repeated, Learn-everything, Learn-concept, No-collaboration	Learn-repeated, Learn-everything, Learn-concept, No-collaboration	Learn-repeated, Learn-everything, Learn-concept, No-collaboration
Number of civilians	115	115	115
Iterations	250	250	250

TABLE 4.1: The parameters for Experiments 1, 2, and 3.

4.4 Results

In this section, we focus on discussing the results from Experiment 3. The results from Experiment 3 provide an insight into how our reactive learning and comparison algorithms affect all three types of agent in the RoboCup OWLRescue environment, while the results from Experiments 1 and 2 show how the learning algorithms respectively affect only the fire brigade and only the ambulance agents in the RoboCup OWLRescue environment. The performances of these two types of agents depend on each other; these experiments do not show the full extent of the benefit of using our reactive learning algorithm (*learn-fragment*). In particular, Experiment 2's ambulance agents save more of their targets than other approaches that don't have perfect foresight, however their performance is hindered by how fast the fire spreads without the fire brigade agents putting out fires. Therefore, in the following four sections, we analyse the results that show our four hypotheses from Experiment 3 and present the results from Experiment 1 and 2 in Appendix A.

4.4.1 Hypothesis 1 — Messages

In order to evaluate the number of messages that are sent by each approach, we first consider the number of tasks an agent attempts to complete. This is because the number of tasks an agent attempts affects the number of messages sent during a simulation: the greater the number of tasks, the greater the need for an agent to acquire new information, and therefore the greater the number of messages are needed compared to an agent that attempts fewer tasks.

In particular, Figure 4.8 shows the average number of tasks that an agent attempts to complete. This figure indicates that in the region of the 1,000th timestep, all agents attempt approximately the same number of tasks, and after this timestep the approaches ordered from highest to lowest number of attempted tasks are *learn-repeated*, *learn-concept*, *learn-fragment*, *learn-everything* and *no-collaboration*. This graph does not clearly indicate which agent is the most successful because an agent's actions may reduce the number of future tasks. For example, a fire brigade may extinguish a fire, preventing the fire from spreading, while another agent was unable to extinguish the fire and thus the fire spreads creating more tasks.

Although all agents approximately attempt the same number of tasks at the 1,000th timestep, agents using our approach have sent fewer messages than those using other approaches (with the exception of the *no-collaboration* approach). At the end of the simulation, our *learn-fragment* approach used the fewest number of messages out of the agents that augmented their ontologies. The *learn-fragment* approach sends 35.0% fewer messages than the next best approach, *learn-repeated*. Despite the fact that the *learn-concept* and *learn-everything* approaches attempt fewer tasks than the *learn-fragment*

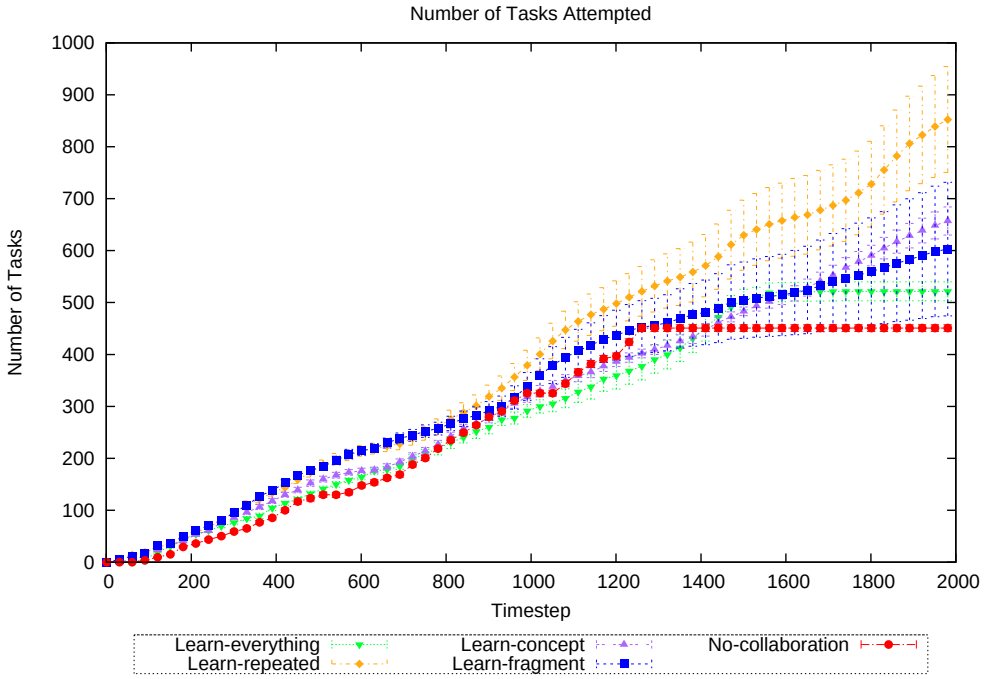


FIGURE 4.8: A graph showing the average number of tasks attempted per agent.

and *learn-repeated*, they send more messages. The *learn-everything* approach sends fewer messages than the *learn-concept* approach, because it acquires significantly more concepts in one timestep and thus has to send fewer messages than the *learn-concept* approach. Even though the *learn-everything* approach augments its ontology with more concepts than the *learn-fragment* approach, it still acquires the definition of more concepts. This suggests that the agents using the *learn-everything* approach augment many concepts which are not used (because it took too long to reach their target, which either died or burnt out) because it sends more messages, and the *learn-fragment* approach benefits from learning about concepts that relate to its domain because it sends fewer messages.

The *learn-concept* approach sends approximately 200% more messages than our approach because it does not augment its ontology with more than one concept at a time. This approach represents the state-of-the-art approaches (discussed in Section 3.3.1). From the graph in Figure 4.9, it can be seen that the different approaches send a similar amount of messages up until timestep 250 because their ontologies require the same information in order to complete the simulation. However, after the 250th timestep the more intelligent approaches (*learn-fragment* and *learn repeated*) start to separate from the *learn-everything* and *learn-concept* approaches because they are benefiting from augmenting only selected concepts into their ontologies.

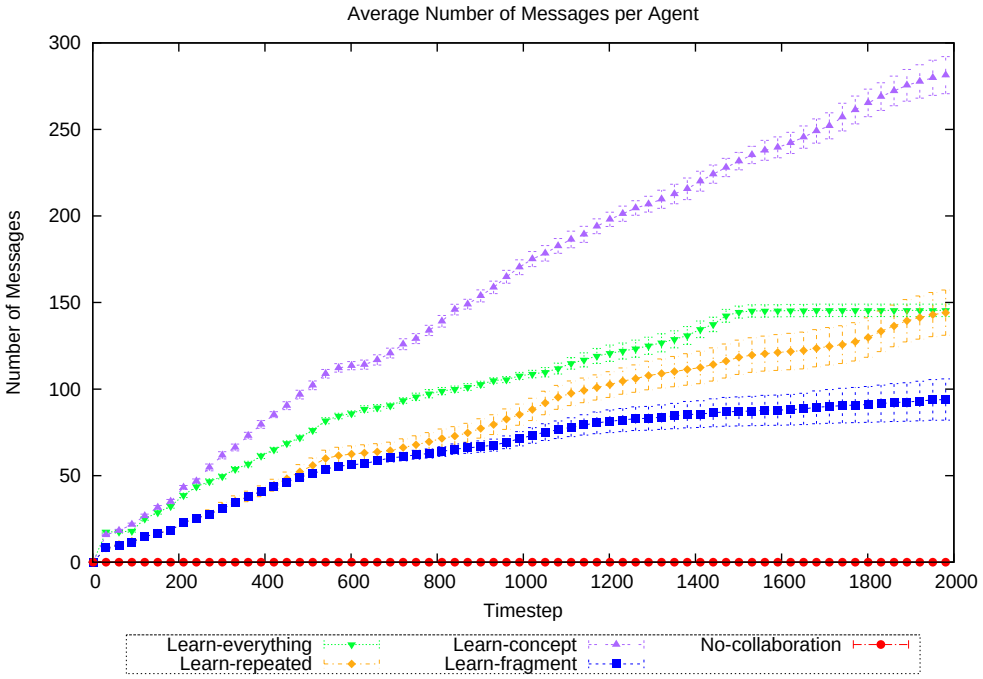


FIGURE 4.9: A graph showing the average number of messages sent per agent.

Table 4.2 shows that the agents require information uniformly throughout the simulation. We note that the agents do not send messages on the first two to three timesteps because the simulation doesn't fully initiate until after the first two timesteps. The average of the maximum number of messages per timestep is shown in the third column of Table 4.2 and shows that our approach has a lower maximum number of messages sent than the other approaches that augment their ontologies. The *learn-everything* and *learn-repeated* approaches have the next lowest maximum number of messages per timestep with a difference of 0.85 messages, and the *learn-concept* approach has the highest with 281.91 messages. This pattern can also be seen in the average number of messages sent per timestep, where our *learn-fragment* approach is the lowest with an average of 65.18, followed by the *learn-repeated* approach with 82.65 messages. This table also shows the average of the total number of messages sent (also shown in Figure 4.9) which shows that our *learn-fragment* approach sends 50.7 fewer messages than the next best approach (*learn-repeated*).

After approximately 1600 timesteps, the *learn-everything* approach plateaus in the number of attempted tasks (see Figure 4.8), the number of messages (see Figure 4.9) and the number of concepts learned (see Figure 4.10). After the 1600th timestep there are no more tasks because the fire burned out, thus there are no buildings to extinguish (see Figure 4.16).

Technique	Average number of timesteps with no messages	Average max per timestep	Average value per timestep	Average final value
Learn-everything	2.18	145.52	101.58	145.52
Learn-repeated	2.15	144.70	82.65	144.70
Learn-concept	2.12	281.91	161.38	281.91
Learn-fragment	2.12	94	65.18	94
No-collaboration	2000			

TABLE 4.2: Statistics for the number of messages sent per agent.

In summary, our hypothesis is confirmed because our approach sends the fewest number of messages (with the exception of the *no-collaboration* approach), outperforming the approach that has perfect foresight (the *learn-repeated* approach) in terms of communication costs. Our approach is able to send the fewest number of messages because our selection technique selects concepts that the agents might need in future. By selecting concepts that are required in future, the agents reduce the overhead of having to learn multiple times, and therefore sends fewer messages requesting concepts.

4.4.2 Hypothesis 2 — Ontology Size

In order to demonstrate that our learning algorithm is the most effective learning approach by evolving a small ontology with the lowest complexity, we must first identify the factors that affect the number of concepts that an agent learns. Specifically we note, that the performance of an agent affects the number of tasks it performs, which in turn affects the number of concepts an agent augments into its ontology. If an agent is unable to extinguish a fire then it is likely that the fire will spread causing more extinguishing tasks. Thus, agents that complete their tasks quickly and successfully can proceed to more tasks earlier, and learn more concepts in the process.

In order to show that our learning algorithm is able to augment its concepts with relevant concepts for its domain tasks, we compare the number of concepts contained within an ontology (shown in Figure 4.10) against the number of tasks it successfully completed (shown in Figure 4.11).

Our approach, *learn-fragment*, has the lowest number of concepts in its ontology, (apart from *no-collaboration* which learns no additional concepts). This means that our approach has the lowest ranked estimated complexity. Specifically, agents that use our approach have 36.6% fewer concepts than the next best approach (*learn-everything*). It was unexpected that the *learn-everything* approach would have the second lowest number of concepts because it learns everything. However, it requires fewer concepts than the *learn-concept* and *learn-repeated* approaches because it attempts fewer tasks (see Figure 4.8). Our *learn-fragment* approach also has the highest average number of

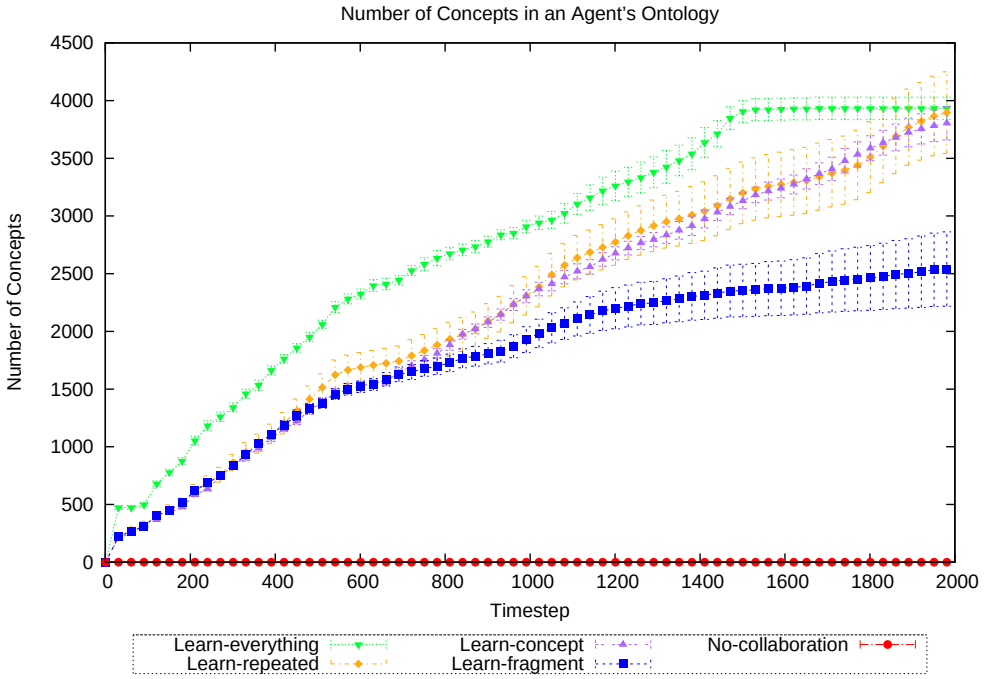


FIGURE 4.10: A graph showing the average number of concepts in each agent's ontology.

successful tasks completed, at 89.47, with the next highest being the *learn-repeated* approach which completed an average of 84.53 tasks, as shown in Table 4.4. The variance of the total number of completed tasks for the *learn-repeated* approach overlaps with the variance and average of the *learn-concept* approach, and our approach learns 65.5% fewer concepts than the *learn-repeated* approach, as shown in Table 4.3. Thus, our technique has a smaller number of concepts and estimated complexity, and is still able to complete approximately the same total number of tasks. While the *learn-everything* and *learn-concept* approaches augment their ontology with approximately the same number of concepts as the *learn-repeated* approach, they complete fewer tasks. The *no-collaboration* approach completes on average fewer than 5 tasks because it does not have the knowledge to select which equipment is necessary to rescue their targets.

Technique	**Average final value**	**Average minimum value**	**Average maximum value**
Learn-everything	2588.03	2237	3640
Learn-repeated	3135.86	545	4106
Learn-concept	3150.6	2297	3460
Learn-fragment	1894.83	1342	2600
No-collaboration	893.07	836	930

TABLE 4.3: Statistics for the number of concepts in each agent's ontology.

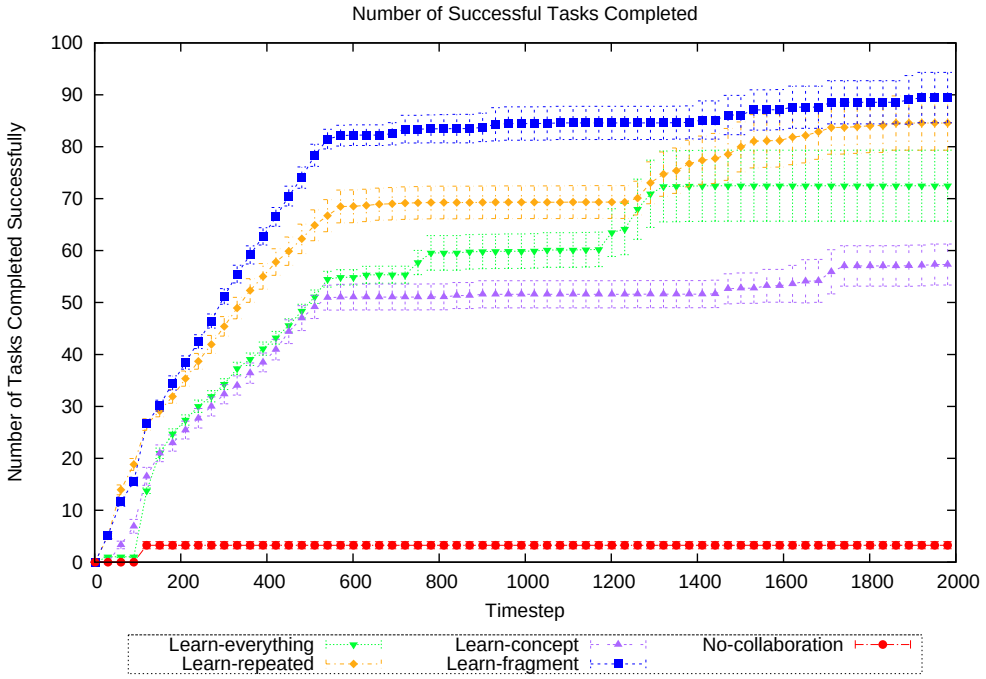


FIGURE 4.11: A graph showing the average number of successful tasks completed by each agent.

Technique	**Average number of successful tasks completed per timestep**	**Average final number of successful tasks completed**
Learn-everything	0.036	72.5
Learn-repeated	0.042	84.53
Learn-concept	0.028	57.32
Learn-fragment	0.045	89.47
No-collaboration	0.0016	3.26

TABLE 4.4: Statistics for the average number of successful tasks an agent performs.

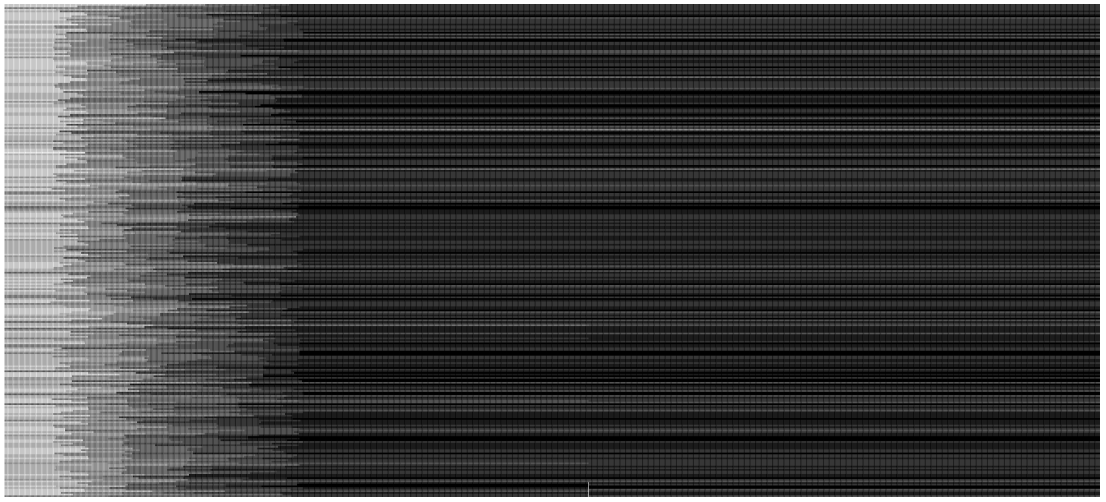
In summary, agents that use our approach augment the fewest concepts of all of the compared approaches (with the exception of the *no-collaboration* approach) and thus have the lowest estimated ontology complexity. Our approach even outperforms the approach that has perfect foresight (the *learn-repeated* approach). This is because it enables agents to select fewer concepts which are related to its domain, to augment into their ontology. Additionally, agents using our approach complete the highest number of tasks successfully, and more of the learning operations lead to successful tasks.

4.4.3 Hypothesis 3 — Learning, Deliberating and Acting

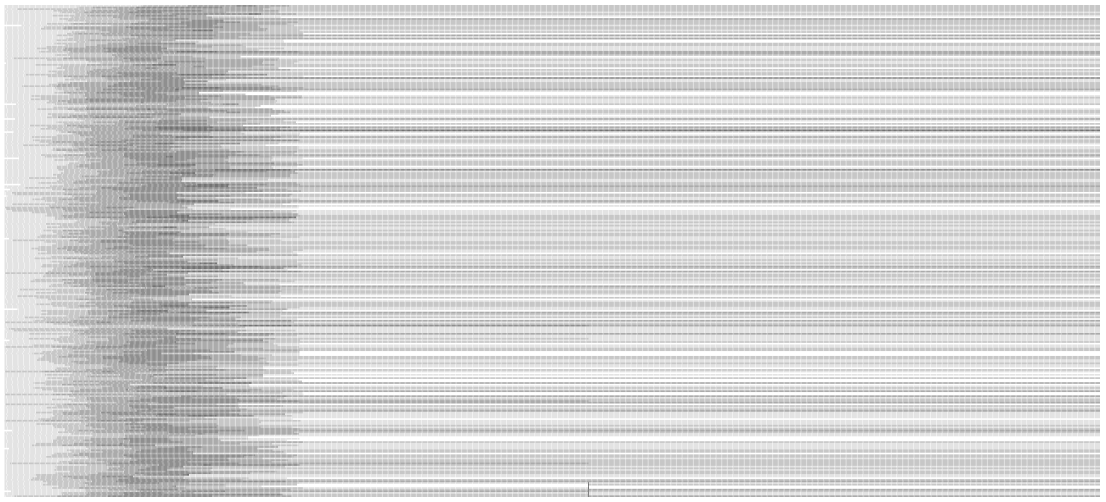
For each approach analysed in our evaluation, we use three types of pixel plots that consider three types of activities an agent undertakes: learning, which includes acquiring, selecting, and augmenting selected new concepts into the ontology; deliberating which actions to take; and acting. In more detail, deliberating is different for each type of agent, each agent performs observations, decision-making and planning during deliberation, and these processes are shown in sequence diagrams in Figures 3.7, 3.8 and 3.9. We show these three types of activities because they categorise all of the tasks that agents perform, while also differentiating between the core type of tasks where the approaches differ in the amount of time spent. Each of the three pixel plots use black to indicate either the time spent learning, deliberating, or acting, and white indicates the time spent on the other types of activities. Each pixel plot is composed with: columns which represent a timestep with 10 pixels, and are 2000 timesteps wide; and rows where each row represents individual agents from 250 RoboCup Rescue simulations from Experiment 3. These pixel plots are designed to give an overview visually of how the agents spend their time. In general there is banding between the rows, this is because these represent agents from different simulations and each agent encounters different tasks. We exclude the pixel plots from the *no-collaboration* approach because the agents do not augment their ontologies. We now proceed to analyse the different agent learning approaches, with regards to the amount of time spent learning, deliberating and acting.

The *learn-everything* approach spends the majority of its time learning (see Figure 4.12(a)) compared to deliberating (see Figure 4.12(b)), and time spent acting. The amount of time the agent spends learning (which includes acquiring and augmenting concepts and not selecting concepts to learn because it is not selective and learns everything) hinders the amount of time that an agent deliberates which actions to take, because after approximately 600 timesteps the agents spend little or comparatively no time acting. The agent using this approach peaks in deliberating during the 200 to 400 timesteps, this is because the agent's ontology has grown large (see Figure 4.10) and therefore it takes longer to deliberate than before 200 timesteps and is not able to spend as much time deliberating after 400 timesteps, because the agent spends more time learning. Despite the fact that the *learn-everything* approach does not analyse which concepts to learn, it still spends more time learning than any other approach (see Figures 4.13(a), 4.14(a), and 4.15(a)), this is because the agent's ontology has grown large, and thus augmenting concepts into it takes longer. Augmenting a large ontology takes more time than augmenting a smaller ontology because the merging algorithm has more axioms to process.

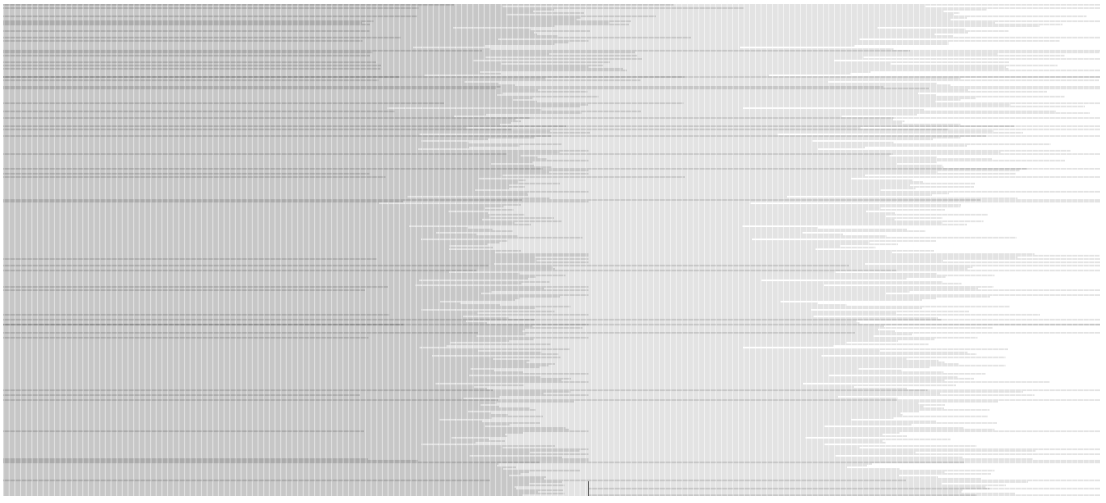
The *learn-concept* approach spends more of its time learning (see Figure 4.13(a)), than deliberating (see Figure 4.13(b)), and acting (see Figure 4.13(c)). The amount of time the agent spends learning increases as the number of timesteps increases, and there-



(a) Pixel-plot showing the time spent learning for the *learn-everything* approach.



(b) Pixel-plot showing the time spent deliberating for the *learn-everything* approach.



(c) Pixel-plot showing the time spent acting for the *learn-everything* approach.

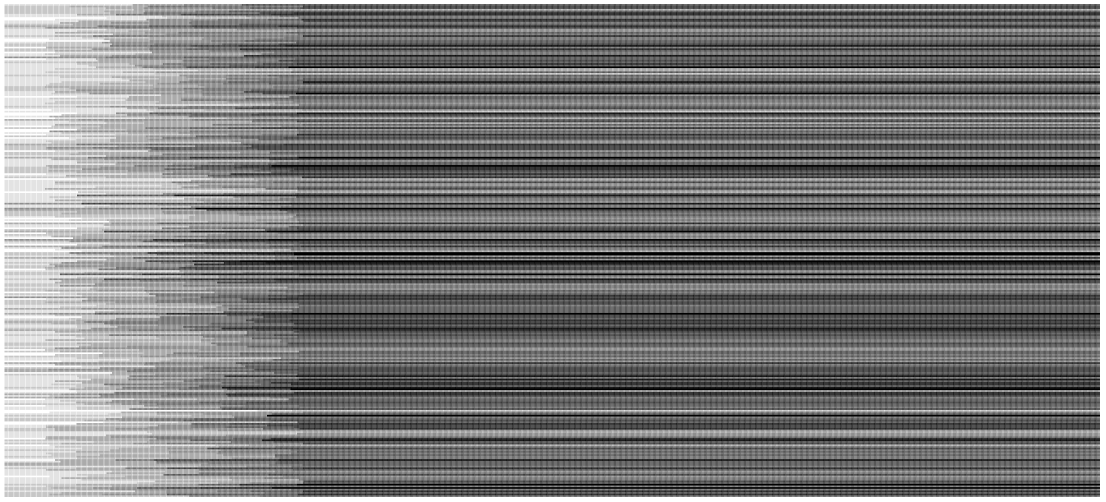
FIGURE 4.12: Pixel plots for the *learn-everything* approach.

fore so does the time spent deliberating. This is because the agent's ontology grows monotonically and thus more time is required to deliberate. This means that the agent can't spend as much time acting as it does learning and deliberating. The *learn-concept* approach spends more time learning because it can only learn one concept at a time. Thus, when multiple concepts are required, agents that learn a subset of the concepts from the received fragments (the *learn-repeated*, and *learn-fragment* approaches) can learn multiple concepts in a single operation, reducing the learning overhead, compared to learning one concept at a time.

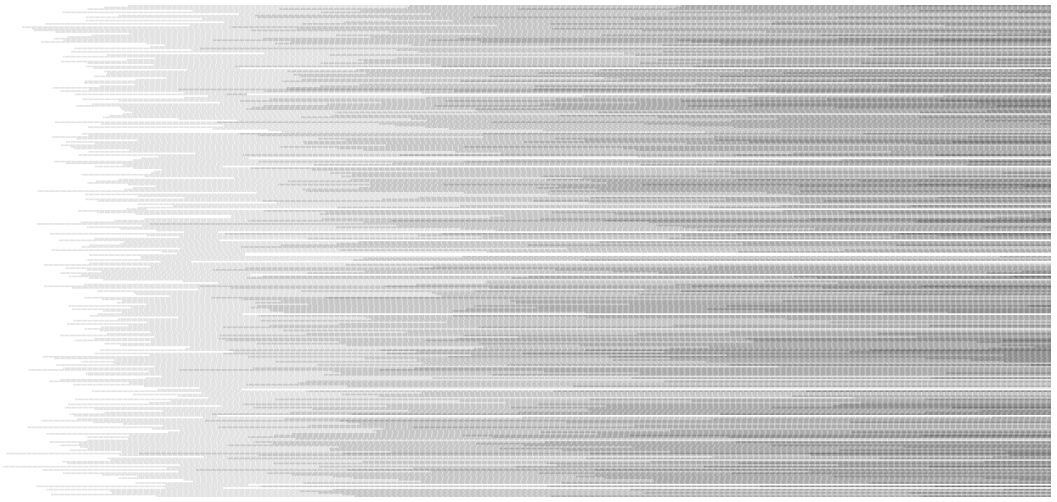
The *learn-repeated* approach spends slightly more of its time learning (see Figure 4.13(a)) than deliberating (see Figure 4.13(b)), and than acting (see Figure 4.13(c)). This approach is designed to optimise the number of concepts in the agent's ontology so that it benefits from learning concepts that will be used more than once and therefore does not retain concepts that are used only once. This means that the agent analyses whether each concept that is augmented will be used again, therefore this increases the time spent learning. However, it spends less time learning (see Figure 4.14(a)) compared to agents using the *learn-concept* approach (which learns one concept at a time) and the *learn-everything* approach (see Figures 4.13(a) and 4.12(a)).

Our approach, *learn-fragment*, spends most of its time learning (see Figure 4.15(a)) and spends less time deliberating (see Figure 4.15(b)) and acting (see Figure 4.15(c)). Compared with the other approaches, it spends the least amount of time learning and deliberating, and therefore can spend the most amount of time acting. In particular, there are a number of agents which spend noticeably less time learning indicated with a lighter contrast of grey than the other agents (see Figure 4.15(a)). These specific agents spent more time acting which is indicated by the darker rows (see Figure 4.15(c)). This shows that there is a relationship between the time an agent spends learning and acting, where an agent can spend more time acting if it spends less time learning. On average our approach spends the most time learning during the 200 to 1000 timesteps. This is because the agent's ontology increases in size faster from timesteps 1 to 900, and the rate at which the ontology grows decreases after 900 timesteps (see Figure 4.10). The amount of time the agent spends deliberating increases over the simulation because the size of the agents' ontologies increases monotonically over time.

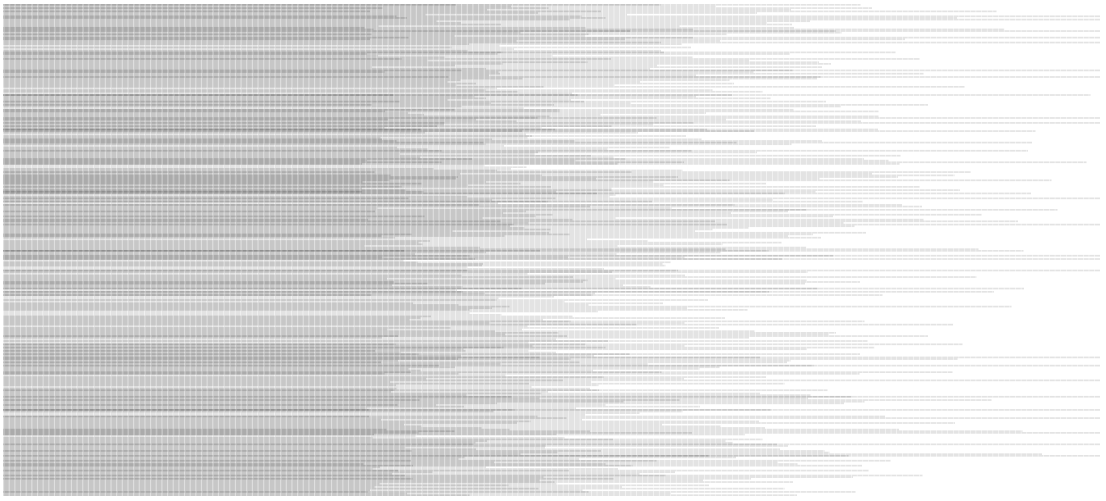
In summary, the agents using our approach spend less time learning because they learn more than one concept at a time but fewer concepts than they receive, and therefore spend less time deliberating because their ontologies are smaller compared to the other approaches that learn. These two factors enable the agents to spend the most time acting, compared to the other approaches (with the exception of the *no-collaboration* approach), so they can rescue their targets faster.



(a) Pixel-plot showing the time spent learning for the *learn-concept* approach.

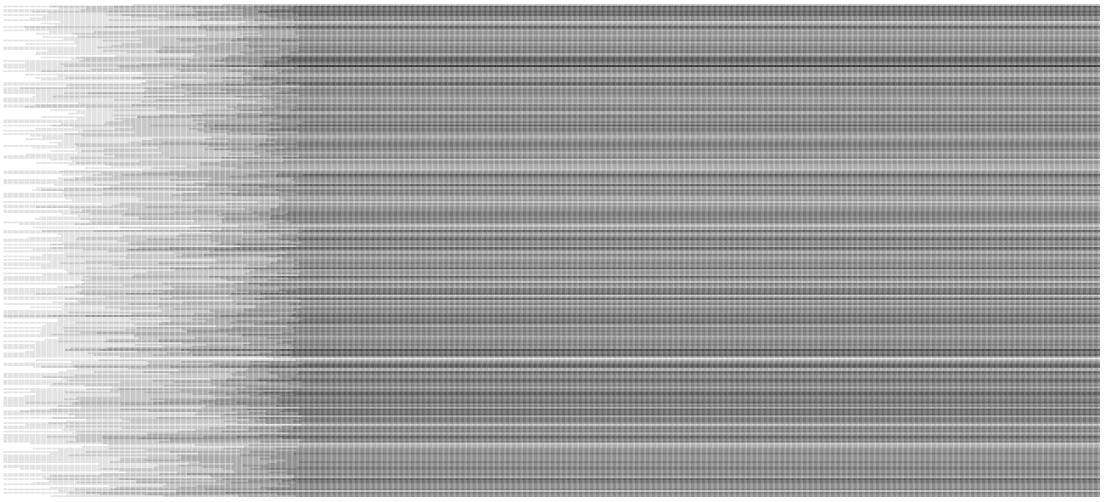


(b) Pixel-plot showing the time spent deliberating for the *learn-concept* approach.

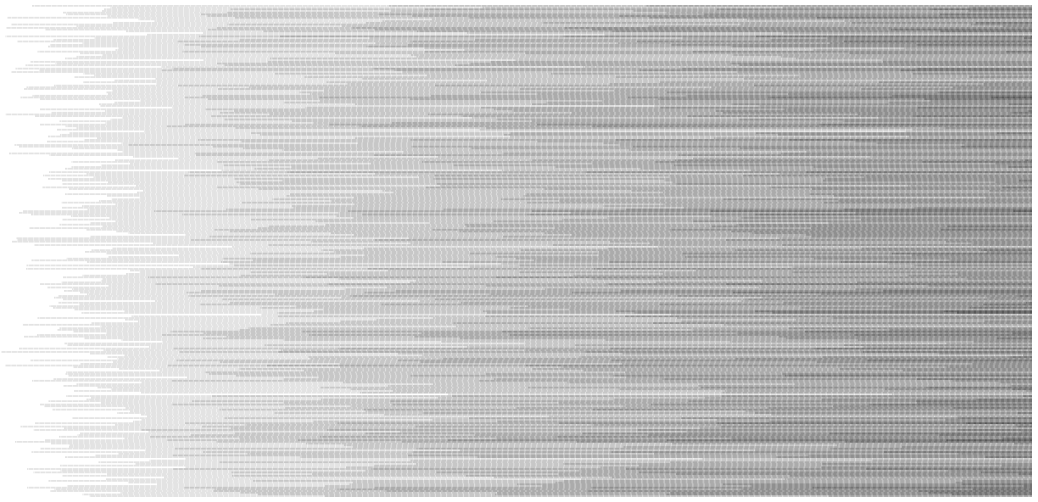


(c) Pixel-plot showing the time spent acting for the *learn-concept* approach.

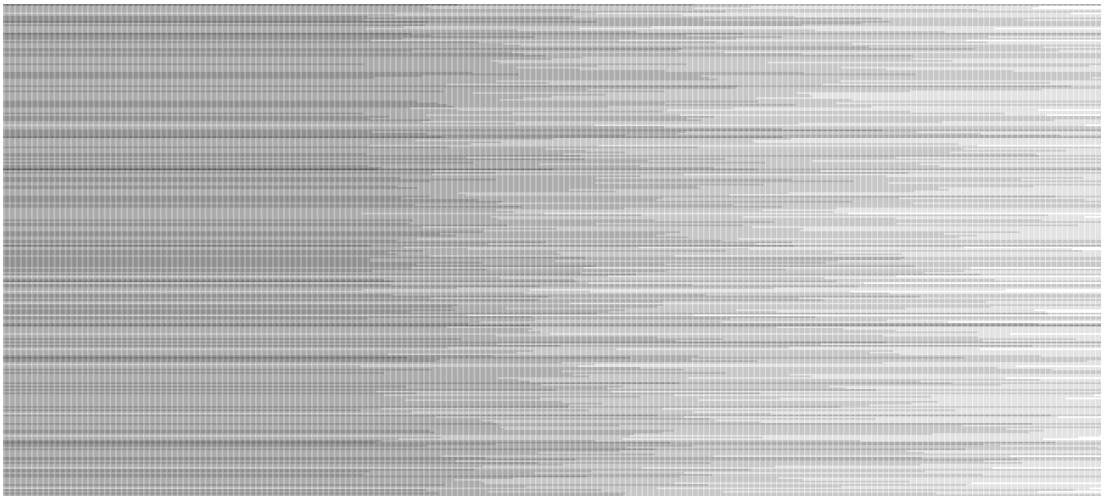
FIGURE 4.13: Pixel plots for the *learn-concept* approach.



(a) Pixel-plot showing the time spent learning for the *learn-repeated* approach.

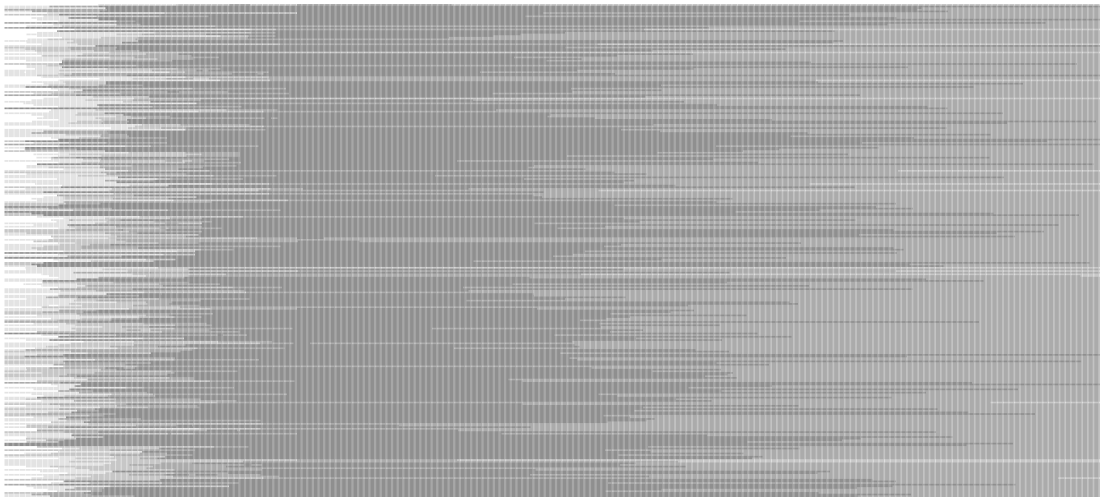


(b) Pixel-plot showing the time spent deliberating for the *learn-repeated* approach.

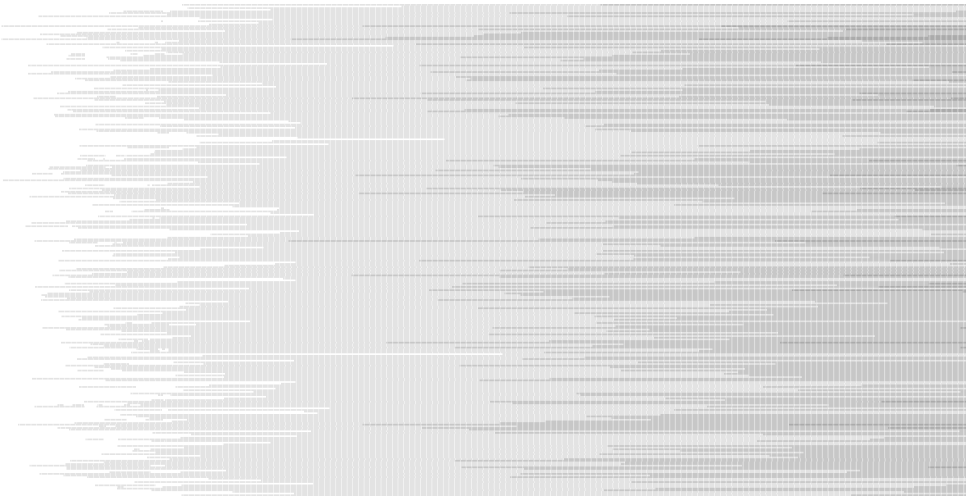


(c) Pixel-plot showing the time spent acting for the *learn-repeated* approach.

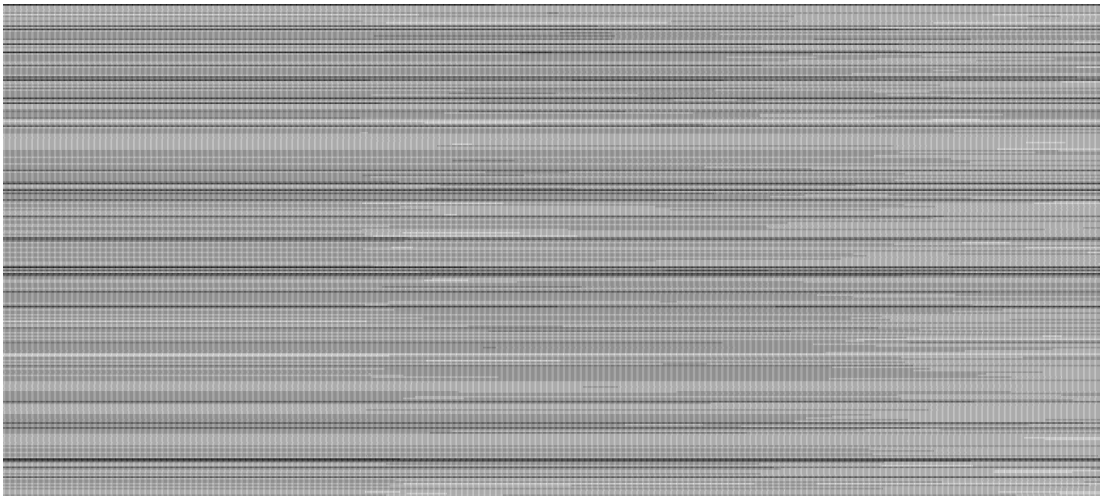
FIGURE 4.14: Pixel plots for the *learn-repeated* approach.



(a) Pixel-plot showing the time spent learning for the *learn-fragment* approach.



(b) Pixel-plot showing the time spent deliberating for the *learn-fragment* approach.



(c) Pixel-plot showing the time spent acting for the *learn-fragment* approach.

FIGURE 4.15: Pixel plots for the *learn-fragment* approach.

4.4.4 Hypothesis 4 — RoboCup Rescue Results

In terms of RoboCup Rescue, our learning approach is able save 21.8% more civilians and 40.6% more of the city compared with any other approach. It is able to extinguish the whole city 102 out of 250 simulations, while the others did not manage to extinguish the whole city. However, the other approaches could not always control the fire, and the city's buildings burned out using the *no-collaboration* (250/250), *learn-everything* (250/250), *learn-concept* (164/250), and *learn-repeated* (168/250) approaches. Our approach was able to control the fires because it spends less time learning and deliberating about its actions, so it can act faster and thus has a better chance at extinguishing fires early in the simulation so they do not spread (see Section 4.4.3). The agents using our approach are able to save 54.7% of the civilians, because the fire agents control the spread of the fire and the ambulance agents spend less time learning and deliberating, than the other approaches.

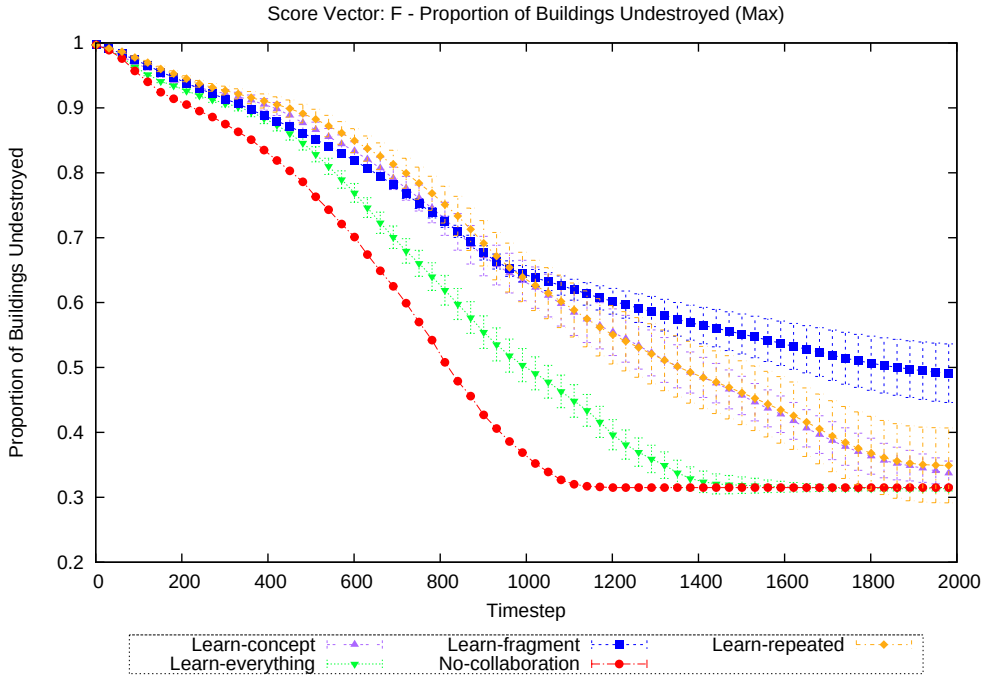


FIGURE 4.16: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

We summarise the behaviour of all of the approaches:

- **The *learn-everything* approach** augments all concepts received from the environment ontologies into the agent's private ontology. However, in contrast to the other approaches, its performance is hindered by the time taken to access the

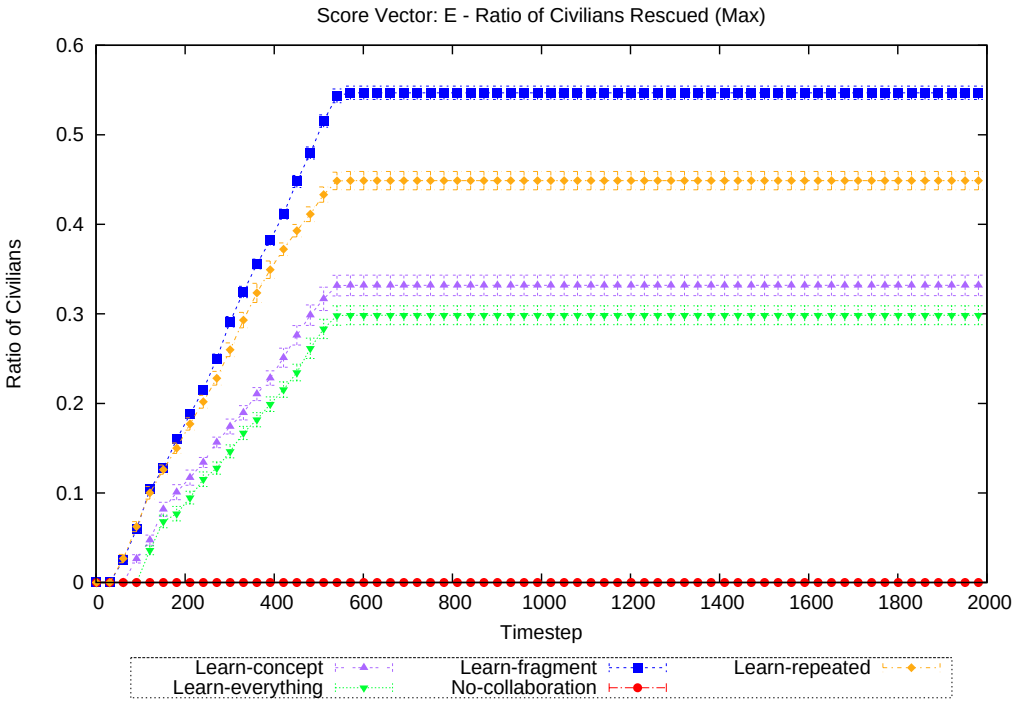


FIGURE 4.17: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

concepts in the agents' ontologies (as shown in Figure 4.12(b)). Thus, these agents did not save as many targets compared with the other approaches (as shown in Table 4.4 and Figures 4.16 and 4.17);

- **The *learn-fragment* approach** enables the agents to learn more than one concept at a time, therefore this approach enables the agents to communicate with the environment ontologies in less time than the other approaches that augmented their ontologies. This approach enables agents to spend less time deliberating their actions than the other approaches, except for the *no-collaboration* approach, which has more light coloured pixels than Figure 4.15(b)). Thus, saves more targets (as shown in Figures 4.16 and 4.17);

- **The *learn-repeated* approach** enables agents to only learn concepts that are used more than once in a simulation. This optimises the time spent augmenting its ontology and time accessing knowledge for rescuing targets because the agents' ontologies only holds information that is useful for the scenario. This approach requires perfect foresight so that it knows which concepts will be required more than once in a scenario, this is unrealistic in rescue scenarios;

- **The *learn-concept* approach** enables the agents to learn one concept at a time from the environment ontologies. Thus, agents using this approach spend longer deliberating their actions compared to agents using the *learn-fragment* and *no-collaboration* approaches (shown in Figure 4.13(b));

- **The *no-collaboration* approach** was able to attend targets faster than other approaches because it spends no time deliberating about its actions. Therefore it can save its targets faster, however the civilians have on average lower health because they cannot select equipment that is optimal to save them (shown in Figure 4.17). This ability to select equipment affects the area of the city damaged by fires and is the worst performing approach in this aspect (shown in Figure 4.16).

In summary our approach, *learn-fragment*, enables agents to balance the trade-off between the time spent decided on which actions to take, and the time spent moving and saving targets. This balance is enabled by allowing the agent to augment a fragment of knowledge to its ontology, thus benefiting from reducing the number of communications with the environment ontologies than the *learn-concept* approach (as shown on Figure 4.13(b), which has more light coloured pixels than Figure 4.15(b)) and maintaining an ontology 36.6% of the size of the *learn-everything* approach's ontology.

4.5 Summary

In this chapter, we presented and evaluated a learning technique that allows agents to augment their ontologies with concepts from other ontologies in their environment. Our approach focuses on learning concepts related to an agent's domain, enabling a higher degree of connectedness between concepts in an agent's ontology, than comparative state-of-the-art approaches. We have published our reactive learning algorithm and our evaluation in Packer et al. (2010a). The results from our evaluation support the hypotheses presented in Section 4.3, by showing that:

1. Augmenting an agent's ontology reduces the number of messages required to complete tasks, compared to agents that learn nothing. This is because our approach learns more than one concept at a time (see Section 4.4.1);

2. Our approach evolves its ontology by incorporating domain-focused concepts and it completes the most tasks with the smallest ontology and estimated complexity compared with the other learning approaches (see Section 4.4.2);

3. Our approach spends the least time learning, deliberating and more time acting compared with the other approaches (see Section 4.4.3);

4. Our approach outperforms the other approaches that do not have perfect foresight, in terms of RoboCup Rescue, by saving on average more civilians and more of the city (see Section 4.4.4).

In this chapter, we have satisfied Requirement 2a from Section 1.3.2 by developing an algorithm that incorporates new domain-focused knowledge into the agent's ontology, which uses a methodology designed to reduce the overhead resource costs of learning regularly required concepts. We have also satisfied Requirement 3 from Section 1.3.3 with our empirical evaluation, where we have assessed the performance of our learning algorithm by comparing the following measures: the time spent learning; the number of messages sent by each agent; the number of successful tasks completed; the size of an agent's ontology; and their performance in RoboCup Rescue using its score vector.

The majority of our results were as expected, however, we did not anticipate that the performance of the *learn-repeated* approach would save fewer civilians and extinguish fewer fires than the *learn-fragment* approach. We expected that because this approach had perfect foresight it would be the most efficient because it would not augment concepts that were used only once, and therefore could maintain the smallest ontology out of the compared approaches. However, this approach was not the most efficient because it has to check each time it learns a concept whether it will be used in the future, while the other approaches did not have to perform this check. Thus, the resource overhead of *learn-repeated* was higher, and it performed worse than expected.

While the results in this chapter show that our approach is outperformed by the *learn-repeated* approach (with respect to the ontology complexity and the total time spent acquiring and learning concepts), the *learn-repeated* approach requires access to all future queries. Our approach, as well as the other benchmark approaches (*learn-concept*, *learn-everything*, and *learn-nothing*) receive the tasks sequentially, and therefore cannot make this optimisation. However, it is unlikely that in a real situation that the agent applying the *learn-repeated* approach would know all future tasks. We note that in an environment with a high chance of repeated tasks, our agent would benefit from predicting which concepts would be useful in the future. Thus, we proceed in the next chapter to present and evaluate an approach that can predict which concepts will be useful in the future.

Chapter 5

A Proactive Learning Algorithm for Evolving Ontologies

This chapter describes our proactive learning algorithm which evaluates the likelihood of a future need to learn specific concepts, so that it can more efficiently learn concepts before they are required. Specifically, our proactive learning algorithm prioritises augmenting an ontology with concepts that have a high likelihood of use for a current task based on completed tasks, using a prediction model. We evaluate this algorithm against our reactive learning algorithm which was shown to be effective in the RoboCup OWLRescue environment. Thus, this chapter fulfils part *b)* of both of our second and third requirements which requires the use of a prediction model to select concepts to learn, and requires the evaluation of this approach, respectively.

To this end, the next section provides motivation for using prediction for learning concepts. In the following section, we detail our proactive learning algorithm. Following that, we evaluate our algorithm against other benchmark algorithms and in the final section we conclude. A glossary of the terms used and those introduced in this chapter are given in Appendix F.

5.1 The Motivation for Prediction to Aid Learning

In order to motivate the need for using prediction while selecting concepts to augment into an agent's ontology, we identify the following key benefits:

1. Increase the likelihood of incorporating knowledge that will enable an agent to complete a task, by modelling the concepts which were useful in completing similar past tasks;

2. Decrease the likelihood of incorporating knowledge which has not been useful in completing past task, and using a prediction model to show which concepts did not contribute to the completion of similar past tasks;

3. Improve performance by maintaining a smaller and more domain focused ontology compared to a reactive algorithm, while also increasing the likelihood of the usefulness of incorporated knowledge;

4. Decrease the amount of time spent processing concepts because by using a prediction model we can eliminate the consideration of learning a concept if it has not been used to support similar tasks in the past.

Given these motivations, we build upon our reactive learning algorithm by enabling an agent to record and consult their own Markov model which captures the sequence of concepts learned and the likelihood of that sequence. Our algorithm uses Markov Models because their use enables our algorithm to fulfil Requirement 2c, as discussed in Section 2.4.3. This Markov model enables an agent to predict concepts which had the highest likelihood of use after requesting the definition of a specific concept. This prediction takes the form of a Markov chain which predicts the concepts which have the highest likelihood of use. Using such a Markov chain, we request a set of concepts that have the highest likelihood of use and select concepts from the received set of fragments to augment into the agent's ontology.

There are two processes required to enable an agent to use the Markov model for prediction. First, an agent must maintain the information in the Markov model by updating it with concepts requested during use. Second, an agent uses the information in the Markov model to select which concepts to augment into its ontology. Thus, we describe these two processes in the following two sections.

5.2 Building an Agent's Markov Model

During the RoboCup OWLRescue simulation an agent requests the definition for concepts in order to complete tasks. We use these requested concepts to build a Markov model for each agent so that it can predict which concepts it will require next. The use of Markov models to predict concepts to learn is novel because they have not been previously applied to the incorporation of concepts into ontologies. In order to build a Markov model, our agent learning scenario must be modelled as a directed graph; we have devised the following method to create our Markov model, which enables us to fulfil Requirement 2c (see Section 1.3.2) of using the Markov model to predict agent learning behaviour, because the Markov model will probabilistically select concepts to learn. Specifically, to make the Markov model, an agent creates a directed graph where the nodes represent concepts and the edges represent the probability that the successor

concept is requested after its predecessor. For example, Figure 5.1 illustrates a Markov model which has four nodes and five edges. The chance that the concept denoted by a is followed by its successor b is 100%, whereas b is followed by c or d, 70% and 30%, respectively.

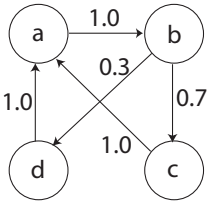


FIGURE 5.1: An example of a Markov model.

In order to situate a Markov model within our framework, presented in Chapter 3, we introduce the following notation. Formally, each task agent a_t has its own ontology o, where $o = \langle do, eo \rangle$ and a Markov model m, where $m \in M$ and M is the set of all Markov models. The Markov model $m = \langle V, E, W \rangle$ where V is a set of nodes, E is a set of edges ($\langle V_i, V_j \rangle, V_i, V_j \in V$), W is a function mapping edges to weights, $W : E \rightarrow [0, 1]$, contained in the model.

When an agent learns a concept denoted by v_o immediately after the concept in v_x, the agent updates its Markov model, by either:

1. Adding a node v_o to its model m if $v_o \notin V$, such that $V' = \{v_o, v_1, \ldots v_n\}$ where V' is the set of nodes after the addition of v_o. The agent also adds the edge e_o to E, where $e_o \langle v_x, v_o \rangle W(e_o) = 1$ such that $e_o \notin E$ and $E' = \{e_o, e_1, \ldots, e_n\}$ where E' is the set of edges after the addition of v_o to m;

2. Updating the value of the edge $e_o(v_x, v_o)$ which connects the nodes v_x and v_o, using Equation 5.1;

$$e'_o(v_x, v_o) = \frac{e_i(v_x, v_o) + 1}{\sum_{e \in E(v_x)} e + 1} \tag{5.1}$$

   where $E(v_x)$ is the set of edges that contain v_x as a predecessor node, and e'_o is the edge after m is updated.

We illustrate the node addition process in detail in Algorithm 10.

In addition to updating a Markov model, an agent can also extract a Markov chain mc^c_m from the Markov model m, representing a node denoting concept c. The chain mc^c_m contains a subset of nodes v_{mc} and edges e_{mc} from m, where $v_{mc} \subset V$ and $e_{mc} \subset E$. Specifically, v_{mc} is a set of nodes which are connected by edges (e_{mc}) that have the highest

Algorithm 10 Algorithm that details the addition of a node to a Markov model.

Require: $parentConcept \leftarrow$ (`the last concept learned`)
Require: $concept \leftarrow$ (`concept to add to Markov model`)
Require: $model \leftarrow$ (`existing Markov model`)

Function: $getLinksFrom(node)$ `returns set of all possible edges from` *node*
Function: $calculateProbability(parent, child)$ `calculates the probability from` *parent* `node to` *child* `node`
Function: $model.setProbability(parent, child, probability)$ `sets the` *probability* `from` *parent* `node to` *child* `node in the model`

```
1: /* add the new edge to the model's graph */
2: model.addEdge(parentConcept, concept)
3: /* loop through all nodes linked from the parentConcept */
4: for node ∈ model.getLinksFrom(parentConcept) do
5:   /* recalculate the probabilities from parentConcept to all con-
     nected nodes */
6:   newProbability ← calculateProbability(parentConcept, node)
7:   model.setProbability(parentConcept, node, newProbability)
8: end for
9: return model
```

weightings. We illustrate the Markov chain extraction process in detail in Algorithm 11, which returns an ordered set of concepts, ordered by probability.

An agent creates a Markov model and updates it through use and thus it evolves over time. The evolution of a Markov model increases the accuracy of its representation of an environment, in our case RCOR, over time because it observes which concepts are required more frequently and is less likely to be affected by the occurrence of infrequently encountered concepts. It is possible that in real life situations the concepts used for a task may change due to different recommendations over time. For example, an agent requests to learn about a particular medication given a certain symptom but this medication is superseded by another because it has fewer side effects. However, this situation will not occur in the RoboCup OWLRescue simulation because recommendations do not change over time. Despite the fact that our simulation does not change its recommendations over time, we update an agent's Markov model through use so that it can more accurately predict which concepts are required. In order to provide a prediction to an agent its Markov model is required to be trained, so that it contains all of the concepts used in the scenario.

As illustration, we now describe how an agent builds a Markov model in the following example. We use the example of a fire brigade agent which learns, in the following order, the concepts: `firefighting_motorcycle`, `motorcycle_firefighters`, `mist_motorcycle`, `car`, `police_car`, `hazardous_chemical`, `aluminium_phosphide`, and `aluminium_carbide`.

Algorithm 11 Algorithm that details the extraction of a Markov chain from a Markov model, returning an ordered set of concepts, sorted by probability.

Input: c_t: The target concept

Function: $getAverage()$ `returns the average number of concepts used to complete a task`

Function: $getHighestNodes(concept)$ `returns a set of nodes in the the form` $\langle parent, node, value \rangle$

```
1: Nodes ← getHighestNodes(c_t)
2: Chain ← Nodes
3: /* iterate for the number of concepts required to complete a task */
4: for a ∈ {1 ... getAverage()} do
5:   Nodes3 ← ∅
6:   /* loop over next highest nodes */
7:   for n ∈ Nodes do
8:     /* add the next highest nodes to the returned chain */
9:     Nodes2 ← getHighestNodes(n.getConceptLabel())
10:    Chain ← Chain ∪ Nodes2
11:    /* use these nodes in the next iteration */
12:    Nodes3 ← Nodes3 ∪ Nodes2
13:  end for
14:  /* reset the next highest nodes for the next loop */
15:  Nodes ← Nodes3
16: end for
17: return Chain
```

1. The fire brigade agent learns about `firefighting_motorcycle` because it requires fast response to fires during heavy traffic. This is the first concept the agent is required to learn about, and thus it initialises a Markov model and adds `firefighting_motorcycle` as its first node (see Figure 5.2);

FIGURE 5.2: A Markov model initialised with a single node.

2. The agent then desires to learn about `motorcycle_firefighters` because they are required to operate the `firefighting_motorcycle`. This node is connected to `firefighting_motorcycle` by a directed edge because it was its successor. So far, given this input, `motorcycle_firefighters` always follows `firefighting_motorcycle` thus the edge's value is 1.0, see Figure 5.3;

3. The agent desires to learn about types of `firefighting_motorcycle` because it requires that the fire suppressant is applied using a mist nozzle, hence it learns about the concept `mist_motorcycle`, shown in Figure 5.4;

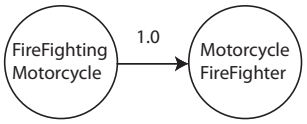


FIGURE 5.3: A Markov model with two nodes, and a single edge denoting a probability of 1.0.

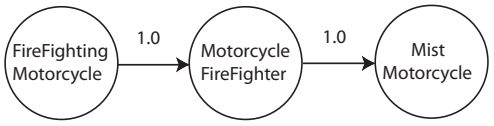


FIGURE 5.4: A Markov model containing three nodes and two edges.

4. The agent then learns the concepts `car`, `police_car`, `hazardous_chemical`, `aluminium_carbide`, and `aluminium_phosphide`. This is shown in Figure 5.5.

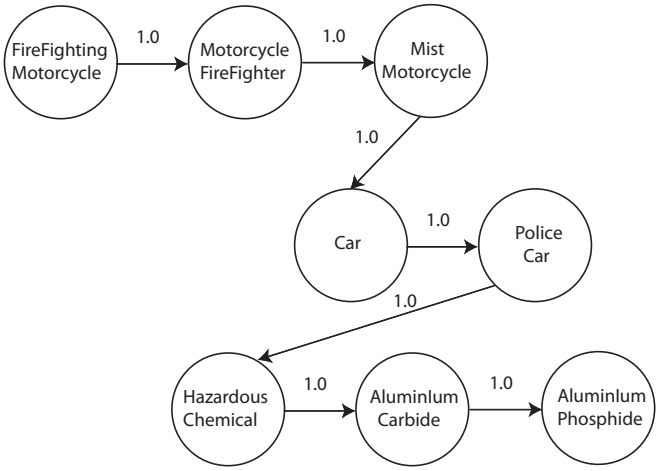


FIGURE 5.5: A Markov model containing eight nodes and seven edges.

This Markov model is created from one simulation run, and thus in this example is unlikely to provide an accurate prediction. This is because after one run only a single relationship from each concept exists, with the probabilities on all edges set to 1.0. Therefore, in order to train the Markov model we update the model over various iterations, which provide multiple possible states from concepts, with the probabilities shared among them. We extend our example with the following set of concepts which were requested by the same fire brigade agent: `firefighting_motorcycle`, `motorcycle_firefighters`, `water_motorcycle`, `hazardous_chemical`, `aluminium_phosphide`, and `aluminium_carbide`.

1. The fire brigade agent learns about `firefighting_motorcycle` because it requires fast response to fires during heavy traffic, and then learns about `mo-`

`torcycle_firefighter` because they are required to operate the `firefighting_motorcycle`. Learning these concepts leaves the edges of the Markov model unchanged because `motorcycle_firefighter` has so far been followed by `firefighting_motorcycle` (see Figure 5.5);

2. The agent then desires to learn about types of `firefighting_motorcycle` and because it requires that the fire suppressant is water it learns about `water_motorcycle`. This changes the structure of the Markov model because it now branches from `motorcycle_firefighter` to the nodes `mist_motorcycle` and `water_motorcycle`. The edge values also change to the mean average of all of the inputs: given the current input there is an equal chance that `mist_motorcycle` and `water_motorcycle` are required to complete a task that uses a `firefighting_motorcycle` (see Figure 5.6).

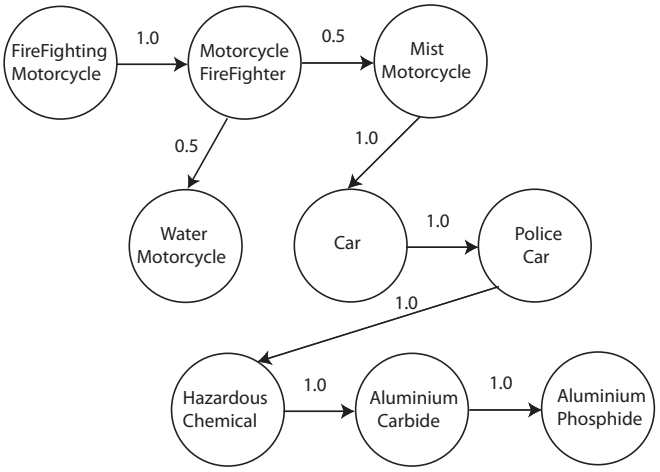


FIGURE 5.6: A Markov model showing the addition of `water_motorcycle`, as a possible next state from `motorcycle_firefighter`.

3. The agent then learns the concepts `hazardous_chemical`, and `aluminium_carbide`, thus resulting in the Markov model presented in Figure 5.7.

We create a Markov model by iterating over many simulations using the above method. This produces a rich structure which can be used to select which concepts to augment into an ontology. We discuss the process of using the Markov model in the next section.

5.3 The Proactive Learning Algorithm

In order to describe our proactive algorithm, we use the same example from our reactive learning algorithm (see Section 4.2) so that we can highlight their differences. Agents using our proactive learning algorithm maintain a Markov model that allows them to predict which concepts will be required in the future. Thus, we use the following example:

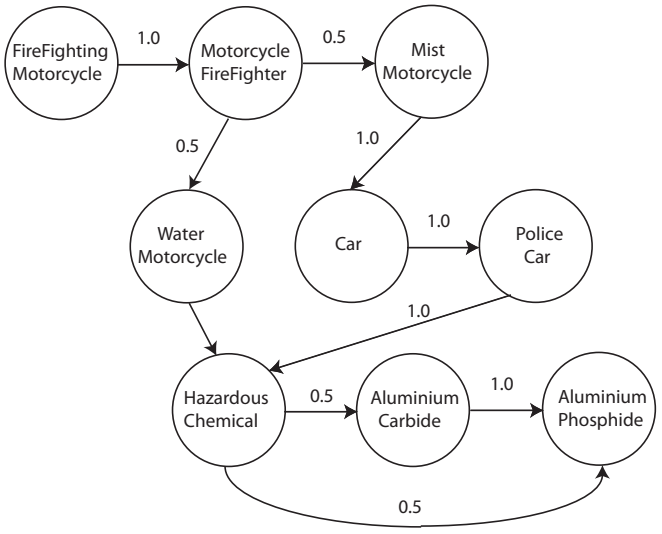


FIGURE 5.7: A Markov model updated with new edges to `hazardous_chemical` and `aluminium_carbide`.

a rescue agent a_t has a private ontology o_t that contains the concepts shown in Figure 5.8, and its own Markov model m shown in Figure 5.9. The Markov model in Figure 5.9 depicts an extract of m, where the stack of grey nodes represent many concepts and the value that connects them is the sum of all the edges connecting them. It desires to learn the concept `firefighting_motorcycle` as used in Japan, because it currently does not use fire fighting motorcycles but would like to be able to rush to the scene of a fire disaster, without concern for such problems as traffic jams and narrow residential roads.

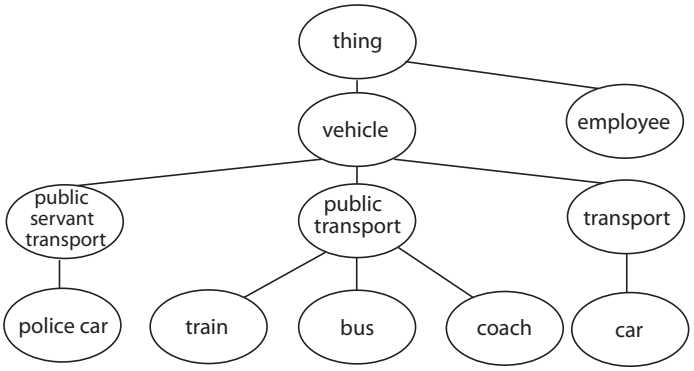


FIGURE 5.8: An example rescue agent's ontology.

Using a Markov model can reduce the cost of acquiring information in two ways: first, by requesting the definition for more than one concept at a time, thus reducing the number of messages sent by an agent; second, by selecting and eliminating concepts to choose from when augmenting an agent's ontology which have a high or low chance of being used, respectively. This increases the chance of augmenting an ontology with concepts that will be useful to complete a task and reduce the chance of augmenting concepts that are unlikely to be used. This means that an ontology should evolve to

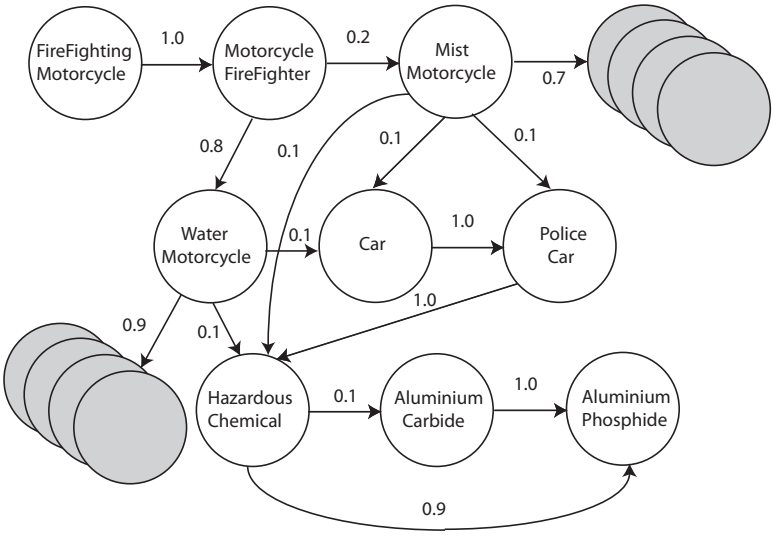


FIGURE 5.9: An extract from the example rescue agent's Markov model, where the grey nodes indicate where the extract connects with the Markov model.

be smaller than our reactive learning algorithm but still enable the completion of tasks. To this end, our next two sections describe how an agent uses its Markov model to: select which concepts to request fragments about; and, select and eliminate concepts from those fragments using the reactive learning algorithm.

5.3.1 Using Prediction for Requesting Concepts

An agent desires to learn about a concept, and finds the node which corresponds to that concept in its Markov model. The agent then extracts a Markov chain comprised of the nodes that are connected with the highest value edges. If two or more edges from the same node have the same highest value, both paths are included in the chain. In our example, a fire brigade desires to learn about `firefighting_motorcycle` and extracts the chain shown in Figure 5.10.

In order for an agent to decide how many concepts it will request from the environment agents, it calculates the average number of concepts it requires per task so that it can estimate how many nodes to use in the chain. This average is calculated for each task performed. We do this because it is likely that over the course of a simulation the agent will have to learn fewer concepts compared to the early stages of the simulation, where the agent's ontology only contains concepts that support its basic operation. We illustrate in the concept request process in Algorithm 11.

In our example, the agent calculates that it requires three nodes from the Markov chain, these include the nodes `firefighting_motorcycle`, `motorcycle_firefighter`, and `water_motorcycle`, and requests fragments from the agents that own the environment ontologies. Thus, the agent receives two fragments from the two environment ontologies:

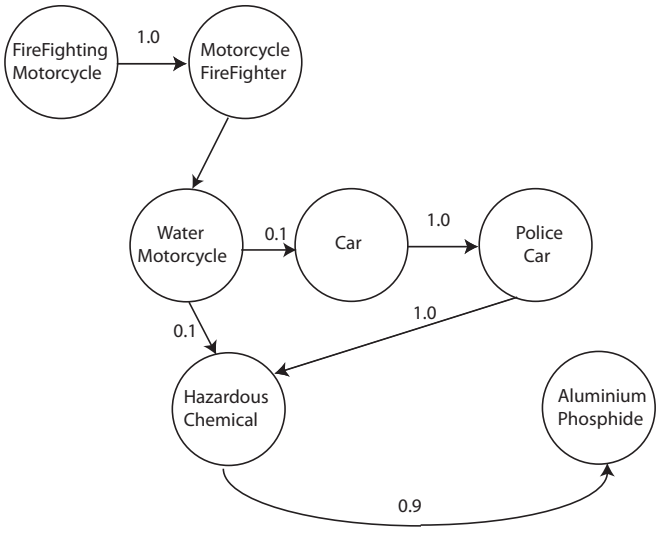
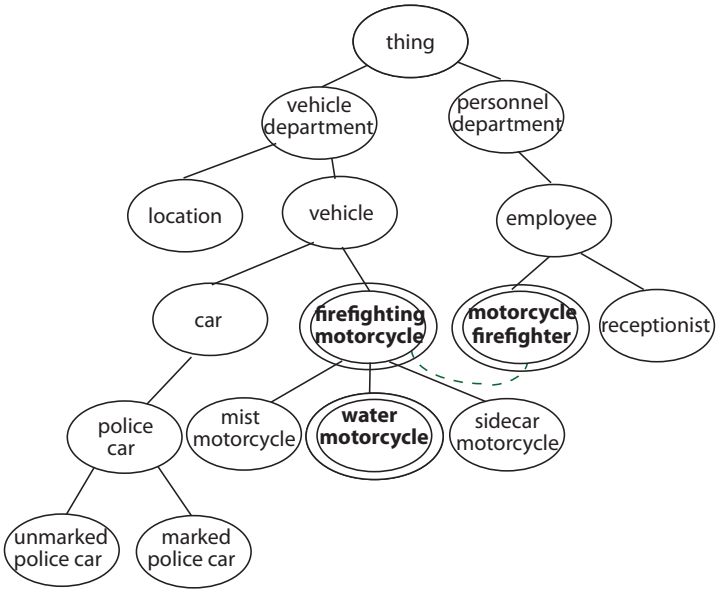
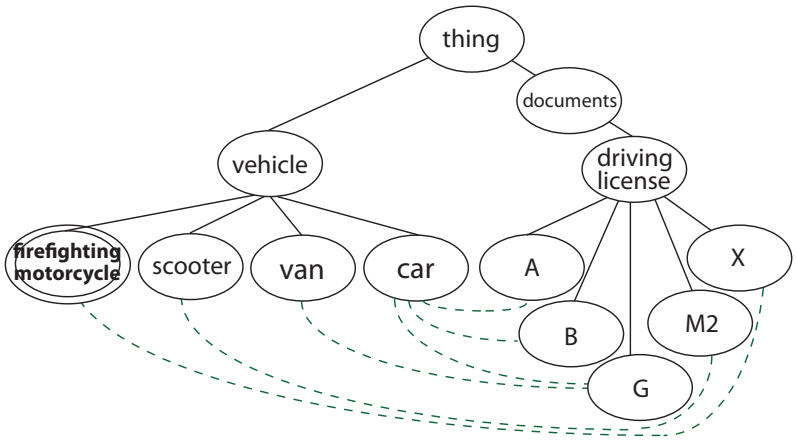


FIGURE 5.10: An extracted chain from the example rescue agent's Markov model, with a root node representing `firefighting_motorcycle`.

the first fragment contains all three of the required concepts shown Figure 5.11(a); and the second fragment contains only one of the requested concepts (see Figure 5.11(b)), this is because the DVLA ontology does not contain information on the concept `motorcycle_firefighters` or `water_motorcycle`).



(a) First fragment, f_1, which represents the concepts `firefighting_motorcycle`, `water_motorcycle` and `motorcycle_firefighter`.



(b) Second fragment, f_2, which represents the concept `firefighting_motorcycle`.

FIGURE 5.11: Two fragments, received from specialist agents in response to a query containing the concepts `firefighting_motorcycle`, `water_motorcycle` and `motorcycle_firefighter`, where the dashed lines represent relationships between two concepts.

Prediction allows an agent to request fragments representing more than one concept. Thus, an agent can reduce the number of times an agent requests fragments compared to using a reactive approach.

5.3.2 Using Prediction for Concept Selection

The fragments that the agent receives are then merged (using the process described in Section 4.2.1), so that the agent can preselect which concepts to use to select from using the Hierarchical and Relational selection process (described in Sections 4.2.3 and 4.2.4). The agent classifies each concept in the merged fragment into one of two categories: contained in the Markov Chain, or not present in the Markov model. Using this categorisation the agent can prioritise learning concepts that are contained in the Markov Chain and remove concepts that are unlikely to be used within the simulation.

In more detail, the agent analyses the concepts in the merged fragment in two stages.

- **Stage 1:** Rank the concepts that appear in the Markov Chain. The agent prioritises learning concepts in the order they appear in the Markov chain and eliminates the selection of concepts that do not appear in the Markov chain. In order to achieve this selection, the agent prioritises concepts in the chain by ranking the concepts in reverse rank order so that the concept connected by the most edges to the starting node has the lowest weighting and the first node has the highest. This process enables the agent to select the concepts that have the greatest probability of being required in future, and therefore are potentially the most useful

for the agent to learn. In our example, the concepts' weightings are shown in Table 5.1;

Concept	Markov chain order	Weighting
FireFightingMotorcycle	1	5
MotorcycleFireFighter	2	4
WaterMotorcycle	3	3
Car	4	2
PoliceCar	5	1
HazardousChemical	4	2
AluminiumPhosphide	5	1

TABLE 5.1: Concepts in the Markov, showing the order of the concepts in the chain, and their corresponding weighting.

- **Stage 2:** Locate concepts that are not common to both the merged fragment and the agent's Markov model. These concepts are not factored into the selection process and will not be incorporated into the agent's ontology. We do not remove these concepts from the merged fragments because removing concepts from a fragment incurs a greater cost than calling a Boolean function which checks whether a concept is in a list.

  In our example, the agent determines that the concepts `location`, `receptionist`, `scooter`, `van`, `driving_license`, `A`, `B`, `G`, `M2`, and `X` are not present in the agent's Markov model. In Figure 5.12, we depict the merged fragment, where the white, yellow, and red nodes represent concepts that are contained in the Markov chain, concepts that are present in the Markov model, and concepts that are not in the Markov model, respectively.

In order to select which concepts an agent augments into its ontology, the agent processes the fragment using the hierarchical and relational selection algorithms presented in Sections 4.2.3 and 4.2.4. During the hierarchical selection process we calculate the concept rating for each concept using Equation 5.2, which weights the concept rating by the concept rankings from the Markov chain, so that the hierarchical selection will include the most likely concepts during its selection.

$$conceptrating'_c = conceptrating_c * rank \tag{5.2}$$

where $conceptrating'_c$ and $conceptrating_c$ is the concept rating used for the proactive and reactive learning approaches, respectively, and $rank$ corresponds to the rank weighting during stage 1. We illustrate the above process in Algorithm 12.

In our example, during the hierarchical selection phase the agent calculates the concept ratings for each concept (shown in Figure 5.13) and selects which concepts to learn

Algorithm 12 Algorithm that details the process of using prediction to select concepts.

Require: $F_d \leftarrow$ (`set of merged fragments from the environment`)
Require: $c_t \leftarrow$ (`concept of interest`)
Require: $model \leftarrow$ (`agent's Markov model`)

Function: $length(Chain)$ `return the length of a Markov chain`
Function: $model.getChain(concept)$ `return a Markov chain for a specific concept`
Function: $model.containsConcept(concept)$ `returns true if the model contains the concept`
Function: $chain.getDistance(a, b)$ `returns the number of edges (distance) from` a `to` b `in the Markov chain.`
Function: $hierarchicalSelection(concept, fragment)$ `returns a fragment that has hierarchical selection performed on it, based on a` *concept* `(see Section 4.2.3).`
Function: $relationalSelection(concept, fragment, weights)$ `returns a fragment that has relational selection performed on it, based on a` *concept* `(see Section 4.2.4).`

1: $AllWeights \leftarrow \emptyset$
2: `/* calculate weightings for all concepts in the fragments, based on their distance from the target concept in the Markov chain */`
3: **for** $c \in F_d$ **do**
4: $Chain \leftarrow model.getChain(c)$
5: `/* set the weight to the inverse of the distance in the chain, so that closer concepts are weighted highest */`
6: $weight \leftarrow length(Chain) - Chain.getDistance(c_t, c)$
7: $AllWeights \leftarrow AllWeights \cup \{\langle c, weight \rangle\}$
8: **end for**
9: `/* perform hierarchical selection on the fragment, instead of using the concept rating (see Equation 4.1) we use the Markov concept rating (see Equation 5.2). */`
10: $C^h \leftarrow hierarchicalSelection(c_t, F_d, AllWeights)$
11: `/* perform relational selection on the output of the hierarchical selection */`
12: $C^r \leftarrow relationalSelection(c_t, C^h)$
13: $ChosenConcepts \leftarrow \emptyset$
14: `/* loop through the concepts selected by the hierarchical and relationship selection algorithms */`
15: **for** $c \in getConcepts(C^r)$ **do**
16: `/* only return concepts that are present in the markov model */`
17: **if** $model.containsConcept(c)$ **then**
18: $ChosenConcepts \leftarrow ChosenConcepts \cup \{c\}$
19: **end if**
20: **end for**
21: **return** $ChosenConcepts$

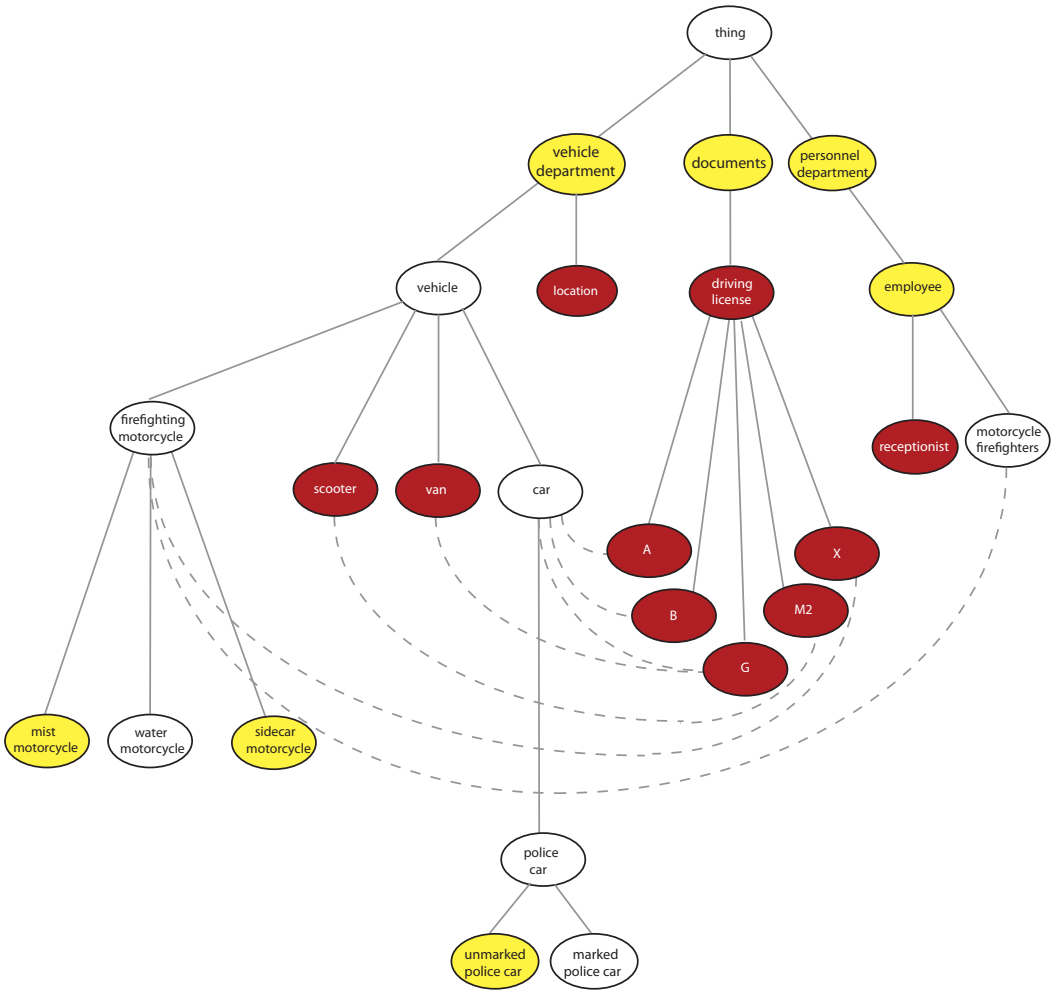


FIGURE 5.12: This ontology fragment represents concepts, that are contained in the Markov chain, present in the Markov model, and not present in the Markov model, by white, yellow, and red respectively.

according to this rating.

After the hierarchical and relational selection algorithms, the agent has chosen to augment the concepts, `vehicle`, `firefighting_motorcycle`, `water_motorcycle`, `car`, `police_car`, and `motorcycle_firefighters` (shown in Figure 5.14).

This set of concepts selected for the agent to learn differs from our reactive learning algorithm. Specifically, the reactive learning algorithm selected the concepts `driving_license`, `X`, `employee`, `mist_motorcycle` and `sidecar_motorcycle`, whereas the proactive learning algorithm chose to disregard these concepts. The learning algorithm chose to learn about the concepts `car` and `police_car`, whereas the reactive learning algorithm did not. The proactive learning algorithm selects concepts which have a higher likelihood of use, the `car` and `police_car`, whereas the reactive algorithm did not select these concepts and chose to learn about the concepts `driving_license` and `X` which do not have a high likelihood of use.

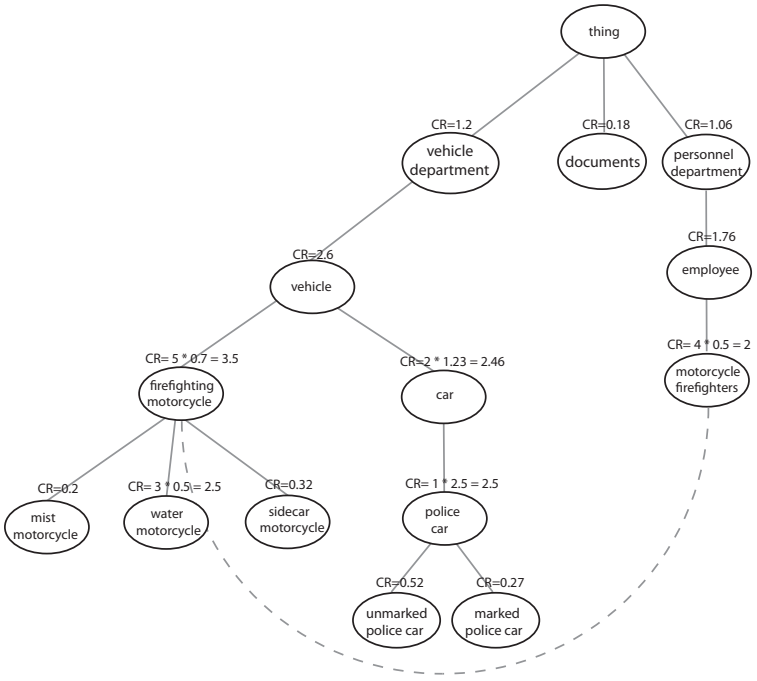


FIGURE 5.13: The concept ratings (shown as CR) of the concepts. The concepts in the Markov model are shown multiplied by their rank weighting.

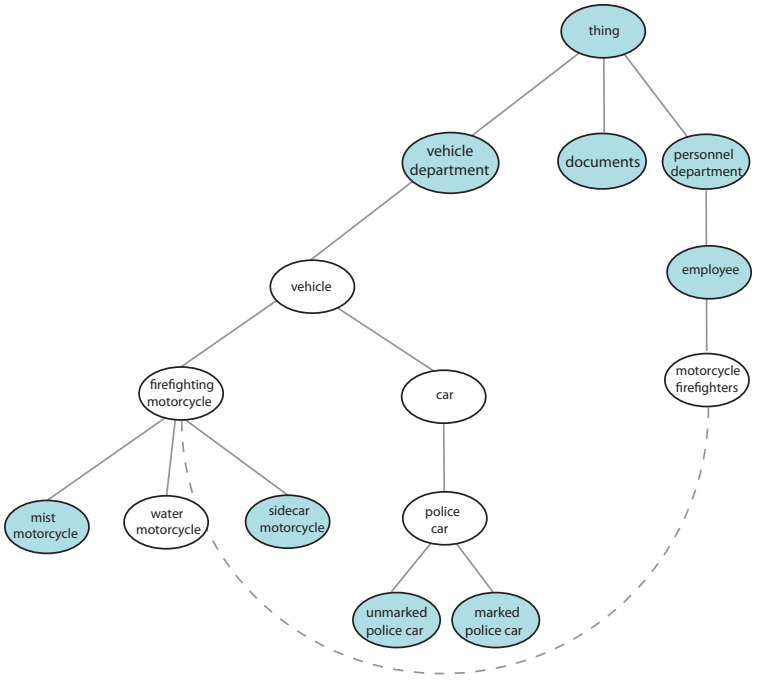


FIGURE 5.14: The concepts selected using prediction to augment into an agent's ontology.

The proactive algorithm can perform better than other approaches discussed in this thesis, because it can compare the agent's current state to previously encountered states in order to learn additional concepts that it predicts will be useful. In doing so, it reduces the number of learning operations an agent performs, which reduces the overhead costs associated with learning. Thus, we hypothesise that the proactive learning will be able to augment an agent's ontology with concepts that have a higher likelihood of use compared to the reactive learning algorithm. We now proceed by describing our empirical evaluation.

5.4 Empirical Evaluation

This section discusses the experiments in our empirical evaluation and the results. Our experiments are used to evaluate the performance of our proactive learning algorithm, benchmarked against our reactive learning algorithm (presented in Section 4.2) and other control learning approaches (detailed in Section 3.3.1).

To this end, we present our hypotheses and how we intend to show them in Section 5.4.1. The next section provides the specific parameters of our experiments. In the following section, we present and discuss the results from the experiments.

5.4.1 Hypotheses

This section presents our hypotheses which show how our approach addresses the contributions detailed in Section 1.4. In more detail, our proactive learning algorithm is required to:

1. Demonstrate a fully automated technique that enables an agent to learn;
2. Augment an agent's ontology, and demonstrate that this is an effective way for an agent to answer continual queries in a specific domain.

In order to demonstrate that our approach addresses our contributions and meets our requirements (see Section 1.3), we aim to evaluate the following hypotheses:

Hypothesis 1: Augmenting an agent's ontology with concepts that are predicted to be useful will reduce the number of messages required to complete tasks, compared to agents that do not use prediction. Specifically, our approach will perform better than our reactive approach and other state-of-the-art approaches because they will predict which concepts have a high chance of being required in the future, and thus can prioritise learning them, and reduce the number of messages

sent. This hypotheses addresses Requirement *2a* which aims to reduce the costs of re-acquiring regularly required knowledge. Such a reduction will show that the proactive learning agent has fewer costs involved in completing tasks compared to the other approaches. We hypothesise this is because there are costs related to sending messages over a network and computational costs to compose messages;

Hypothesis 2: Our approach will enable an agent to augment its ontology with relevant concepts, based on its domain and the likelihood of their use. This is because our proactive learning algorithm uses a Markov model to model the likelihood of the progression of concepts requested over many simulations, and can use this as a prediction method to select concepts that are relevant. In order to incorporate concepts that related to an agent's domain, our proactive algorithm also makes use of the hierarchical and relational selection algorithms which have been shown to augment an agent's ontology with domain related concepts. Specifically, our approach will perform better in terms of the number of concepts in the ontology than the *learn-fragment* approach, because we reduce the number of concepts that an agent can choose to augment. This entails that our approach will evolve the ontology with the lowest estimated complexity (see Section 3.4). We measure the success of our hypothesis by comparing the number of tasks that are attempted and completed, and the size of an agent's ontology. The greater the number of task attempted and the smaller the ontology is the more domain focused the ontology is;

Hypothesis 3: Agents using our proactive learning algorithm will spend the least total time acquiring and learning concepts because they benefit from using prediction to reduce the number of requests for the definition of concepts. This hypothesis considers the effectiveness of our approach against the benchmark approaches, by comparing the total cost of each;

Hypothesis 4: Agents using our proactive learning algorithm will perform better than using other approaches in terms of the RoboCup Rescue Score. We hypothesise that, because of the above hypotheses, the agents will be able to save more of the city from fire and rescue more civilians.

In the next section, we present the experiments used to show these hypotheses and detail their parameters.

5.4.2 Experimental Setup

We evaluate the effectiveness of our proactive learning algorithm by running three experiments, which have the same parameters as Experiments 1, 2, and 3 in Chapter 4 with the exception to the approaches that are compared. These experiments use the

proactive learning algorithm, the *learn-markov* approach, and compares its performance against the benchmark approaches *learn-fragment*, *learn-repeated*, *learn-everything*, and *no-collaboration*. In more detail, Table 5.2 contains the parameters used in our three experiments.

	Experiment 1	**Experiment 2**	**Experiment 3**
Environment ontologies	10	10	10
Number of agents	20	20	30
Number of fire brigades	10	0	10
Number of police	10	10	10
Number of ambulance	0	10	10
Number of time steps	2000	2000	2000
Map	Kobe	Kobe	Kobe
Benchmark approaches	Learn-repeated, Learn-everything, No-collaboration, Learn-fragment	Learn-repeated, Learn-everything, No-collaboration, Learn-fragment	Learn-repeated, Learn-everything, No-collaboration, Learn-fragment
Number of civilians	115	115	115
Iterations	250	250	250

TABLE 5.2: The parameters for Experiments 1, 2, and 3.

5.5 Results

In this section, we show the results from Experiment 3 where all three types of agent (fire brigade, ambulance, and police) use our proactive learning algorithm. We focus our discussion on the results from Experiment 3 because they showcase the benefits of the algorithm used because the performance of each agent is dependant on each other. For example, Experiment 2's ambulance agents' performance is limited because the spread of the fire is not controlled by the fire brigade agents, causing the deaths of more civilians. We present the result of Experiments 1 and 2 in Appendix B, and the following four sections, we analyse the results that show our four hypotheses.

5.5.1 Hypothesis 1 — Messages

The predominant factor in the number of messages an agent sends is the number of tasks it attempts to complete. For each task an agent attempts it will potentially require additional information as it encounters unforeseen variables (for example, new chemicals contained in buildings or civilians exhibiting new symptoms). The number of tasks an agent attempts is dependant on the agent's environment which is reactive to the completion of tasks or adversely the incompletion of tasks may result in more future tasks. For example, a fire brigade may extinguish a fire, preventing the fire from spreading, while another agent was unable to extinguish the fire and thus it spreads

creating more extinguishing tasks. In particular, our proactively learning algorithm (*learn-Markov*), attempts the lowest number of tasks per simulation (see Figure 5.15). It attempts on average approximately 400 tasks per simulation, whereas the *learn-fragment* approach averages approximately 600 tasks per simulation.

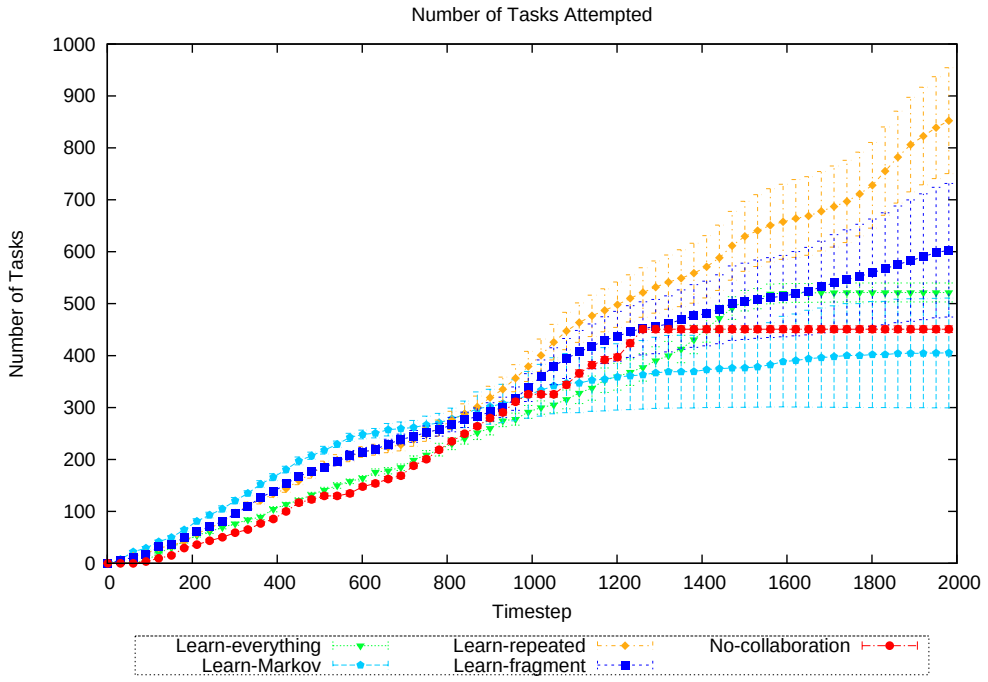


FIGURE 5.15: A graph showing the average number of tasks attempted per agent.

The *learn-Markov* approach sends the fewest number of messages because it was able to predict which concepts had the highest likelihood of being used for future tasks, and thus was able to prioritise which concepts to incorporate into an agent's ontology, reducing the number of messages. Despite the *learn-Markov* approach attempting approximately two thirds of the number of tasks than the next best approach (*learn-fragment*), it sends 49% fewer messages.

In particular, Table 5.3 shows that the *learn-Markov* approach sends 49% fewer messages overall than the *learn-fragment* approach (as noted above). We can also see that the maximum average number of messages sent is also lower, with the *learn-Markov* approach sending 44.1 maximum messages per timestep, compared with the *learn-fragment* approach's 100.11.

In summary, we have confirmed our hypothesis because our approach sends the fewest number of messages and outperforms our reactive learning algorithm, the *learn-fragment* approach and the *learn-repeated* approach which has perfect foresight.

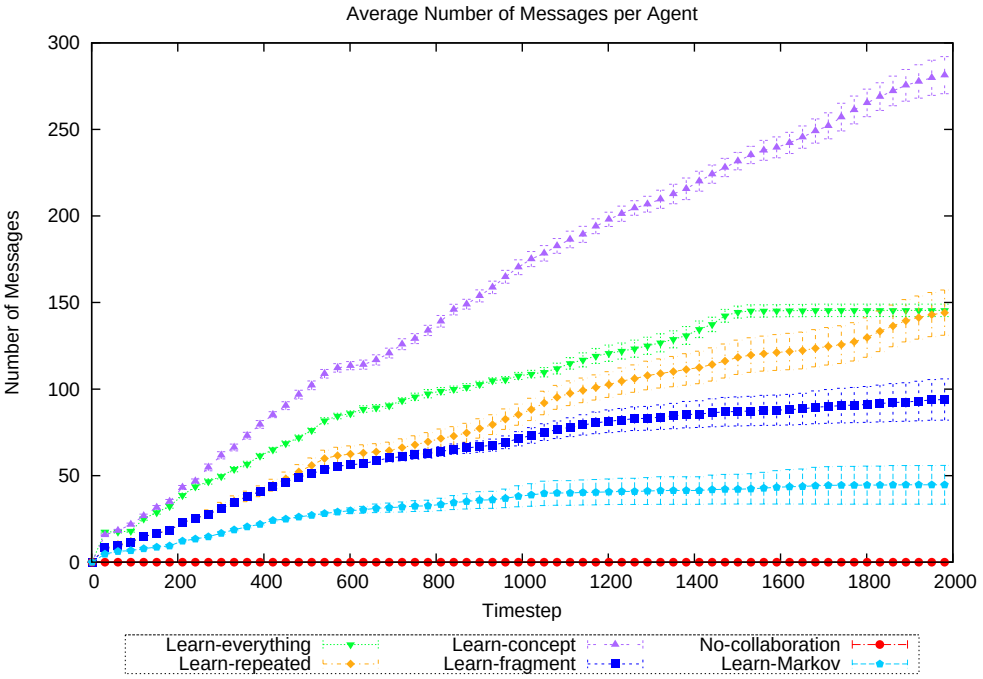


FIGURE 5.16: A graph showing the average number of messages sent by per agent.

Technique	Average number of timesteps with no messages	Average max per timestep	Average value per timestep	Average final value
Learn-everything	2.18	145.52	101.58	145.52
Learn-repeated	2.15	144.70	82.65	144.70
Learn-concept	2.12	281.91	161.38	281.91
Learn-fragment	2.12	94	65.18	94
No-collaboration	2000			
Learn-Markov	2.34	44.71	33	44.71

TABLE 5.3: Statistics for the number of messages sent per agent.

5.5.2 Hypothesis 2 — Ontology Size

In order to show that our proactive learning algorithm is able to evolve ontologies so that they are domain focused and have the lowest ontology complexity, we compare the number of concepts contained within an ontology (shown in Figure 5.17) against the number of tasks it successfully completed (shown in Figure 5.18).

The agents using the *learn-Markov* approach have the lowest number of concepts in their ontology and thus the smallest estimated ontology complexity (apart from the *no-collaboration* approach which learns no additional concepts). Specifically, the *learn-*

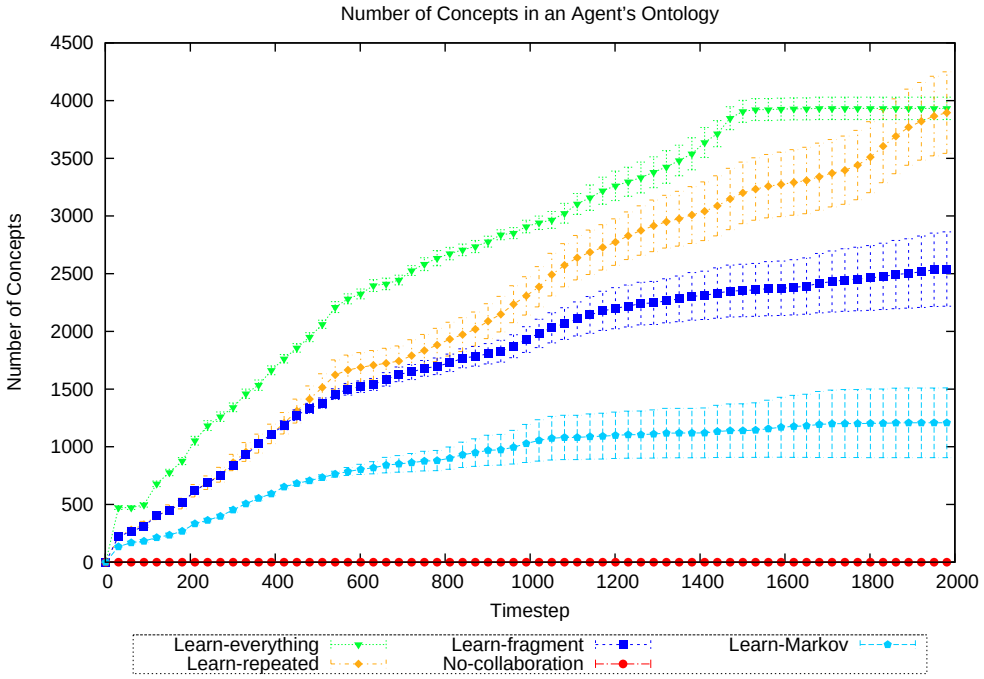


FIGURE 5.17: A graph showing the average number of concepts in each agent's ontology.

Markov approach augmented agent's ontologies with 56.8% fewer concepts than the next best approach (*learn-fragment*). Our *learn-Markov* approach has the highest average number of successful tasks completed, at 125.75, with *learn-fragment*, the next highest, completing 89.47. Thus, the *learn-Markov* technique learns fewer concepts and has the lowest ontology complexity, and is still able to successfully complete 40.5% more tasks than the next best approach, *learn-fragment*.

Technique	**Average final value**	**Average minimum value**	**Average maximum value**
Learn-Markov	1208.25	697	2413
Learn-everything	2588.03	2237	3640
Learn-repeated	3135.86	545	4106
Learn-fragment	1894.83	1342	2600
No-collaboration	893.07	836	930

TABLE 5.4: Statistics for the number of concepts in each agent's ontology.

In summary, we have confirmed our hypothesis because agents that use our proactive approach augment the fewest concepts, and thus have the lowest ontology complexity of all of the compared approaches (with the exception of the *no-collaboration* approach). It outperforms our other reactive approach (*learn-fragment*) and the *learn-repeated* ap-

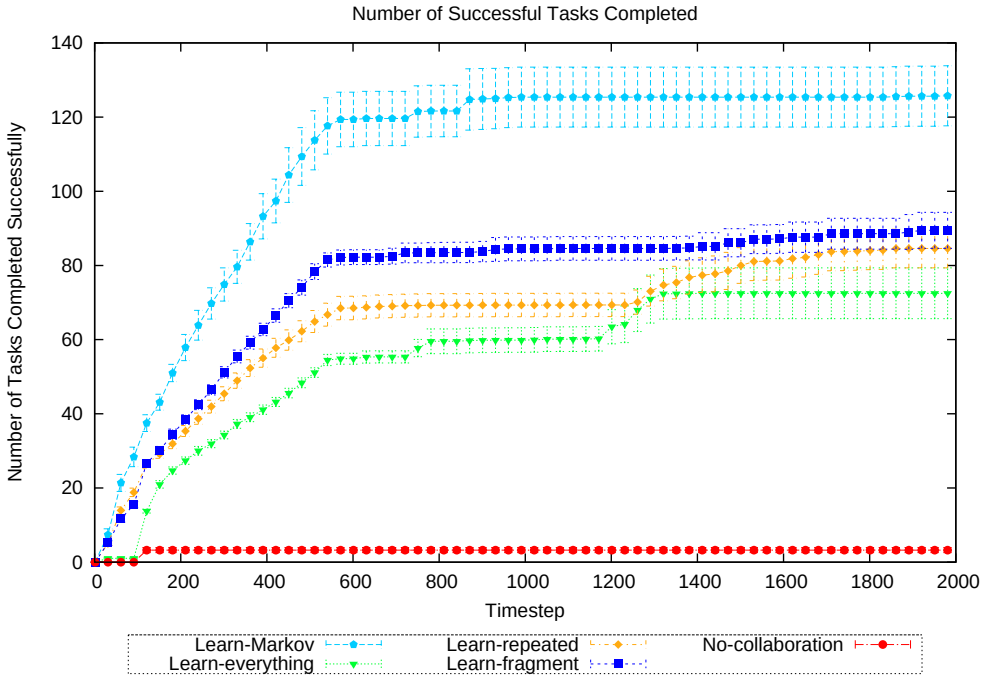


FIGURE 5.18: A graph showing the average number of successful tasks completed by each agent.

Technique	Average number of successful tasks completed per timestep	Average final number of successful tasks completed
Learn-Markov	0.062875	125.75
Learn-everything	0.036	72.5
Learn-repeated	0.042	84.53
Learn-fragment	0.045	89.47
No-collaboration	0.0016	3.26

TABLE 5.5: Statistics for the average number of successful tasks an agent performs.

proach which has perfect foresight.

5.5.3 Hypothesis 3 — Learning, Deliberating and Acting

In Section 4, we use three types of pixel plots to provide a visual representation of how agents spend their time. However, in order to visualise the amount of time that the *learn-Markov* approach spends on its actions, we also include a pixel plot that shows how much time agents spends using their Markov models. Similar to the pixel plots in Section 4.4.3, we use black pixels to indicate the time the agent spents using its Markov model.

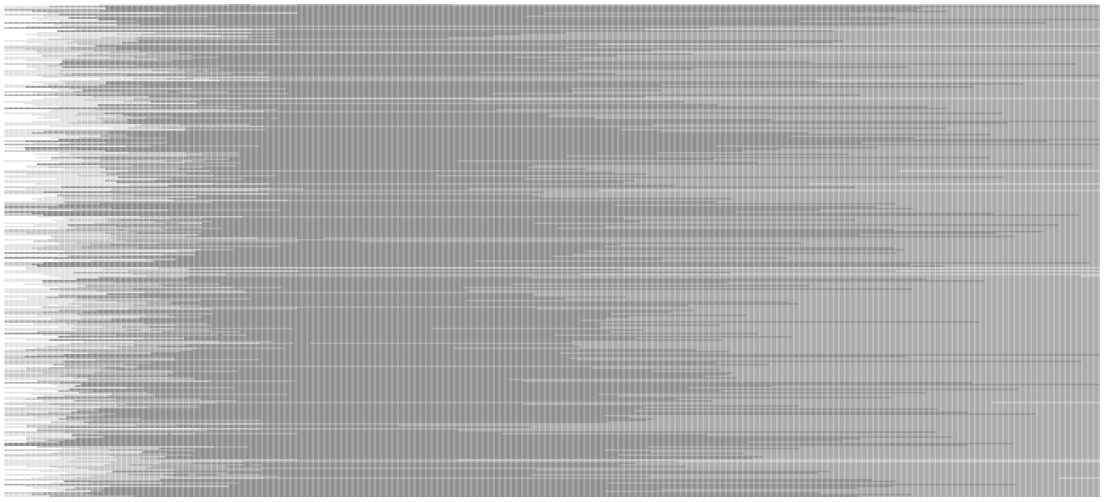
This information can show whether the agent using the proactive learning algorithm is hindered time wise by using its Markov model. In this section, we only compare the *learn-Markov* approach to the *learn-fragment* approach, because the *learn-fragment* approach performed the best from the other benchmark approaches and are analysed in detail in Section 4.4.3. We now proceed to compare the *learn-Markov* and *learn-fragment* approaches.

Agents using the *learn-Markov* approach spend most of their time acting (see Figure 5.20(c)), then learning (see Figure 5.20(a)), then deliberating (see Figure 5.20(b)), and then using their Markov model (see Figure 5.20(d)). Both the *learn-Markov* and *learn-fragment* approaches allow agents to spend most of their time acting (see Figures 5.20(c) and 5.19(c)), however the *learn-Markov* approach spends more time acting than the *learn-fragment* approach. Agents using the *learn-Markov* approach also spend less time learning, and spend slightly more time deliberating than agents using the *learn-fragment* approach. They can spend more time deliberating because they spend less time learning, and therefore they will be able decide on the actions to take quicker than those using the *learn-fragment* approach. Thus making the acting more successful because fires are less likely to spread and civilian symptoms are treated promptly. In more detail, the agents spend less time learning and deliberating in the first 200 timesteps, because in this time the agents have fewer concepts in their ontology (see Figure 5.17). The Markov model is predominately used in the beginning of the scenario, and after 1000 timesteps the agents learn significantly fewer concepts than in the first 1000 timesteps. This is because the Markov model was able to predict which concepts to prioritise learning and therefore reduces the need to learn concepts in the second half of the scenario.

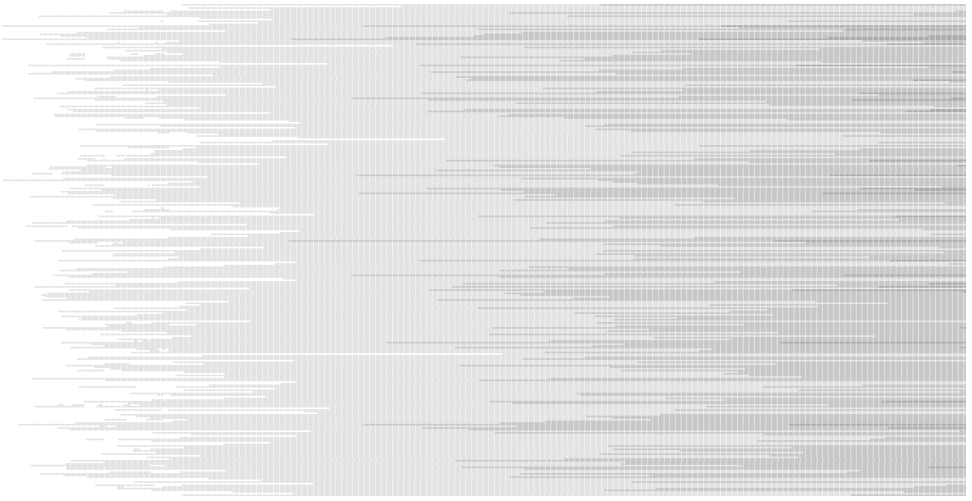
In summary, we have confirmed our hypothesis because the agents using our proactive learning algorithm spend less time learning because they predict which concepts have a high likelihood of use and learn them early in the simulation. Therefore, they can spend more time acting and deliberating about their actions. This enables agents using the *learn-Markov* approach to rescue their targets earlier in the simulation compared to the other approaches.

5.5.4 Hypothesis 4 — RoboCup Rescue Results

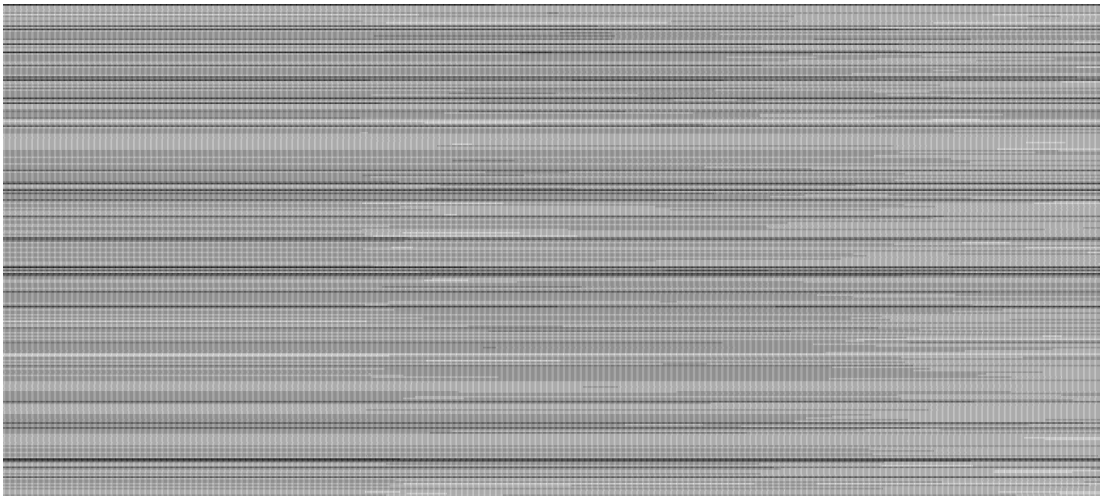
In terms of the RoboCup Rescue framework, our proactive learning approach (*learn-Markov*) outperforms our reactive learning approach, *learn-fragment*. The *learn-Markov* approach saves 21% more civilians and 59% more of the city compared with any other approach. Our approach manages to get the fires under control earlier than the other approaches, where the average amount of city saved begins to plateau at approximately 1000 timesteps, whereas the other approaches reduce the rate of the fire spreading through the city (shown as a reduced gradient on Figure 5.21). The *learn-Markov* approach was able to control the spread of the fire, and save more civilians (see Figure 5.22)



(a) Pixel-plot showing the time spent learning for the *learn-fragment* approach.

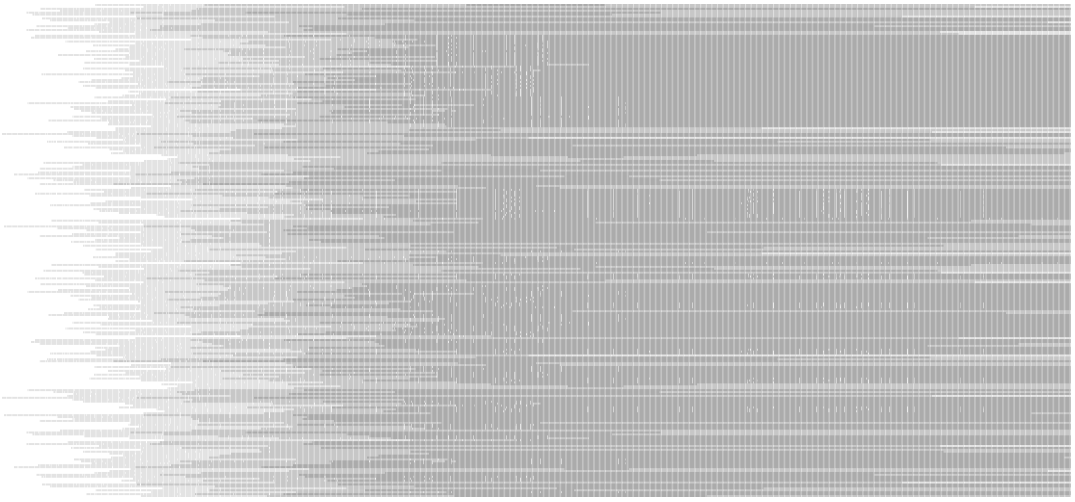


(b) Pixel-plot showing the time spent deliberating for the *learn-fragment* approach.

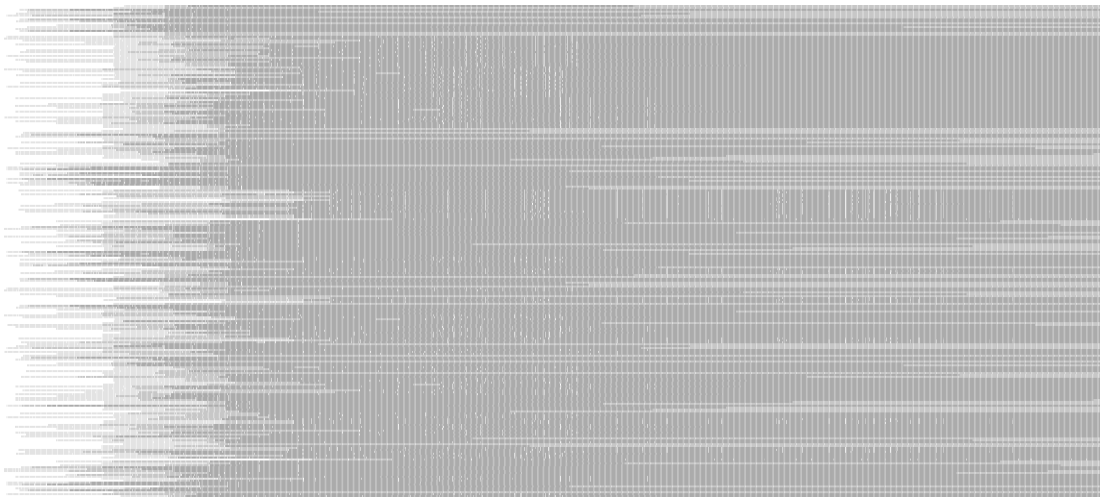


(c) Pixel-plot showing the time spent acting for the *learn-fragment approach.*

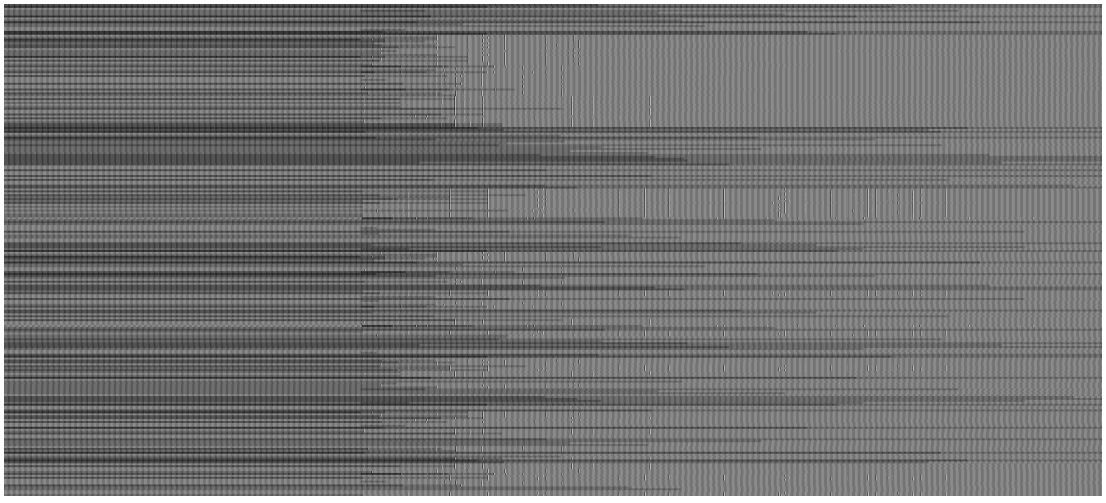
FIGURE 5.19: Pixel plots for the *learn-fragment* approach.



(a) Pixel-plot showing the time spent learning for the *learn-Markov approach.*

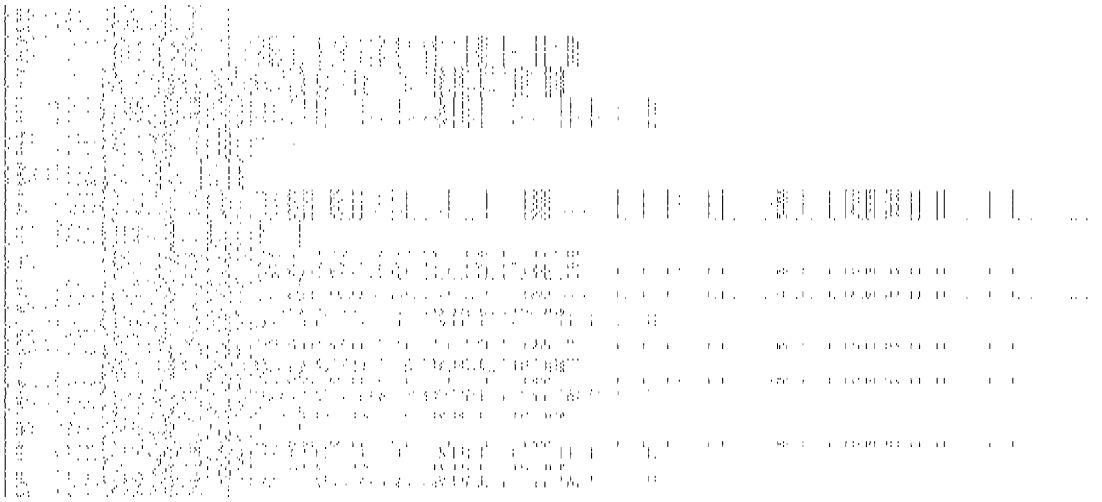


(b) Pixel-plot showing the time spent deliberating for the *learn-Markov* approach.



(c) Pixel-plot showing the time spent acting for the *learn-Markov* approach.

FIGURE 5.20: Pixel plots for the *learn-Markov* approach.



(d) Pixel-plot showing the time spent using the Markov model for the *learn-Markov* approach.

FIGURE 5.20: Pixel plots for the *learn-Markov* approach.

because it was able to successfully complete tasks in a shorter amount of time because it could do so with learning fewer concepts. The agents using the *learn-Markov* approach were able to save 78% of the civilians because the fire agents control the spread of the fire, and the ambulance agents spend less time learning and deliberating, and more time rescuing, compared to the other approaches.

In summary, we have confirmed our hypothesis because our proactive approach, *learn-Markov*, enables agents to complete 40.5% more tasks successfully than the other approaches, while learning 56.8% fewer concepts. It uses a Markov model to proactively learn additional concepts, and thus learns more efficiently.

5.6 Summary

We have presented and evaluated our proactive online learning algorithm which is used to augment an agent's ontology based on information that has been required in the past, given the state of the current task. This algorithm is proactive because it uses prediction to determine concepts that have a high likelihood of use, and selects those concepts to augment into an agent's ontology. In order to predict which concepts have a high likelihood of use, we use a Markov model that represents the order in which concepts have been acquired in the past. In our evaluation, we compared our proactive learning approach in the setting of RoboCup Rescue. Our evaluation shows that our proactive learning approach:

1. Sends fewer messages than our reactive learning approach, because it was able to predict which concepts would be useful for future tasks. The agents using this approach: request more than one concept at a time; prioritise augmenting

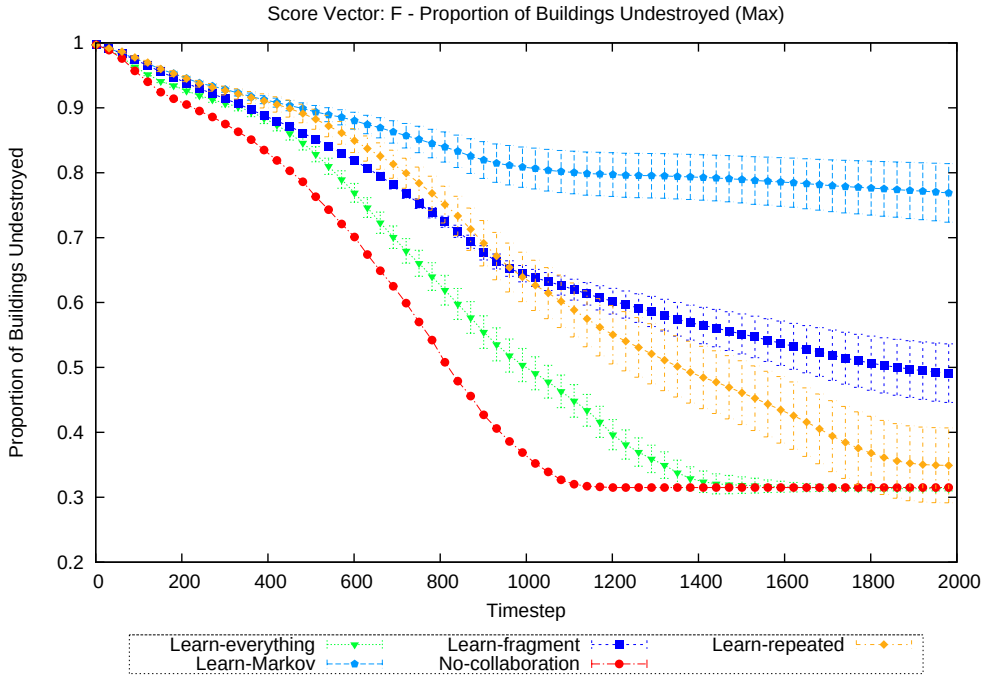


FIGURE 5.21: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

an ontology with concepts that have a high likelihood of use; and do not learn concepts that have been used in a simulation (see Section 5.5.1);

2. Incorporates fewer concepts than our reactive approach, and successfully completes the highest number of tasks with the smallest ontology compared with the other learning approaches. Thus, this approach creates a more domain focused ontology than any other approach without perfect foresight (see Section 5.5.2), while having the lowest estimated complexity;

3. Spends the least total time acquiring and learning concepts, because it can predict which concepts have a high likelihood of use for future tasks (see Section 5.5.3);

4. Outperforms our reactive learning approach, in terms of the RoboCup Rescue score vector, because it saves on average more civilians and extinguishes more burning buildings. This is because it sends fewer messages, maintains a smaller ontology, and thus spends the least amount of time acquiring and learning concepts compared to our reactive learning algorithm (see Section 5.5.4).

While we expected that the *learn-Markov* approach would outperform our *learn-fragment* approach because it could predict, we did not expect that it would spend more time

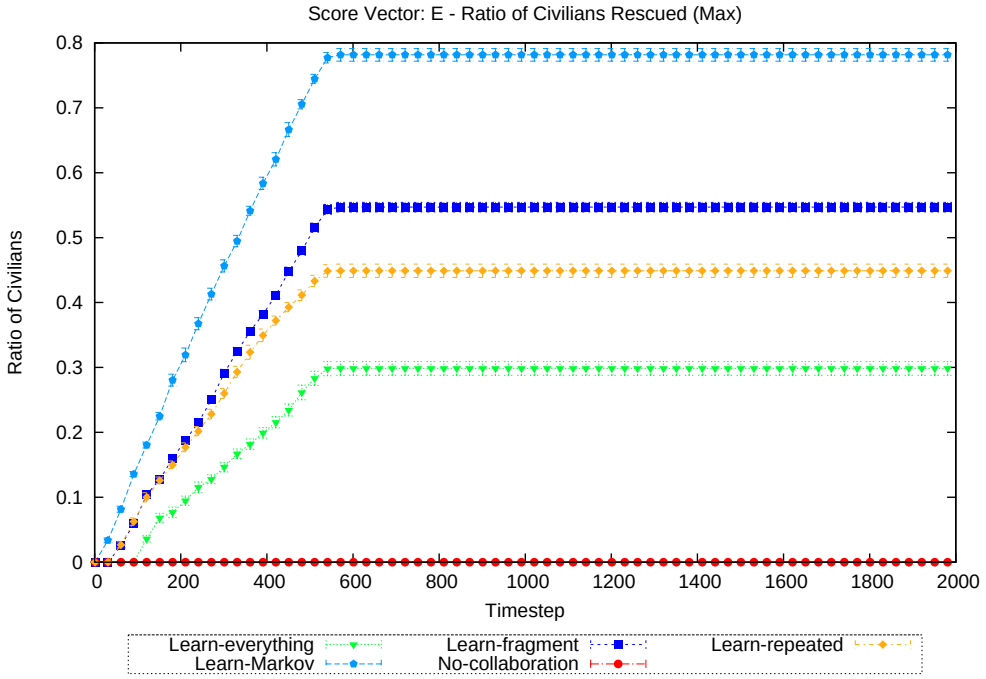


FIGURE 5.22: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

deliberating about its actions. We did however anticipate that our approach would spend less time learning and more time acting. The *learn-Markov* approach was able to complete more tasks successfully, as shown in Figure 5.18 because it could learn the concepts needed for those tasks ahead of time, using its prediction model. Specifically, by completing tasks successfully, it spent less time travelling from task to task, which can be seen by the lower number of attempted tasks, in Figure 5.15. Less time travelling equates to more time deliberating, since travelling is a time when deliberation is not performed; instead the agent is waiting to see if it has reached its destination. Thus, by being able to attempt tasks and successfully complete them, rather than being forced to abandon them, the agent spends less time travelling and instead spends more time on actions and more time deliberating about them.

In this chapter, we have satisfied Requirement 2b from Section 1.3.2 by developing an algorithm that incorporates new domain-focused knowledge into the agent's ontology, which uses a methodology designed to reduce the overhead resource costs of learning regularly required concepts through using prediction. We have also satisfied Requirement 3 from Section 1.3.3 through performing our evaluation where we have assessed the performance of our second learning algorithm using the following measures: the time an agent spends using its Markov model; the time spent learning; the number of messages

sent by each agent; the number of successful tasks completed; how domain focused the agent's ontology is; and their performance in regards to the RoboCup Rescue score vector.

In this chapter and the previous, we consider the relationships between the time spent learning, deliberating and acting and found that the larger the ontology the more time the agent spent learning. Therefore in the next chapter, we present an approach that removes concepts from an agent's ontology so that the agent can benefit from spending less time deliberating and more time acting. For future work, we intend to extend our online learning algorithm so that it can effectively evolve ontologies in an open environment (see Section 7.3). In an open environment our approach would need to consider:

1. Scale issues, because it would be possible to augment the ontology with more fragments than the RoboCup OWLRescue framework and thus the number of concepts in the ontology could rapidly increase over a short timeframe. Without altering our learning algorithm it would be possible that our approach would be only as effective as the *learn-everything* approach;
2. Trust issues, because our approach does not consider who is best to learn from;
3. Consistency issues, because our framework does not contain any contradictory axioms about concepts.

Chapter 6

A Forgetting Algorithm for Evolving Ontologies

This chapter describes our online forgetting algorithm which prunes concepts which are deemed not to be useful for future tasks. By so doing, the algorithm reduces the size of an agent's ontology and so improves the efficiency of its use of the ontology. Specifically, this algorithm enables an agent to evaluate, select and remove irrelevant concepts from its ontology. In this chapter, we also evaluate the algorithm against other state-of-the-art approaches to evaluate the effectiveness of our approach. Thus, this chapter fulfils Requirements 2c and 3c which require the development of a technique that removes concepts that are determined to be not useful for future tasks and evaluate this approach.

To this end, the next section provides the motivation for forgetting concepts from an ontology. Then, in the following section, we detail our forgetting algorithm. Following that, we evaluate our algorithm against other benchmark algorithms and then in the final section we conclude. A glossary of the terms used and those introduced in this chapter is given in Appendix F.

6.1 The Motivation for Forgetting

In order to motivate the need for forgetting concepts from an ontology, we first consider the costs associated with a large ontology (as introduced in Section 1.1.2). Specifically, when using an ontology, an agent incurs costs with hosting, maintaining, and using it (see Section 2.2.2); the larger the ontology, the greater the need for physical memory and time to access the required information. Therefore, an agent can benefit from reducing its vocabulary, and we have identified three types of situation when agents would benefit from forgetting:

1. Performance: An agent determines that the slow response times from enacting operations on its ontology is hindering its performance. This may happen because of a degradation in performance, caused by an agent augmenting:

   (a) A large amount of information into its ontology. Performance degrades because more memory and CPU resources are required to load the ontology and thus increases the time it takes to use it, as it increases in size. In our case, there are specific time constraints on RoboCup Rescue agents, where the time it takes for an agent to query its ontology is too long for it to make decisions within a single RCR timestep. Therefore the agent is motivated to forget concepts in order to reduce the time taken to use its ontology, so that it can take actions within the timestep. For example, a RoboCup Rescue agent determines that it spent too long in the last timestep deliberating about its action, and thus not performed it within that timestep. Thus the agent is motivated to improve performance by forgetting concepts from its ontology, and thus spend less time working out its next action;

   (b) Concepts which increase the complexity of the agent's ontology. These include: concepts which use constructs with a high time and space reasoning complexity (see Section 2.1.3); and fragments that contain many relationships, thus increasing the edge-connectivity (with relationships) between the concepts in the agent's ontology. Therefore, an agent can augment its ontology with a fragment that can increase the overhead of querying its ontology disproportionately for its size. The agent can reduce the overhead by removing concepts, which reduces the degree of connectivity between the concepts in the ontology, and the likelihood of the ontology containing high complexity constructs.

2. Specialisation: An agent can specialise in a particular domain, thus any concepts unrelated to this domain can be removed from its ontology. This may happen because of:

   (a) Specialisation before the scenario, where an agent's strategy and or its capabilities are predetermined. An agent may augment its ontology with information that it cannot use because it was unable to determine if the information would be useful when augmenting its ontology. This may be because the agent does not possess the skills required to utilise the information. For example, a fire agent's ontology contains information about first aid kits and their equipment, and during its lifetime it discovers additional information about other medical equipment such as intubation equipment, which can only be used by trained professionals. Thus, in order to reduce the resources required to load and query the ontology, an agent can remove this information from its ontology because it does not improve their response to extinguishing fires;

(b) Specialisation after the scenario, where an agent's strategy may change during its lifetime. An agent might choose to specialise because it provides it with the greatest utility, and thus it desires that its ontology only contains concepts useful to complete tasks in this speciality. For example, in the RoboCup OWLRescue scenario, the control centre for the fire agents calculates that the agents are more efficient when they do not change their vehicle, therefore the centre only allocates the fire agents' targets based on their vehicles' capabilities. Due to the change of strategy, the fire agents decide to remove concepts that are unrelated to their vehicles. The agent still desires to learn about new chemicals because they have different evacuation, protective equipment, and disposal recommendations.

3. Relevance: An agent may contain information about concepts that become out of date or are irrelevant to the agent.

(a) An agent may contain out of date information about the equipment used in specific scenarios. In this case, an agent may choose to either: forget irrelevant concepts because it reduces the storage costs, this becomes more important when a domain changes frequently; or deprecate the concept in the ontology because this provides the agent with more semantics. For example, a vehicle may be deprecated or replaced by a better model, thus it is advantageous to remove the information about that vehicle. Also, a new treatment may be found to cause an adverse reaction in patients so an alternative treatment is always recommended, thus the agent will not require the information about the new treatment and can remove it from its ontology. For the RoboCup Rescue scenario, this situation is unlikely to occur because of the short duration of the simulation, and the knowledge in the scenario is static;

(b) An agent may possess information about concepts that are irrelevant to the agent. In our example, the HazChem specification contains chemicals that do not require any special handling or equipment in certain scenarios. While retaining knowledge about chemicals that are un-reactive is useful to an agent, information about the chemical's properties is not. Therefore, if a fire agent has incorporated an un-reactive non-oxidising chemical into its ontology, then it can remove properties about this chemical without affecting the outcome of its performance.

Given the above motivations, we proceed to present our forgetting approach.

6.2 The Forgetting Algorithm

When an agent decides to contract its ontology, the agent must first evaluate the concepts in its ontology to select which concept to remove, second select a fragment of the concept that is deemed to be the most irrelevant, and third remove the concept so that the ontology remains consistent (this process is depicted in Figure 6.1). In our RoboCup Rescue example, the agents use a capacity on the number of concepts in their EO to decide whether to forget, if the number of concepts in an agent's EO reaches the capacity, then it forgets. The capacity of the agent's EO is determined by the average number of concepts in their ontology when they are unable to complete an action in a single timestep. However, other reasons for triggering the forgetting action could include: a higher level of specialisation, for example a fire brigade could specialise in only extinguishing fires in industrial areas thus removing the need to know about chemicals that are exclusive to the commercial and residential areas (see Section 3.2.1); and, removing irrelevant concepts, for example, a fire brigade agent will not require to know how to handle chemicals that are not flammable and therefore can remove these properties about these concepts.

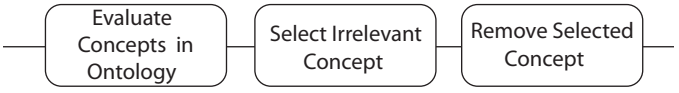


FIGURE 6.1: The forgetting process

In our scenario, we only consider removing concepts that have been learned through participating in tasks because we do not want to change an agent's core knowledge (see Section 3.1). For example, a fire brigade agent will require a different core set of concepts than an ambulance team because they are specialised. Thus, we only remove concepts from an agent's EO. The following three sections describe how we enable an agent to automatically evaluate the concepts within its ontology, select a fragment to prune from the ontology, and remove the fragment.

In order to situate our forgetting algorithm, we use a running example: a fire brigade agent's ontology is hindering its actions because it is taking too long to decide which action to take. The fire brigade is therefore motivated by performance to forget concepts from its ontology (as described in Section 6.1). The fire brigade agent has augmented its ontology with information about fire vehicles so that it can co-ordinate and operate such vehicles during a rescue. The augmented information details functional as well as aesthetic details about vehicles, which includes information about a vehicle's body colour. The colour of the vehicle is not important to the agent because it does not improve the success of extinguishing fires. We represent the agent's ontology using a graph (see Figure 6.2), where the nodes represent concepts, the lines represent subsumption relationships, and the dashed lines represent class restrictions and relationships between concepts.

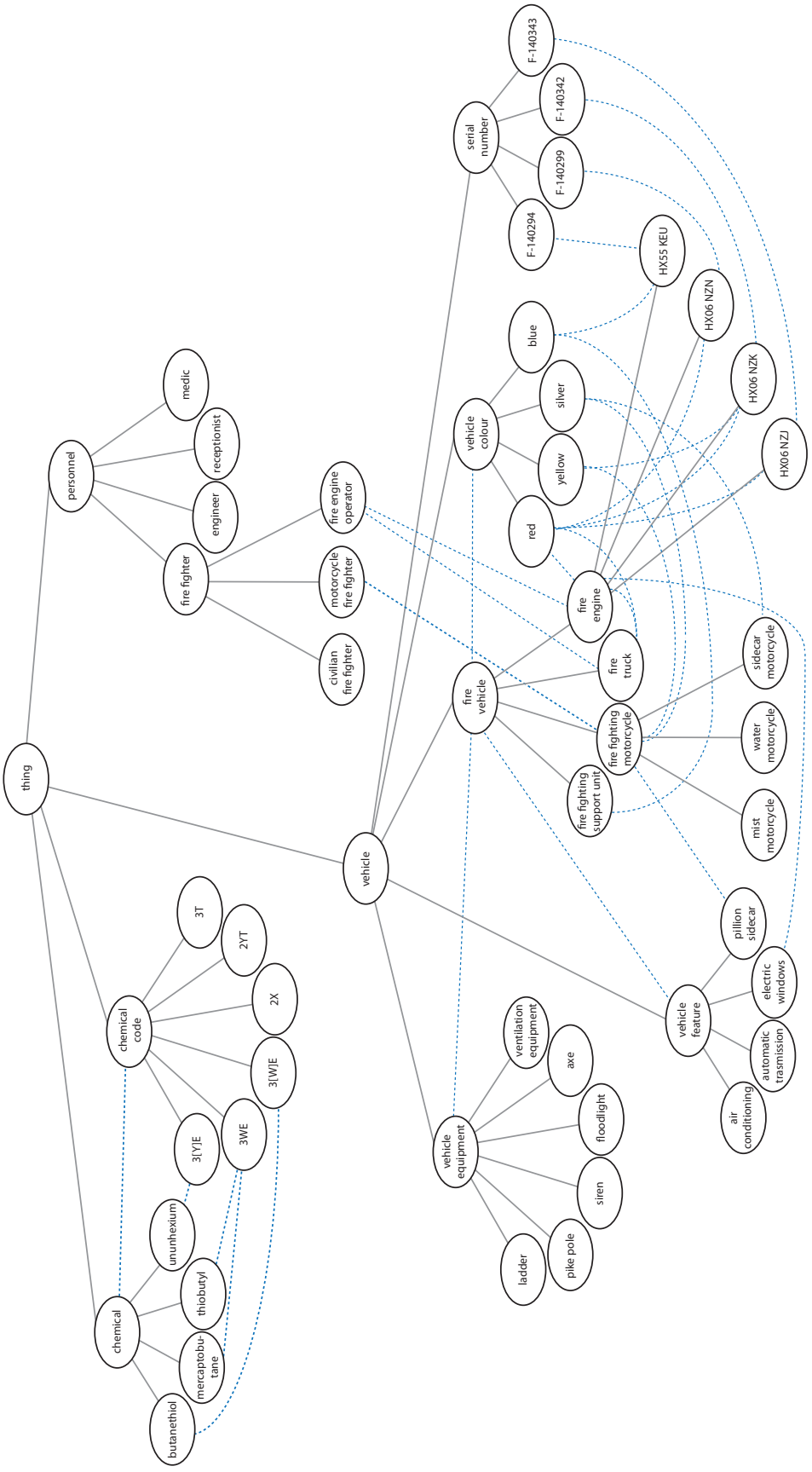


FIGURE 6.2: A fire agent's ontology, comprised primarily of concepts that describe vehicles, chemicals and personnel.

6.2.1 Evaluate Concepts

Once a task agent determines that it needs to contract its ontology, it decides which concepts it wants to remove. Our approach enables an agent to evaluate the concepts in its EO using two influential factors:

1. How recently and frequently the concept is used to answer queries: This approach aims to reduce the cost of acquiring regularly required concepts so we therefore adopt the Least Recently, Frequently Used (LRFU) (Lee et al., 1997) approach that is used in caching scenarios to select concepts to remove from a task agent's ontology. We order the concepts by their LRFU value and in our algorithm we use this ranking to evaluate which concept to forget.

   Since LRFU takes into account concepts that have been used frequently and recently, the LRFU value effectively measures how "useful" a concept has been to the agent. The use of LRFU therefore meets Requirement 2c (see Section 1.3.2) which requires that forgetting algorithms must remove concepts that are the least useful;

2. The cost of the original acquisition of the concept: We record the cost of acquiring each concept so that our approach can avoid forgetting concepts that are expensive to re-acquire. This cost is recorded in milliseconds and is recorded by our learning approach, in order to be used here. We order the concepts by their acquisition cost, and in our algorithm we use this ranking to evaluate which concept to forget.

   Concepts require different amounts of resources in order to be learned, because each concept's fragment is different. Therefore by taking this into account, we can avoid forgetting concepts that will cost the most to re-learn. Thus, the use of the acquisition cost enables our algorithm to meet Requirement 2c (see Section 1.3.2) which requires that forgetting algorithms must optimise the amount of resources they use.

We use the sum of these two factors to calculate a concept forgetting value (CFV), to provide a measure of how useful a concept is to an agent, and how expensive it will be if it has to be relearned after being forgotten. We calculate the CFV by summing LRFU and AC in this way because the CFV then embodies both notions stated by our requirements. The equation used to calculate the CFV is shown below, which we use to rank candidate concepts to forget:

$$CFV = LRFU + AC \tag{6.1}$$

where CFV is the concept forgetting value of a concept in an agent's EO, the $LRFU$ is a ranking of the LRFU value for concept c, and AC a ranking of the the acquisition

value of concept c. A low CFV weighting indicates that the concept has not been used recently, often, and was inexpensive time wise to acquire, and a high CFV weighting indicates that the concept is used recently, frequently and was expensive to acquire.

Summing the LRFU rank and the AC rank allows an agent to select which concepts to forget. Ideally an agent desires to forget concepts with the lowest AC and LRFU ranking and summing these two factors will identify the lowest LRFU and AC. If an agent's ontology does not contain a concept with a low LRFU and AC ranking, then summing the numbers will identify concepts with either:

1. a low LRFU and a medium to high AC. This means concepts that are not used frequently or recently can be forgotten even if their AC rank is high, this is because it does not benefit the agent to keep concepts that not frequently or recently used even if they were expensive to acquire.
2. a low AC and medium to high LRFU. This means concepts that are cheap to acquire can be re-acquired cheaper than other concepts, and this enables agents to prioritise keeping concepts that are more expensive to acquire but are used less frequently. We adopt this strategy because our main focus is to reduce the amount of time spent either learning or forgetting to reduce the total time acquiring and using knowledge.

The agent will not forget concepts with high or medium AC and LRFU ranks, because the summing of these values will be higher than the others discussed above.

The LRFU value is calculated for each concept in the agent's EO using Algorithm 13. This algorithm shows how an agent calculates the LRFU value for each of its ontology's concepts, where $concept(EO)$ is a function that holds the set of concepts in an agent's EO, $T = \{\langle t_1, c_u 1\rangle, \ldots, \langle t_n, c_u n\rangle\}$ is a set of tasks where each task is a tuple representing the task (t) and the set of concepts required to complete the task (c_u), all concepts in c_u are a subset of $concept(EO)$, and all concepts in the EO have a LRFU weighting which is represented using a tuple $\langle c, lrfu\rangle$. The LRFU weighting for each concept is calculated over time. After each time period, each concept's LRFU value is calculated, by increasing the value by 1 if it is used and decaying it exponentially when it is not. In our RoboCup Rescue example, a concept's LRFU weightings are calculated each timestep and are represented by tasks in Algorithm 13 because an agent has to complete a task per timestep. Depending on the scenario, it may be appropriate to weight the AC or LRFU differently. For example, if network bandwidth fluctuates, the acquisition cost is time-sensitive, and therefore it would be appropriate to weight it lower than the LRFU. In our environment available bandwidth does not change, and therefore we do not apply weightings when calculating the CFV.

Once a concept's LRFU value has decayed, the acquisition cost becomes more influential in the weighting, thus an agent can determine which concept from a set of concepts that

Algorithm 13 Algorithm showing how the LRFU value is calculated for each concept in an agent's ontology.

Require: $concepts(EO) \neq \emptyset$
Require: $T = \{\langle t_1, c_u 1\rangle, \ldots, \langle t_n, c_u n\rangle\}$ `/* T is the set of tasks, where tasks require a set of concepts to complete them. */`
Require: $c_u \subset concepts(EO)$
Require: $\forall c \in concepts(EO) = \langle c, lrfu\rangle$

```
1: /* loop through the concepts required for each task */
2: for c_u ∈ T do
3:   /* loop through each concept in the agent's ontology */
4:   for c ∈ concepts(EO) do
5:     if c ∈ c_u then
6:       /* if the concept is used in the task, increment its LRFU */
7:       c = ⟨c, lrfu + 1⟩
8:     else
9:       /* otherwise exponentially degrade it */
10:      c = ⟨c, lrfu * ln 2⟩
11:    end if
12:  end for
13: end for
```

have the same LRFU value to forget. It is more likely that concepts will have different acquisition costs due to different agents' network locations and bandwidths, than a different LRFU value because concepts decay exponentially. Performance-wise, it is better for the agent to forget concepts that are inexpensive to acquire because the cost of re-acquiring them is less, compared to concepts that are expensive to acquire.

In our fire brigade example, the agent calculates the CFV value for all of the concepts in its ontology, see Table 6.1. The concept `red` has the lowest CFV value, indicating that this concept is the least frequently and recently used concept in the agent's ontology. The `red` concept is not imperative for completing a task, and its absence from the ontology will not negatively affect the agent's behaviour. If it became imperative to complete a task (for some unforeseen reason), the agent could re-incorporate it into its ontology.

To summarise, the agent selects the concept with the lowest rating in its EO to remove. The agent aims to remove a fragment representing the selected concept, this is described in the next section.

Concept	LRFU	AC	LRFU Rank	AC Rank	CFV
Red	0.901091335	3	1	2	3
F-140299	0.935748694	6	2	4	6
F-140343	0.935748694	0	2	1	3
F-140342	0.956543109	3	4	2	6
Yellow	0.970406053	9	5	7	12

Table 6.1: Example LRFUs, ACs, rankings and CFVs.

Concept	LRFU	AC	LRFU Rank	AC Rank	CFV
Serial Number	0.984268996	6	7	4	11
Silver	0.977337525	11	6	8	14
F-140294	1.039720771	16	9	11	20
Blue	1.018926355	8	8	6	14
Vehicle Colour	1.455609079	16	10	11	21
Automatic Transmission	3.812309493	13	11	9	20
Pike Pole	4.921344982	17	12	13	25
Air Conditioning	5.475862726	15	13	10	23
Floodlight	6.030380471	22	14	15	29
Ventilation Equipment	6.037311943	22	15	15	30
Axe	6.175941379	20	16	14	30
Water Motorcycle	6.238324625	24	17	17	34
Siren	6.376954061	25	18	18	36
Ladder	6.446268779	26	19	19	38
Vehicle Feature	6.446268779	27	19	20	39
Pillion Sidecar	6.792842369	33	21	22	43
Electric Windows	7.763248422	34	22	23	45
Mist Motorcycle	8.317766167	31	23	21	44
HX06NZJ	8.387080885	36	24	25	49
HX06NZN	8.872283911	35	25	24	49
Sidecar Motorcycle	9.010913347	40	26	30	56
HX06NZK	9.010913347	37	26	26	52
Vehicle Equipment	9.149542783	37	28	26	54
HX55KEU	9.28817222	37	29	26	55
2X	9.565431092	40	30	30	60
3[Y]E	9.704060528	44	31	33	64
3[W]E	9.704060528	39	31	29	60
3WE	10.05063412	48	33	37	70
2YT	12.19939038	43	34	32	66
Civilian Fire Fighter	12.19939038	46	34	35	69
Chemical Code	12.47664925	46	36	35	71
3T	12.6845934	45	37	34	71
Motorcycle Fire Fighter	13.93225833	53	38	40	78
Butanethiol	14.69472023	48	39	37	76
Ununhexium	14.69472023	49	39	39	78
Receptionist	15.17992325	55	41	43	84
Chemical	15.94238515	53	42	40	82
Thiobutyl	16.21964403	53	43	40	83
Fire Engine Operator	16.98210592	57	44	44	88
Medic	17.60593839	59	45	45	90

Table 6.1: Example LRFUs, ACs, rankings and CFVs.

Concept	LRFU	AC	LRFU Rank	AC Rank	CFV
Fire Fighting Support Unit	18.02182669	64	46	47	93
Fire Fighter	19.13086218	62	47	46	93
Mercaptobutane	19.89332408	64	48	47	95
Fire Fighting Motorcycle	19.89332408	66	48	49	97
Engineer	20.10126824	68	50	50	100
Personnel	21.4875626	70	51	53	104
Fire Engine	22.2500245	68	52	50	102
Fire Truck	23.01248639	69	53	52	105
Fire Vehicle	30.56779066	73	54	54	108

Table 6.1: Example LRFUs, ACs, rankings and CFVs.

6.2.2 Select Concepts

Once the agent has selected a concept it desires to forget, it creates a fragment representing that concept so that the agent can benefit performance-wise from a smaller ontology. We also hypothesise that an agent can benefit from removing more than one concept at a time so that it can perform forgetting less often than forgetting methods that forget fewer concepts (as proposed by other state-of-the-art approaches, see Section 2.2.2).

In order to select concepts to prune, the agent generates a fragment representing the selected concept and selects concepts with a similar CFV weighting. In order to detail how our agent selects the concepts to remove, we formally introduce the components described above. Let: l be the capacity limit at which the agent is required to prune concepts from its EO; W be the set of weightings for the concepts contained in the EO, where $W = \{w : C \in concepts(EO) \wedge w = weight(c)\ \}$ and $c_1 \ldots c_n \in concepts(EO)$; f_{o_q,c_t} be the fragment representing the concept to be forgotten, where o_q is the task agent's ontology (where $o_q = DO \cup EO$) and c_t is the concept to be forgotten; $W_{f_{o_q,c_t}} = \{W : C \in f_{o_q,c_t} \wedge w = weight(c)\ \}$ be the set of concept weightings associated with the concepts contained in f_{o_q,c_t}, where $concepts(f_{o_q,c_t}) = \{c_1, \ldots c_n\}$. Using this formal notation we describe how we select the concepts to forget in Algorithm 14.

In our fire brigade example, the concept `red` has been selected as the concept to forget. Using our approach, a fragment is then generated, based on `red`. The fragment generated is shown in Figure 6.4, which has been produced using the approach presented by Seidenberg and Rector (2006) (see Algorithm 5 in Section 3.2.4), as used by the specialist agents to disseminate fragments. The fragment is used to select concepts from,

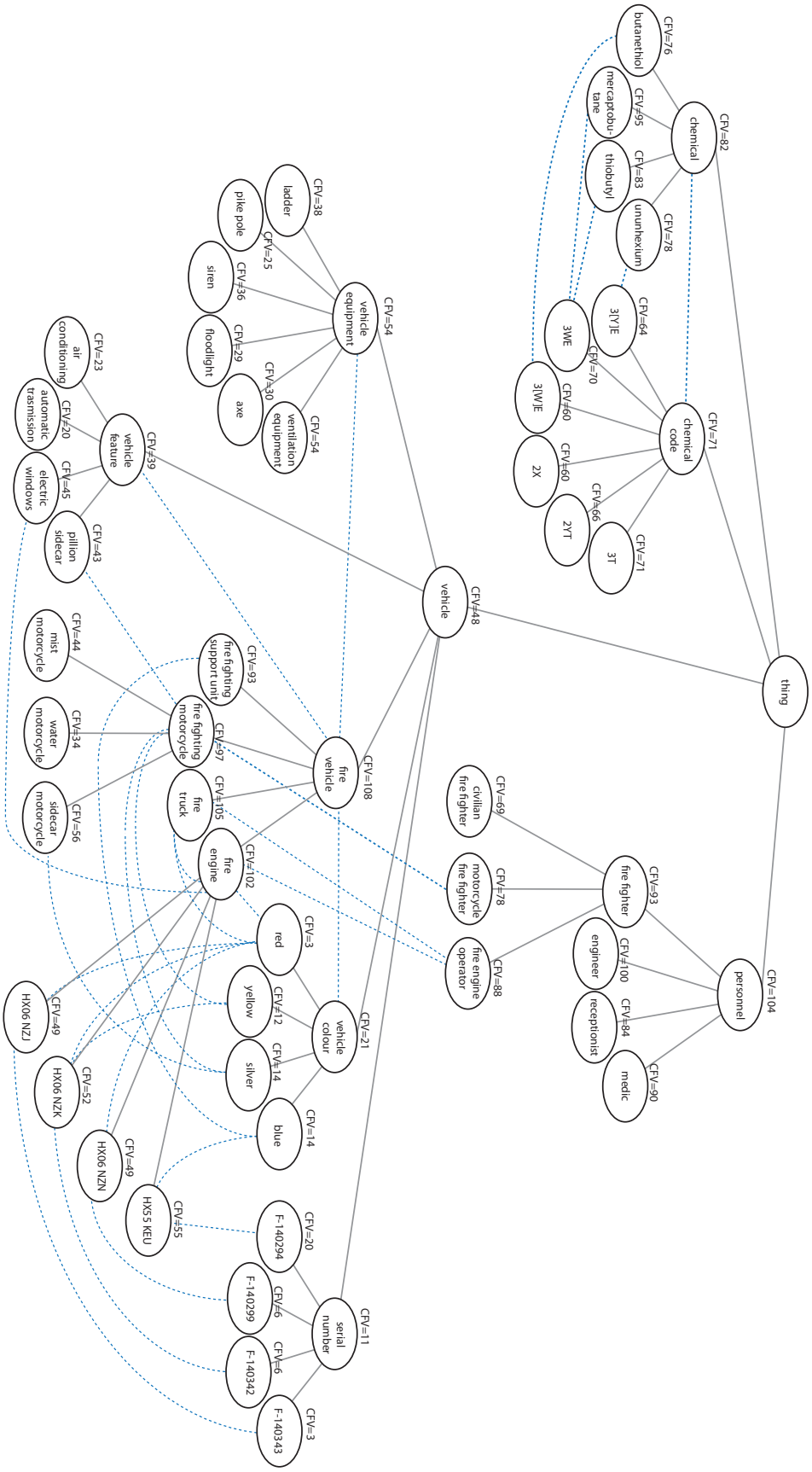


FIGURE 6.3: A fire agent's ontology, comprised primarily of vehicles, chemicals and personnel, showing the CFV weighting of each concept.

Algorithm 14 Algorithm of the Lowest Weighted Concept Selection Technique, used to select the concepts to be pruned from an agent's EO.

Function: *getLowestWeightedConcept(concepts):* `return the concept with the lowest weighting`
Function: *getWeight(concept):* `return the weight of the concept`

Require: $F_d \leftarrow$ fragment received from merging process
Require: $t \leftarrow 10$

1: `/* find the concept with the lowest concept weighting */`
2: $c_t \leftarrow getLowestWeightedConcept(concepts(EO))$
3: $w_{c_t} \leftarrow getWeight(c_t)$
4: $ConceptsToRemove \leftarrow \{c_t\}$
5: `/* loop through all concepts in the fragment */`
6: **for** $c \in concepts(F_d)$ **do**
7: `/* add to the concepts selected to be removed if this concept's weighting is within the threshold` t `*/`
8: **if** $|w_c - w_{c_t}| \leq t$ **then**
9: $ConceptsToRemove \leftarrow ConceptsToRemove \cup \{c\}$
10: **end if**
11: **end for**
12: **return** $ConceptsToRemove$

to forget. To select fragments to select, the CFV values of the concepts are compared, using the process shown in Algorithm 14. The CFV value of our selected concept `red` is $w_{c_t} = 3$, and in our experiments we use a threshold of $t = 10$. Therefore any concept in our fragment that has a CFV between 0 and 13 will be removed; these concepts are highlighted in Figure 6.5.

Once our agents have selected the concepts they desire to remove they then can remove these concepts from their ontology, this is described in the next section.

6.2.3 Remove Concepts

After the agent has selected the concepts that it desires to remove, the agent then prunes these from its ontology. In order to prune these concepts from an ontology we employ the technique proposed by Eiter et al. (2006) (see translation Tables 2.2 and 2.3 in Section 2.2.2) so that the ontology remains consistent. We illustrate three basic examples in Figure 6.6, where the concept B is removed from each example. In the first example, the axioms $A \sqsubseteq B$, $B \sqsubseteq C$ are replaced with $A \sqsubseteq C$; in the second example $A \sqsubseteq B$, $B \sqsubseteq C$ and $B \sqsubseteq D$ is replaced with $A \sqsubseteq C$ and $A \sqsubseteq D$.

In our fire brigade example, we remove the concepts `red`, `yellow`, `serial_number`, `F-140299`, `F-140342` and `F-140343`. By pruning these concepts, the agent's ontology will be as shown in Figure 6.7. Of note is that the removal of `serial_number` means that its

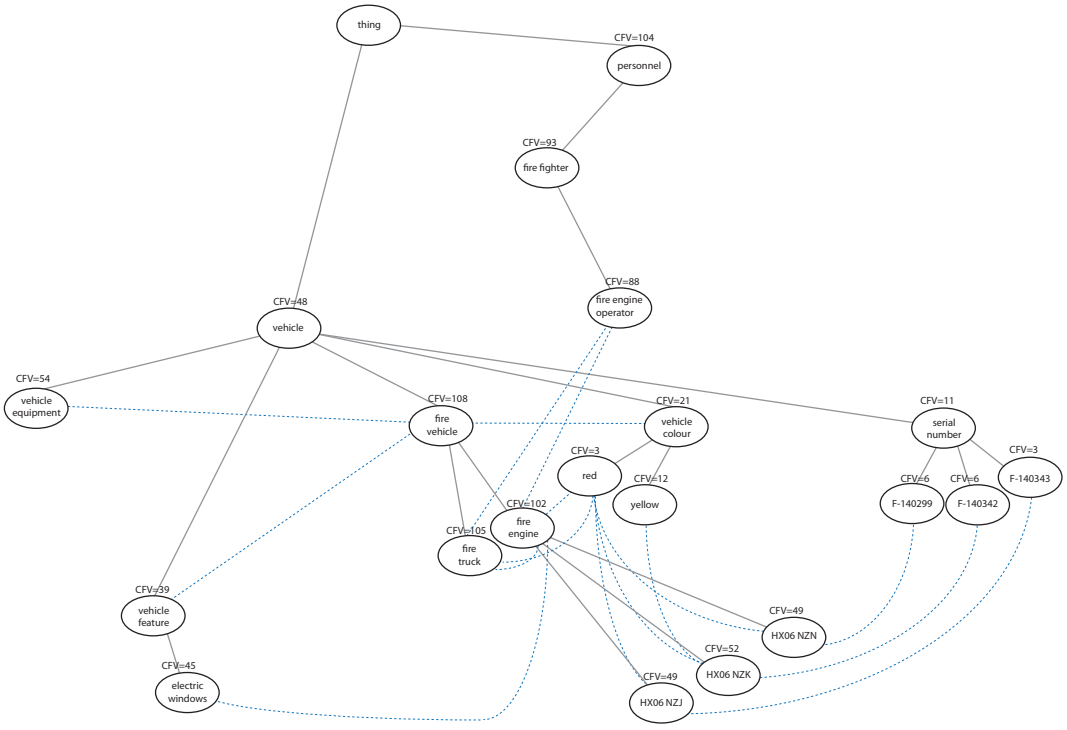


FIGURE 6.4: A fragment from a fire agent's ontology, focused on the concept `red`.

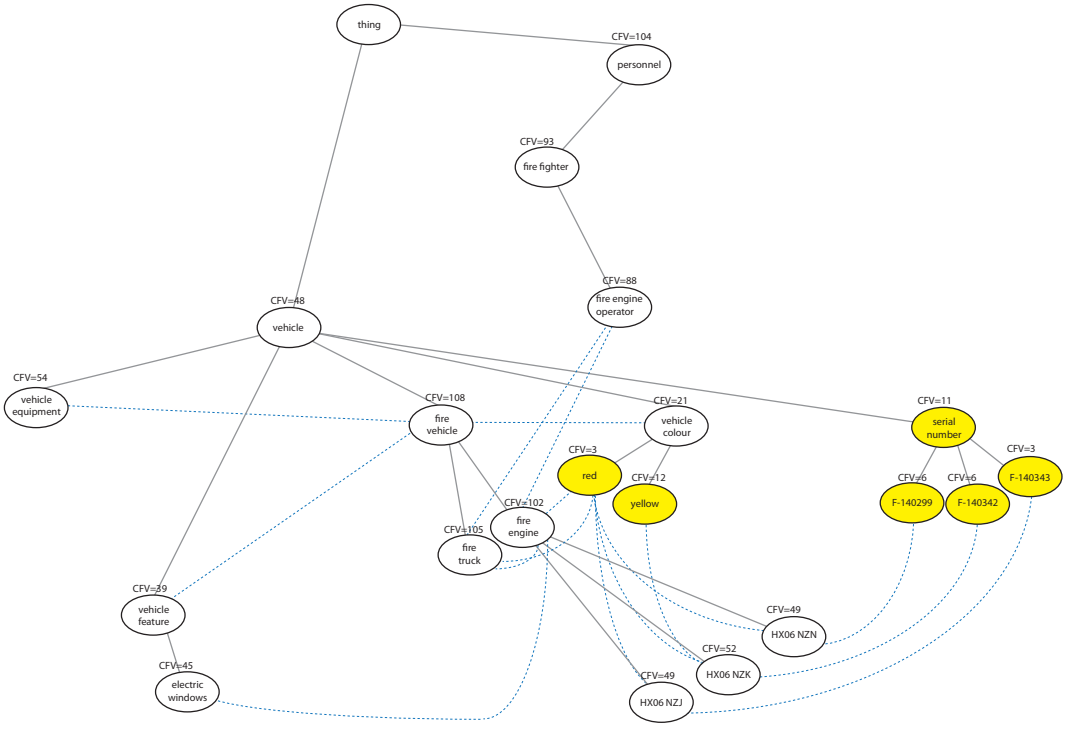


FIGURE 6.5: A fragment from a fire agent's ontology, focused on the concept `red`, with concepts to be removed highlighted.

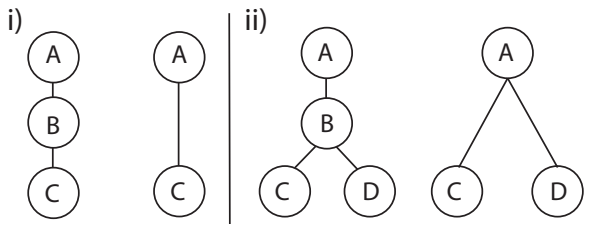


FIGURE 6.6: Three examples of removing concept B from an ontology.

child `F-140294` will be made a child of `vehicle`, using the methodology detailed above.

6.3 Empirical Evaluation

This section discusses our empirical evaluation which evaluates the performance of our online forgetting algorithm, against benchmark forgetting approaches (detailed in Section 3.3.2). To this end, we present our hypotheses and how we intend to show them in Section 6.3.1. The next section provides the specific parameters of our experiments. In the following section we present and discuss the results from the experiments.

6.3.1 Hypotheses

This section presents our hypotheses, which show how our approach addresses the contributions detailed in Section 1.4. In more detail, our forgetting algorithm is required to:

1. Demonstrate a fully automated technique that enables an agent to forget;
2. Demonstrate that pruning an agent's ontology is an effective method that allows an agent to answer continual queries in a specific domain.

In order to show that our approach addresses the contributions above and the first research objective (see Section 1.3), we aim to show the following hypotheses:

- **Hypothesis 1:** Our approach will send a similar number of messages to the other approaches, despite forgetting more concepts than the other approaches. The number of messages indicates that the agents are attempting tasks, and a similar number of messages used will show that the agent is not affected by removing concepts from its ontology. In order to show this hypothesis, we will compare the number of messages sent over the simulation, the number of tasks attempted and the number of times an agent had to relearn concepts which had been forgotten;

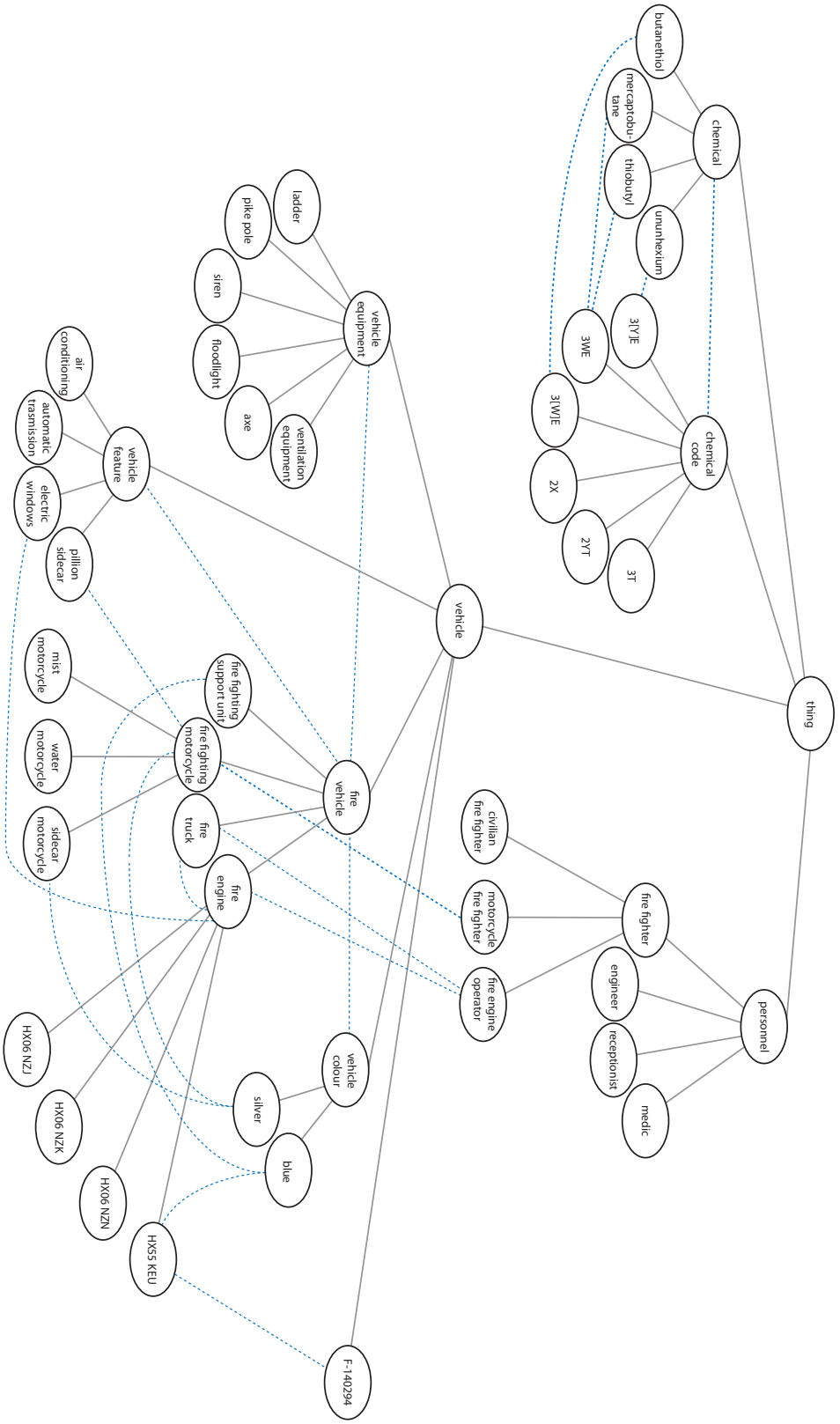


FIGURE 6.7: The agent's ontology after the selected concepts have been forgotten.

- **Hypothesis 2:** Our approach is the *best* approach for enabling an agent to regulate the size of its ontology. We define *best* in terms of the size of an agent's ontology and the number of tasks an agent successfully completes. Therefore, our approach will maintain the lowest estimated complexity (see Section 3.4) compared to the other forgetting approaches. This hypothesis shows that an approach that can complete the most tasks is the most successful in regulating the information in an ontology, because an agent benefits from limiting the information it hosts;
- **Hypothesis 3:** Our approach spends the least amount of time of forgetting, compared with the *forget-concept*, *forget-tree*, and *forget-redundant* approaches. This hypothesis shows that out of the approaches which forget a concept or a set of concepts, a connected fragment is the best structure to remove from the ontology, in terms of time. We aim to show this hypothesis by comparing the total time spent forgetting for the *forget-fragment*, *forget-concept*, *forget-tree*, and *forget-redundant* approaches;
- **Hypothesis 4:** Agents using our forgetting algorithm will perform better than using other approaches in terms of their RoboCup Rescue score vector. We hypothesise that, because of the above hypotheses, the agents will be able to save more of the city from fire and rescue more civilians.

In the next section, we present the experiments used to show these hypotheses and detail their parameters.

6.3.2 Experimental Setup

We evaluate the effectiveness of our forgetting algorithm by running three experiments, which have the same parameters as Experiments 1, 2, and 3 in Chapter 4 with the exception to the compared approaches. These experiments use the forgetting algorithm, the *forget-fragment* approach, and compares its performance against the benchmark approaches *forget-concept*, *forget-tree*, *forget-redundant*, *forget-nothing*, and *no-collaboration*.

In particular, we use the capacity of 130 concepts for the fire brigade agents' ontologies, and the capacity of 400 concepts for the ambulance agents' ontologies, as the measure of when an agent should forget. This is because the agents in our scenario were unable to complete an action in a single timestep when they had more than these respective number of concepts in their ontologies. The fire brigade agents and ambulance agents have different capacities because different amounts of concepts are learned, on average, when dealing with fires and when dealing with civilians. Specifically, ambulance agents request fragments about symptoms, treatments, medications and contraindications, whereas the fire brigade agents request fragments about chemicals, vehicles, chemical emergency action codes, chemical compounds, fire suppressants and HazChem identification numbers.

	Experiment 1	**Experiment 2**	**Experiment 3**
Environment ontologies	10	10	10
Number of agents	20	20	30
Number of fire brigades	10	0	10
Number of police	10	10	10
Number of ambulance	0	10	10
Number of time steps	2000	2000	2000
Map	Kobe	Kobe	Kobe
Benchmark approaches	Forget-concept, Forget-tree, Forget-redundant, Forget-nothing, No-collaboration	Forget-concept, Forget-tree, Forget-redundant, Forget-nothing, No-collaboration	Forget-concept, Forget-tree, Forget-redundant, Forget-nothing, No-collaboration
Number of civilians	115	115	115
EO capacity	500	500	500
Iterations	250	250	250

TABLE 6.2: The parameters for Experiments 1, 2, and 3.

The ambulance-related fragments on average provide more links between these concepts than the fire-related fragments, for example there are multiple medications per treatment, however there is only one fire suppressant per chemical. Therefore the ambulance agents encounters more concepts and thus requests more fragments per task than the fire brigade agents.

6.4 Results

In this section, we discuss the results from Experiment 3 and show how our forgetting algorithm can benefit all of the agents in the RoboCup OWLRescue scenario. The results from Experiments 1 and 2 show how the fire brigade and ambulance agents are affected by the environment in isolation. In the following four sections, we analyse the results that show our four hypotheses from Experiment 3 and present the results from Experiments 1 and 2 in Appendix C. We have presented the results from Experiments 1 and 2 in an appendix, because they show the same trend as the results from Experiment 3. However, they enable us to show that our approach is able to perform consistently for both the fire brigade and ambulance agents even though they learn from different subsets of the environment ontologies and thus has the potential to learn different numbers of concepts (see Table 3.8 in Section 3.2.4).

6.4.1 Hypothesis 1 — Messages

In order to analyse which approach requires the fewest messages, we first consider the three factors which directly affect the number of messages sent during a simulation:

- **The number of concepts an agent relearned**: The relearning of concepts increases the number of messages sent because the agent has to resend messages to reacquire a concept. It is therefore desirable to relearn the fewest number of concepts possible. We show the average number of concepts relearned by approaches which forget concepts in Figure 6.8, which is cumulative, and Table 6.3 which contains statistics about the number of concepts relearned;

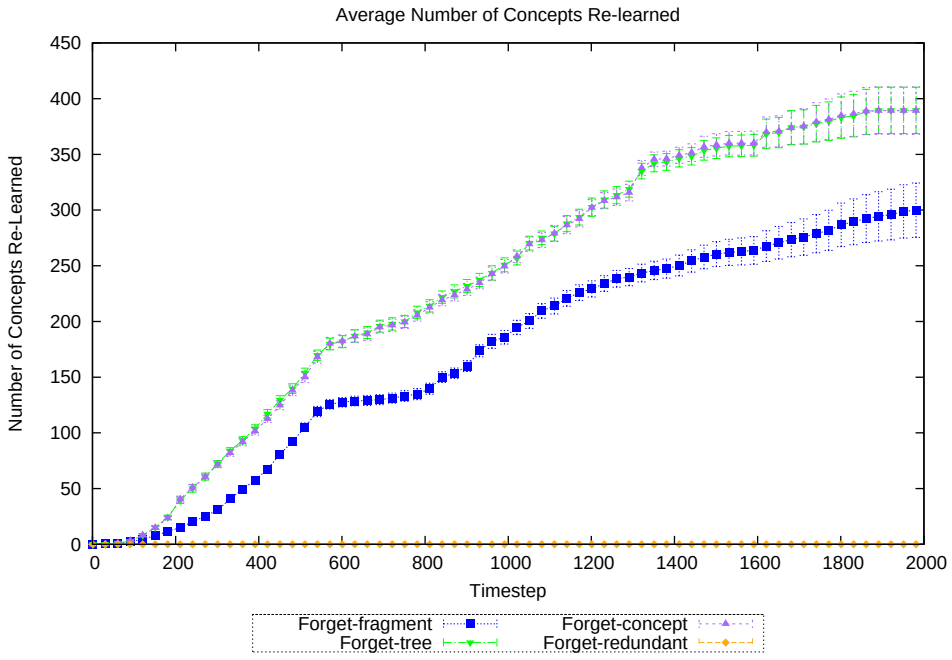


FIGURE 6.8: A graph showing the average number of times a concept is re-learned by agents using each approach.

Approach	Average number of concepts relearned per timestep	Average total number of concepts relearned
Forget-fragment	0.1506	301.29
Forget-tree	0.1948	389.5
Forget-concept	0.1946	389.21
Forget-redundant	0	0

TABLE 6.3: Statistics showing the average number of times a concept is relearned by agents using each approach.

- **The number of tasks attempted**: This affects the number of messages sent because each attempted task has the potential to require new information to decide on which action to take, therefore the greater the number of attempted tasks the greater the number of potential messages. For Experiment 3, we show the average

number of tasks attempted by an agent in a cumulative graph in Figure 6.9. We note that tasks attempted is not the same as tasks that were successfully completed because an attempted task is a task that is started but not necessary finished, and therefore we cannot use this as a success measure;

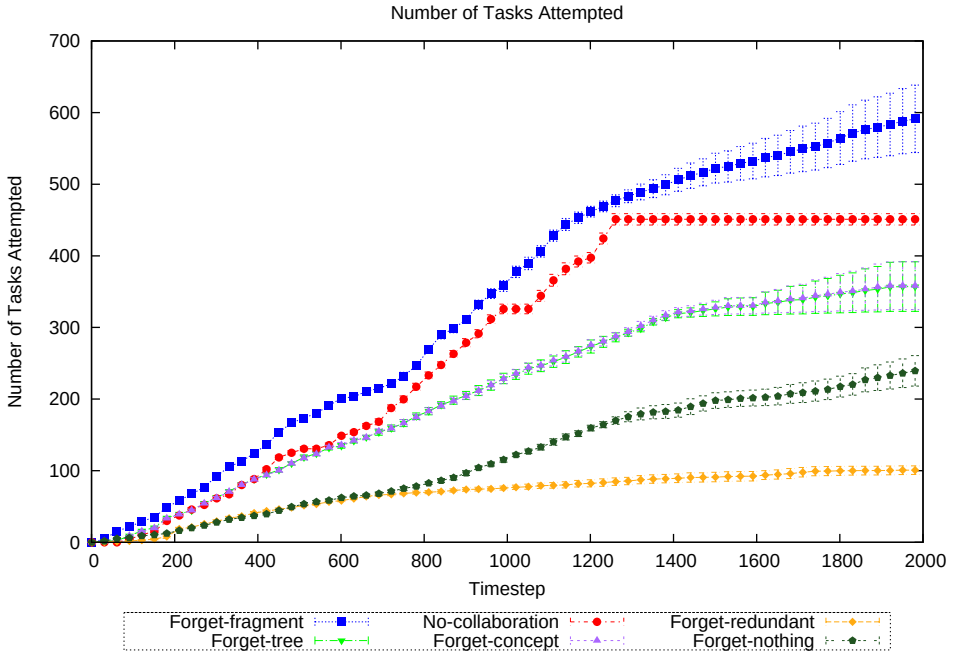


FIGURE 6.9: A graph showing the average number of tasks attempted by agents using each approach.

Approach	Average total number of tasks	Average minimum number of tasks	Average maximum number of tasks
Forget-fragment	594.36	452	923
Forget-tree	357.07	291	474
No-collaboration	451	426	479
Forget-concept	358.36	294	481
Forget-redundant	100.93	90	125
Forget-nothing	241.31	221	285

TABLE 6.4: Statistics of the number of tasks attempted by agents using each approach.

- **The number of tasks successfully completed**: The greater the number of tasks an agent successfully completes, the more accurate the concept rating is for each concept in the agent's EO. This is because the agents use the concepts in their EO more frequently (because they spend more time completing tasks successfully), and therefore they can accurately rate which concepts are least recently

and frequently used. We show the average number of tasks attempted by an agent in a cumulative graph in Figure 6.10.

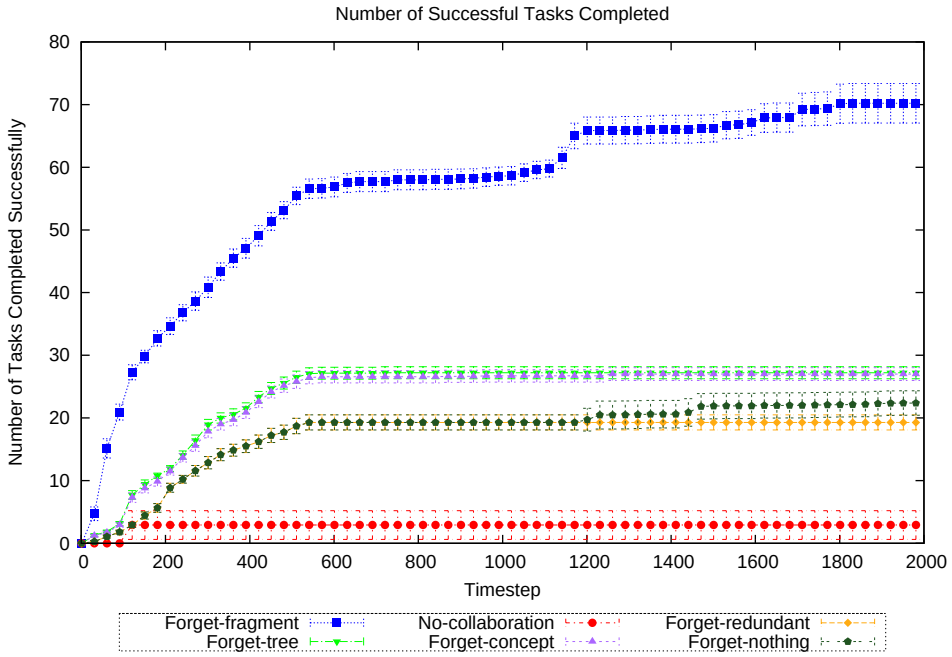


FIGURE 6.10: A graph showing the average number of tasks completed successfully by agents using each approach.

Approach	Average number of tasks per timestep	Average total number of tasks
Forget-fragment	0.0351	70.29
Forget-tree	0.0136	27.21
No-collaboration	0.0015	2.92
Forget-concept	0.0135	27
Forget-redundant	0.0096	19.29
Forget-nothing	0.0112	22.38

TABLE 6.5: Statistics of the average number of tasks completed successfully by agents using each approach.

These three factors increase the likelihood that an agent will send more messages in a simulation. Overall our approach, *forget-fragment*, attempts to complete on average 600 tasks which is the highest number of tasks attempted and relearns the fewest number of concepts (with the exception of the *forget-redundant* approach which does not relearn concepts because it has perfect foresight). Although our approach attempts to complete a high number of tasks, and relearns on average approximately 190 fewer concepts than

the other approaches it sends on average the fewest number of messages out of the approaches that forget and do not have perfect foresight (see Figure 6.11 and Table 6.6).

Our approach performs better, in terms of the number of messages sent, than other approaches, because it is able to perform forgetting tasks less often, and therefore is able to complete more tasks successfully (see Figure 6.10). Specifically, our approach, *forget-fragment* is able to complete more tasks successfully because it spends less time forgetting than the other agents, and thus can spend more time completing tasks. Completing tasks at a fast rate allows the agents to control the spread of the fires around the city environment, and by reducing the spread of fires reduces the chance that agents will encounter buildings that will burn out, allowing more agents to extinguish each fire. These factors increase the likelihood that fires will be put out, and therefore increases the number of successful tasks completed (see Figure 6.10). The rate at which our approach successfully completes tasks allows the agent to rate which concepts to forget more accurately than the other forgetting approaches (with the exception of the *forget-redundant* approach which has perfect foresight) and is therefore able to forget the least useful concepts reducing the number of concepts that need to be relearned (see Figure 6.8), and messages that need to be sent (see Figure 6.11).

The *forget-concept* and *forget-tree* approaches performed almost identically with regards to the number of messages (see Table 6.6), tasks attempted, and concepts relearned. This is because the *forget-tree* approach did not perform optimally because it was unable to find trees that were connected by the concept ratings, and subsequently removed the same number of concepts as the *forget-concept* approach. The *forget-redundant* and *forget-nothing* approach performed the same in regards to the number messages sent, this is because the *forget-redundant* approach has perfect foresight and does not relearn any concepts. The *no-collaboration* approach reaches a plateau in the number of tasks attempted at approximately timestep 1250 (see Figure 6.9), which indicates that there are no more tasks in the environment that are possible for the agents to attempt. This occurs because agents using the *no-collaboration* approach are unable to learn additional information when it is required for tasks, and thus any task that requires additional information is not possible for the agents to attempt. A consequence of this is that fires spread in an uncontrolled fashion through the environment, which initially offers more potential tasks in the form of building fires to extinguish, but leads to zero potential tasks when the buildings burn out completely, and all of the civilians die.

Table 6.6 shows averaged results from Experiment 3, and includes statistical information about the number of messages sent over a simulation and in a timestep.

In summary, we confirm our hypothesis that agents that use our approach, *forget-fragment*, send fewer messages than other benchmark approaches (with the exception of approaches that do not forget and do not have perfect foresight). This is because agents that use our approach complete more tasks successfully and spend more time working,

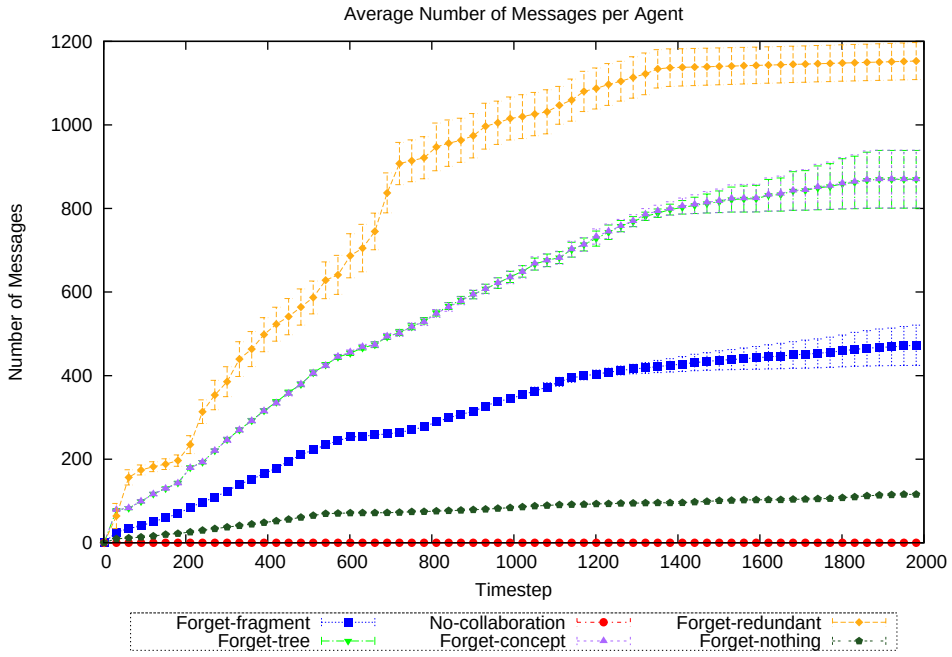


FIGURE 6.11: A graph showing the average number of messages sent by agents using each approach.

Approach	Average timesteps with no messages	Average max messages per timestep	Average number of messages per timestep	Average total number of messages
Forget-fragment	2.14	474.21	311.50	474.21
Forget-tree	2.14	869.89	581.19	869.89
No-collaboration	2000			
Forget-concept	2.21	870.42	581.86	870.42
Forget-repeated	2.14	1153.36	853.06	1153.36
Forget-nothing	2.24	116.32	76.88	116.32

TABLE 6.6: Statistics showing the average number of messages sent by agents using each approach.

rather than forgetting concepts, so the concept ratings applied to concepts in the agents' ontologies are more accurate, and concepts that are useful are retained. Fewer relearns are therefore required, and fewer messages are required to be sent.

6.4.2 Hypothesis 2 — Task Focused Ontology

In order to show that our forgetting algorithm successfully maintains a domain focused ontology and thus the lowest estimated complexity, we compare the number of concepts in agents' ontologies with the number of tasks attempted and completed (see Figures 6.9 and 6.10, and Tables 6.4 and 6.5). Agents using our *forget-fragment* approach attempt and complete on average more tasks than any other forgetting approach (see Figures 6.9 and 6.10), and therefore they require more concepts than the other approaches. However, the average number of concepts in an agent's EO that uses our approach has the lowest number of concepts in its ontology and thus the lowest ontology complexity, at any time in the simulation (with the exception of agents using *no-collaboration* and *forget-redundant* which has perfect foresight). This is because our approach forgets a subset of concepts and can therefore maintain a smaller ontology. The *forget-concept* approach maintains its ontology at approximately 499 concepts after reaching the capacity at 100 timesteps (see Figure 6.12), this is also the case for *forget-tree* because it is performing at its worse case. Without forgetting, the ontology would have grown to an average of 3143.68 concepts (see approach *forget-nothing* in Table 6.7) while our approach had an average of 358.81. Both our approach and the *forget-redundant* approachs' EO peaked in size around the 600th timestep because the civilians in the scenario are either all dead or rescued, therefore the agents required fewer concepts in their ontology from there on in. After the 600th timestep our approach maintains an average of 358.81 concepts in its ontology. The *forget-redundant* approach's EO contains all concepts that are required to complete all future tasks at any given timestep. Therefore, we can see that the agents that use our approach, *forget-fragment*, had to relearn many concepts during each scenario, however we discuss the effect of the approach's ontology size and relearning on their performance in our fourth hypothesis (see Section 6.4.4).

Approach	**Average final number of concepts**	**Average maximum number of concepts**
Forget-fragment	358.81	468.25
Forget-tree	484.08	524.87
No-collaboration	0	0
Forget-concept	484.09	524.79
Forget-redundant	0	1679.21
Forget-nothing	3143.68	3143.68

TABLE 6.7: Statistics of the number of concepts in the ontologies of agents using each approach.

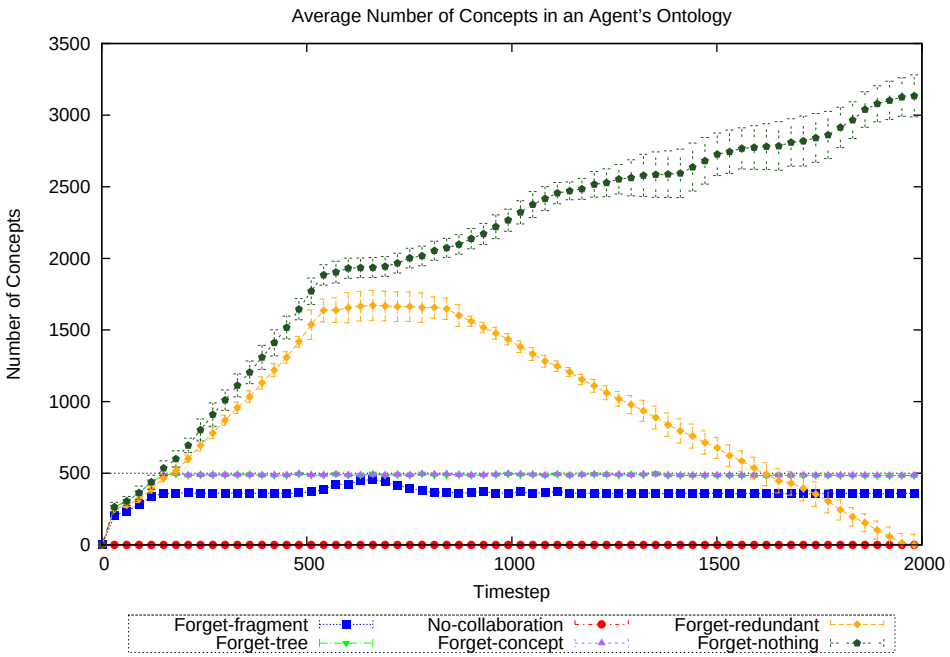


FIGURE 6.12: A graph showing the average number of concepts in the ontology of agents using each approach.

In summary, we confirm our hypothesis by showing that our approach, *forget-fragment*, maintains a smaller ontology and estimated complexity than approaches that forget (with the exception of the *forget-redundant* approach which has perfect foresight), and enables agents to complete more tasks than any forgetting approach. This is because our approach forgets more than one concept at a time, and the other approaches forget fewer concepts per timestep.

6.4.3 Hypothesis 3 — Time Spent Forgetting

Our approach, *forget-fragment*, spends the least amount of time forgetting, as shown in Figure 6.13. In order to show the results more clearly, we exclude the results from the *forget-redundant* approach in Figure 6.14 to reduce the scale of the y-axis. Using the results shown in Figure 6.14 we can see that our approach spends less time forgetting, this is because it forgets a set of concepts, rather than forgetting concepts one at a time, like *forget-concept*.

In more detail, agents using the *no-collaboration* approach do not spend any time forgetting, because they do not learn concepts. Similarly, agents using the *forget-nothing* approach do not forget any concepts, and therefore also do not relearn concepts. Agents using our approach, the *forget-fragment* approach, spend the least amount of time for-

getting compared to the other forgetting approaches and spend an average of 114.52ms forgetting. The next lowest approach, *forget-concept*, spends an average 686.45ms forgetting, which is 600% of the time spent by our approach. The time spent forgetting by agents using the *forget-tree* approach is comparable to that of those using the *forget-concept* approach, this is because the *forget-tree* approach is performing at its worse case. The *forget-redundant* approach spends the most time forgetting, averaging at 8503.4ms over the course of 2000 timesteps, which is 1154% of the time taken by the next largest approach, *forget-tree*. This is because the *forget-redundant* approach forgets one or more concepts when they will no longer be required to complete future tasks, thus the agent is required to analyse which concepts to remove each timestep.

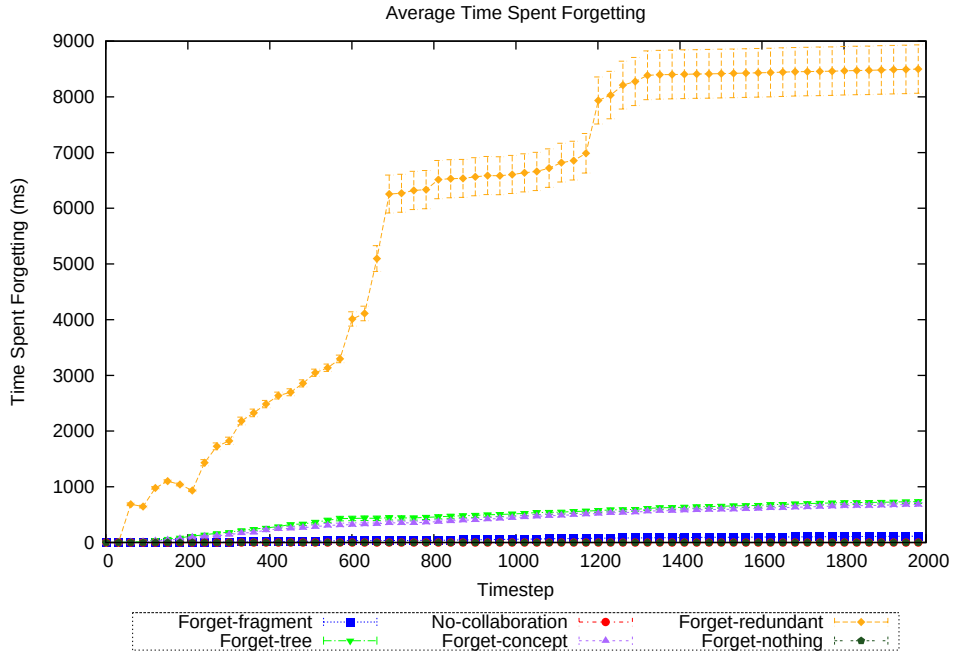


FIGURE 6.13: A graph showing the average time spent forgetting by agents using each approach.

Approach	Average total time spent forgetting
Forget-fragment	114.52
Forget-tree	736.57
No-collaboration	0
Forget-concept	686.45
Forget-redundant	8503.4
Forget-nothing	0

TABLE 6.8: Statistics of the average time spend forgetting by agents using each approach.

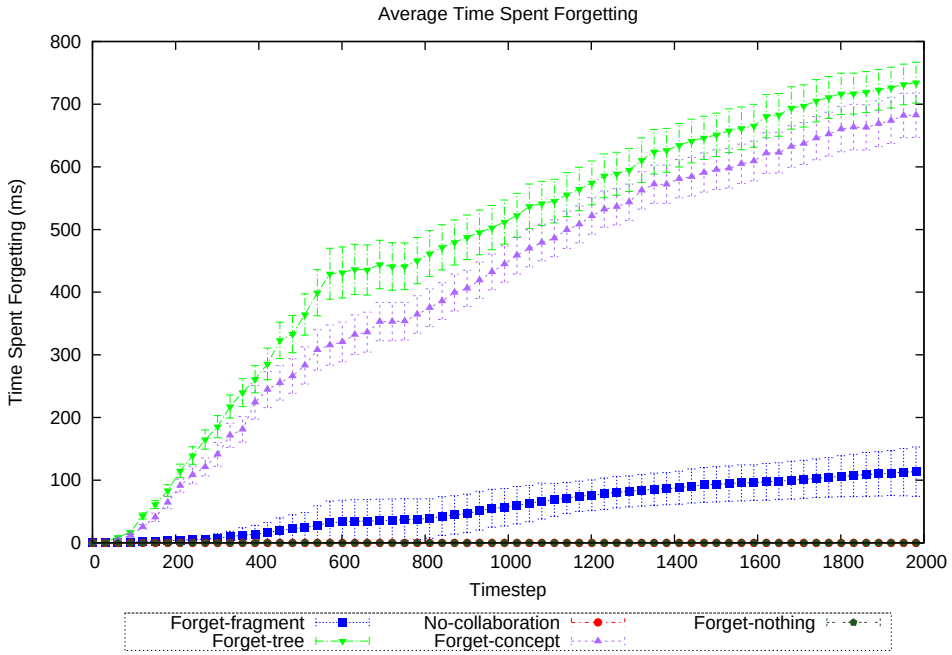


FIGURE 6.14: A graph showing the average time spent forgetting by agents using each approach, excluding *forget-redundant*.

In summary, we confirm our hypothesis by showing that our *forget-fragment* approach spends the least amount of time forgetting, of any approach that forgets concepts, which it achieves by forgetting multiple related concepts at once. Our approach spends only 16.7% of the time spent forgetting of the approach with the next lowest time spent forgetting, *forget-concept* (see Table 6.8).

6.4.4 Hypothesis 4 — RoboCup Rescue Results

We compare our results using the standard measures of the RoboCup Rescue *score vector* scoring system (see Section 2.5.3): saved civilians and buildings unburned. The agents using the *forget-fragment* approach outperform the agents using benchmark approaches, by having the highest average number of civilians rescued, 112% more than the next highest approach, *forget-concept*, as shown on Figure 6.15. The agents using the *forget-fragment* approach also outperform the agents using benchmark approaches by retaining 78% more unburned buildings than the next highest approach, *forget-reundant*, see Figure 6.16.

We now discuss the results of each of the approaches in more detail:

- The *forget-fragment* approach was able to save 36.2% of civilians, and 56.3% of

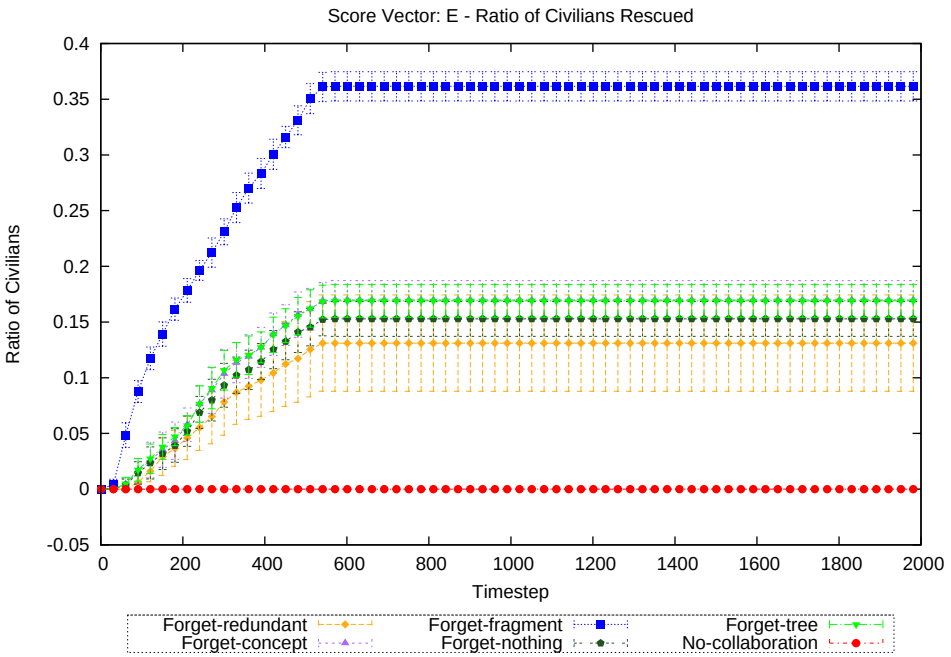


FIGURE 6.15: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

the city. This approach was the most successful in terms of the RoboCup Rescue scenario. This is because our approach:

- Spends the least time relearning concepts compared with other forgetting approaches (with the exception of the *forget-redundant* approach, see Table 6.3), and therefore incurred fewer costs associated with acquiring new concepts allowing agents to save more civilians and extinguish more burning buildings (see Figures 6.16 and 6.15);
- Spends the least amount of time forgetting, on average our approach spent 114.52ms forgetting where as the next best forgetting approach (*forget-concept*) spent 686.45ms which is approximately an increase in magnitude of 6. This allows agents using this approach to spend more of their time saving their targets compared to other forgetting approaches, and therefore performed better than the other forgetting approaches in regards to the RoboCup Rescue score vector (see Figures 6.16 and 6.15);
- Sends the fewest number of messages (53.6% of the next lowest approach) because it relearned fewer concepts than the other forgetting approaches (see Figure 6.11 and Table 6.6). This enables agents to reduce the amount of time spent sending and augmenting concepts into their ontologies and more time to save their targets as seen in Figures 6.16 and 6.15;

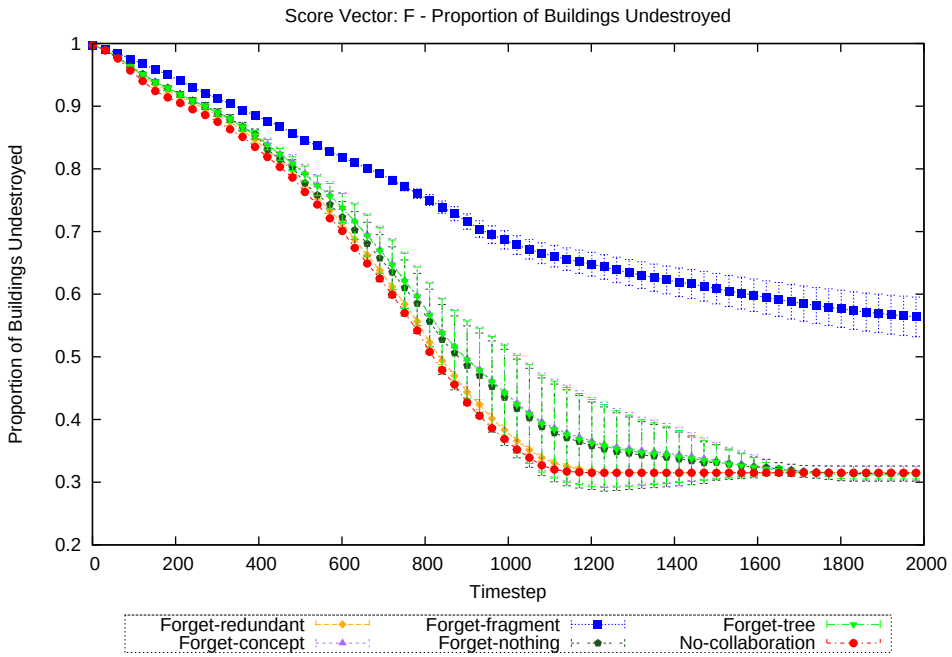


FIGURE 6.16: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

- Reduces the size of its ontology by forgetting more than one concept at a time. This enables agents using this approach to spend less time analysing which concepts to forget and remove from their ontology because they forget fewer times than the other forgetting approaches (with the exception of the *forget-redundant* approach). Thus, our agents benefited from being able to perform actions in a single time because of the capacity of the agent's EOs and removed concepts fewer times than the other approaches. This enabled the agents to save the greatest number of targets as seen in Figures 6.16 and 6.15;
- Completes the highest number of attempted tasks (19.3% more than the next highest approach, see Figure 6.9), and therefore was able to save more civilians and extinguish more burning buildings than the other approaches (see Figures 6.16 and 6.15).

- The *forget-redundant* approach was able to save 13.1% of civilians, and 31.5% of the city. In terms of RoboCup Rescue this approach performed the second best, but saved 63.8% fewer civilians and 44.0% fewer burning buildings than our approach. This is because the *forget-redundant* approach:
  - Spends the highest amount of time forgetting, it spends 1154% of the time

forgetting than the next closest approach (the *forget-tree* approach), thus preventing its agents from spending as much time saving targets compared with all other approaches. Hence, agents using this approach saved the fewest civilians and extinguished the fewest buildings compared with approaches that augmented their ontologies;

- Sends more messages than the *forget-fragment* approach, it sends 274% more messages (see Figures 6.11 and 6.6). Therefore, agents using this approach can spend less time saving civilians and extinguishing fires (with the exception of the *no-collaboration* approach which does not have the vocabulary to complete most tasks);
- Reduces the size of its ontology by removing one or more concepts that will not be used in future tasks, so that it does not have to relearn any concepts (see Table 6.3). This approach is hindered by the size of the agents' ontologies and therefore is unable to make many actions in a timestep (see Figure 6.12) and therefore is the second worst approach to use in RoboCup Rescue simulations (see Figures 6.16 and 6.15);
- Completes the lowest number of attempted tasks (with the exception of the no-collaboration approach), and completes 71.4% fewer tasks than the next best approach (the *forget-concept* approach);
- Spends no time relearning concepts (see Table 6.3) because it has perfect foresight it can accurately determine which concepts are of no use to an agent. This analysis causes the agent to spend the most time forgetting and thus agents using this approach save the second fewest targets (see Figures 6.16 and 6.15).

- The *no-collaboration* approach was able to save 0% of civilians, and 31.5% of the city. In terms of RoboCup Rescue this approach performed the worst; it saved 100% fewer civilians and 44.0% fewer burning buildings than our approach. This is because this approach:
  - Does not augment its ontology with new concepts, and therefore the agent cannot complete any tasks that its initial ontology is designed to support. Each civilian had symptoms that were unknown to the agent and therefore could not stabilise them so that they could move them to the refuge (see Figure 6.16). The fire brigade agents were able to extinguish 31.5% of the city because these 31.5% of the fires could be extinguished with suppressant on the vehicle that the agent operated (see Figure 6.15);
  - Completes the lowest number of attempted tasks, it completes 15% of the tasks completed by the next best approach (*forget-redundant*) and completes 52% fewer tasks than the *forget-fragment* approach.

- The *forget-concept* approach was able to save 17.0% of civilians, and 31.4% of the

city. In terms of RoboCup Rescue this approach performed the joint third best, and saved 53.0% fewer civilians and 44.2% fewer burning buildings than our approach. This approach's performance was similar to that of *forget-tree*, because the *forget-tree* approach performed at its worst case. Despite their similarity, *forget-concept* is marginally more consistent by 0.05%. In summary, this approach:

  - Relearns on average 389.21 concepts, which is 28.2% greater than the learn fragment approach (see Table 6.3). Therefore agents using this approach sent more messages (see Figures 6.11 and 6.6) than the learn fragment approach and thus are hindered by the costs associated with acquiring and augmenting new concepts. Therefore, agents using this approach are unable to spend as much time saving their targets as the *forget-fragment* approach (see Figures 6.16 and 6.15);
  - Reduces the size of its ontology by one concept at a time and therefore once an agent's ontology has reached its capacity it will contain 499 concepts (see Figure 6.12). This means when an agent augments its ontology with more than one concept then it will have to forget the same amount of concepts, this effectively swaps in and swaps out the concepts used for an action with the least frequently and recently used concepts. This is inefficient because an agent has to forget when it learns, therefore the agent can spend less time (see Figure 6.13 and Table 6.8) saving its targets than other approaches which do not forget every time they learn. It spends 29.2% more time forgetting than the *forget-fragment* approach;
  - Completes the fourth highest number of attempted tasks, and is only marginally worse then the *forget-tree* approach. This is because the agent reduces the size of its ontology by one concept at a time and when the *forget-tree* approach is not performing at its worse case, it removes more than one concept at time, and therefore spends marginally less time forgetting concepts.

- The *forget-tree* approach was able to save 16.9% of civilians and 31.2% of the city. In terms of RoboCup Rescue this approach performed the joint third best, and saved 53.3% fewer civilians and 44.6% fewer burning buildings than our approach. This approach's performance was similar to *forget-concept*, because it performed at its worst case where the majority of the time the algorithm could not remove more than one concept at a time because the concept selected to be removed did not have a similar concept rating. Despite their similarity *forget-concept* is marginally more consistent by 0.05%. In summary, this approach:

  - Relearns on average 389.5 concepts, which is 28.2% greater than the learn fragment approach (see Table 6.3). Therefore agents using this approach sent more messages (see Figures 6.11 and 6.6) than the learn fragment approach and thus are hindered by the costs associated with acquiring and augmenting new concepts. Therefore, agents using this approach are unable to spend as

much time saving their targets than the *forget-fragment* approach (see Figures 6.16 and 6.15);

- Reduces the size of its ontology by one or more concept at a time, however the majority of the time this approach removes only one concept at a time because there are no concepts that are directly connected by properties with a similar concept rating. Therefore once an agent's ontology has reached its capacity the majority of the time it contains 499 concepts (see Figure 6.12). This is similar to the *forget-concept* approach because an agent will swap in concepts that it uses and thus is inefficient because an agent has to forget when it learns. Therefore the agent can spend less time (see Figure 6.13 and Table 6.8) saving its targets than other approaches which do not forget every time they learn. It spends 29.2% more time forgetting than the *forget-fragment* approach;
- Completes the third highest number of attempted tasks, and is only marginally better then the *forget-tree* approach. This is because the agent reduces the size of its ontology by one or more concepts at a time and when it is not performing at its worse case, it removes more than one concept at time, and therefore spends marginally less time forgetting concepts than the *forget-concept* approach.

- The *forget-nothing* approach was able to save 15.3% of civilians, and 31.3% of the city. In terms of RoboCup Rescue this approach performed the fourth best, but saved 137.0% fewer civilians and 79.4% fewer burning buildings than our approach. This is because this approach:
  - Augments its ontology with new concepts and does not remove any concepts over the simulation. Therefore, agents' EOs grow monotonically, and do not relearn concepts (see Figure 6.8) or spend time evaluating which concepts to forget and removing them (see Table 6.8). Agents using this approach had the highest number of concepts in their ontologies. In more detail, they had 549% more concepts in their EOs than those in the next highest approach, *forget-concept* (see Table 6.7);
  - Sends the joint fewest number of messages (see Figure 6.11). In particular, it sends 24.7% fewer messages than the *forget-fragment* approach (the next lowest approach). This is because this approach does not relearn any concepts (see Figure 6.8), allowing agents more time to save their targets in comparison with the forgetting approaches. However, agents using this approach were unable to complete their actions in a timestep and therefore could not fully benefit from this advantage, and thus scored low in regards to RoboCup Rescue (see Figures 6.16 and 6.15);
  - Completes the second lowest number of tasks attempted (see Figure 6.9). It completed 31.8% of the tasks that our *forget-fragment* approach completed

(see Table 6.4). Therefore, agents using this agent could not save as many civilians or extinguish as many buildings as the *forget-fragment*, *forget-tree* or *forget-concept* approaches (see Figures 6.16 and 6.15).

In summary, we confirm our hypothesis by showing that agents which use our approach are able to save 113.0% more civilians and 79% more of the city compared to the next best approach (*forget-concept*). Our approach was able to outperform the other approaches because it relearns fewer times (see Section 6.4.1), sends fewer messages (see Section 6.4.1), has the smallest ontology (see Section 6.4.2), and spends less time forgetting (see Section 6.4.3) than the other forgetting approaches that did not have perfect foresight. It outperformed approaches that forget concepts (*forget-concept* and *forget-tree*) and had perfect foresight (*forget-redundant*), because it maintains a smaller ontology which decreases the amount of time it took to deliberate about its actions, and does not spend time analysing which concepts are required for future tasks.

6.5 Summary

In this chapter, we present our forgetting approach which removes the least recently and frequently used concepts from an agent's ontology. Our forgetting approach is applied to the RoboCup Rescue scenario (described in Chapter 3), and is novel because it enables an agent to selectively choose concepts to prune from an ontology with the use of a rating which determines how useful it is to store concepts in an ontology (partially satisfying Objective *2c*). This enables an agent to reduce the number of concepts represented in its ontology and reduces the ontology's potential complexity, thus reducing the time it takes to use, host and manage the ontology. We have published our forgetting algorithm and our evaluation in Packer et al. (2010b). The results from our evaluation support the hypotheses presented in Section 6.3.1, by showing that:

1. Our approach removes concepts from an agent's ontology and decreases the number of messages sent, because it does not have to relearn as many concepts as the other approaches (see Tables 6.3 and 6.6);
2. Our approach enables agents to complete the greatest number of tasks, while maintaining an optimal sized ontology and thus has the lowest estimated complexity (see Tables 6.5 and 6.7);
3. Our approach spends the least total time forgetting (see Table 6.8);
4. Our approach outperforms the other approaches that do not have perfect foresight, in terms of RoboCup Rescue, by saving on average more civilians and more of the city (see Figures 6.15 and 6.16).

The majority of our results were as expected, however, the performance of the *forget-tree* approach was unanticipated because its performance was not significantly different to the *forget-concept* approach. We expected that because the *forget-tree* approach would be able to forget more than one concept at a time and therefore negate the thrashing behaviour observed in the *forget-concept* approach. Where thrashing refers to two or more processes, our learning and forgetting algorithms, accessing a shared resource, the agent's ontology, repeatedly such that serious system performance degradation occurs because the system is spending a disproportionate amount of time just accessing the shared resource. This is because the *forget-tree* approach performed at its worst case scenario, by selecting one concept to remove at a time.

In view of these results, we claim that our approach maintains the size of an agent's ontology, and thus enables agents to achieve better results. Hence we have contributed to advancing the state-of-the-art in the domain of ontology learning and forgetting as we mentioned in our research Requirements 2c and 3c (see Sections 1.3.2 and 1.3.3). For future work, we intend to extend our framework so that it is an open environment. We hypothesise that our approach will be effective in removing concepts in an open environment. However, we note that because it is possible that an ontology could grow extremely large and therefore could perform considerably more forgetting operations than in the RoboCup OWLRescue framework, using our forgetting approach could hinder an agent's performance. Therefore, for future work we will evaluate our approach's effectiveness and build upon our forgetting algorithm so that we can evolve an ontology in an open environment effectively (see Section 7.3).

Chapter 7

Conclusions

This chapter concludes the thesis by summarising our underpinning requirements and contributions to the state-of-the-art which were outlined in Chapter 1. We then discuss the assumptions and limitations in this work. Following that, we detail how to overcome these limitations by extending this research in the future.

7.1 Summary

As stated in Chapter 1, the focus of this thesis is to aid in the development of ontology evolution techniques by developing online learning and forgetting algorithms for evolving an ontology through use. As noted in Section 1.3, we divided this research objective into three parts:

1. The development of an evaluation framework for evolving ontologies;
2. The development of the following online ontology evolution algorithms:
   (a) A reactive learning algorithm;
   (b) A proactive learning algorithm;
   (c) A forgetting algorithm.
3. An empirical evaluation mechanism that tests the efficiency of such online evolution algorithms.

In the following, we summarise how we fulfil these requirements with our state-of-the-art contributions.

In order to to fulfil Requirement 1 (see Section 1.3.1), we developed an agent framework called RoboCup OWLRescue (RCOR) which enables its agents to evolve their ontologies

by learning and forgetting concepts. The RCOR framework: i) simulates tasks, ii) provides agents with their own ontologies, iii) provides ontologies from which agents can learn and iv) provides standard success measures. These four properties are required to fulfil Requirement 1 (see Section 1.3.1). In more detail, we state how each of these properties are fulfilled:

1. **Task Environment:** The task environment is required to simulate agent based tasks which are generated according to a scenario which can be affected by the outcome of an agent's actions. In order to develop such an environment, we extended the standard RoboCup Rescue (RCR) platform because it simulates fires, injured civilians, and blockades, for fire brigade, ambulance and police agents to rescue, respectively. When agents rescue targets from this simulation it affects the outcome of future events within the environment (see Section 2.5.3). The RoboCup OWLRescue extension provides more attributes associated with rescuing a target (e.g. chemicals for buildings, symptoms for civilians, and heights for road blockades). This provides agents with information that they can deliberate over in order to decide on the action to take. For example, an ambulance agent can now decide which equipment to use to treat an injured civilian because the civilian has symptoms. Before the extension, the ambulance could not treat an injured civilian because it had no attributes relating to injuries, a civilian's only attribute was its health points. The extension also allows fire brigade agents to decide which equipment and suppressant to use to extinguish buildings which contain chemicals. Before the extension, the fire brigade agents could not decide on which suppressant to use because of two reasons: it did not have any suppressant equipment to choose from; and it did not have to extinguish fires that contained chemicals.

2. **Agent Model:** Agents are required to have their own ontologies so that they can deliberate about which actions to take given a task, and to employ ontology learning and forgetting algorithms. We implement this by extending RCR agents so that each agent has its own ontology (see Section 3.2.3).

3. **Domain Ontologies contained within the Environment:** The framework is required to provide domain ontologies that agents can learn from so that they can complete tasks that they could not before. We meet this requirement by creating and providing a set of environment ontologies from real data sources that contain information about: recommendations and guidelines for extinguishing fires, and treating civilians; and vehicle and equipment information (see Section 3.2.4).

4. **Performance Evaluation Measures:** We also require that our framework enables us to evaluate the differences in performance of agents using learning and forgetting algorithms, in terms of both the framework's scenario and comparison measures. We accomplish this by using the RoboCup Rescue score vector (see

Section 3.5.1) and using concrete and abstract comparison measures (see Section 3.5.2).

By fulfilling Requirement 1 (see Section 1.3.1), we contribute the first semantic extension to RCR by simulating tasks with more parameters, enabling agents to use their own ontologies, providing domain ontologies, and developing performance measures (see Contribution 3, in Section 1.4). This framework has been peer reviewed in the following two papers, Packer et al. (2010a) and Packer et al. (2010b).

Requirement 2 (see Section 1.3.2) requires the development of ontology evolution algorithms that enable agents to evolve their own ontologies which support tasks. We identified in Section 1.3.2 the three types of evolution algorithm we require:

(a) **An algorithm for incorporating new relevant knowledge into an agent's ontology (learning).**

The algorithm which we have presented in Chapter 4, fulfils Requirement 2a, with a reactive online learning algorithm which augments an agent's ontology based on information that is required to complete a task. This algorithm is reactive because it is used in response to a task, and is only used when information is required. The algorithm selects a set of concepts to augment into an agent's ontology given a target concept to learn. In order to select these concepts the algorithm uses the following steps. It first analyses the agent's ontology compared with the fragment to determine whether the fragment has a higher depth than that of the agent's ontology. The algorithm selects a set number of levels from the fragment from which to select, and limits the number selected to the average depth of the fragment and the agent's ontology. Second, the agent rates the concepts by the number of times that the concepts are referenced in the agent's ontology, and averages these ratings per level, which enables the algorithm to select the best levels to learn from. Finally, the target concept and concepts which relate to it through subsumption or properties such as properties that define the domain and range of a concept are selected (see Section 2.1.1). Through this algorithm, the agent selects a set of concepts from a fragment, reducing the size of the ontology and only augmenting concepts that closely relate to the agent's ontology. Thus keeping its vocabulary domain focused. We have satisfied Requirement 2a from Section 1.3.2 by developing an algorithm that learns new domain-focused knowledge into the agent's ontology, which uses a methodology designed to reduce the overhead resource costs of learning regularly required concepts.

(b) **An algorithm to predict which knowledge will be required in a scenario (learning).**

The algorithm which we have presented in Chapter 5 fulfils Requirement 2b, by presenting an online learning algorithm that is used to augment an agent's ontology

based on information that has been required in the past, given the state of the current task. This algorithm is proactive because it uses prediction to select which concepts to augment an ontology with. In particular it focuses on those that have high likelihood of use and in order to predict these concepts, we use a Markov model that represents the order in which concepts have been acquired in the past. We have satisfied Requirement 2b in Section 1.3.2 by developing an algorithm to predict the knowledge which will be required in future tasks. Our algorithm satisfies the requirement because it enables an agent to learn those concepts that have a high probability of occurring in the future, and avoid augmenting its ontology with those that do not.

(c) **An algorithm for removing irrelevant knowledge from an agent's ontology (forgetting).**

The algorithm which we have presented in Chapter 6 fulfils Requirement 2b, by presenting a forgetting algorithm which removes concepts from an ontology when an agent determines that the ontology is hindering its performance. The forgetting algorithm selects concepts that are Least Recently and Frequently Used (LRFU) and this value is weighted with a ranking that represents the relative cost (in terms of time) to acquire it. The acquisition cost of the concept is used as a factor because there is a trade-off between reacquiring cheap information and retaining information that was expensive to acquire. We have satisfied Requirement 2c in Section 1.3.2 by developing an algorithm that forgets the least recently and frequently used concepts from an agent's ontology. This approach satisfies our requirement because it is designed to prevent an agent's ontology from expanding without limit by removing concepts that are deemed the least useful.

By fulfilling Requirement 2, we contribute the first fully automated algorithms to enable an agent to learn and forget concepts from its ontology by developing three algorithms: a reactive learning algorithm; a proactive learning algorithm; and a forgetting algorithm (see Contribution 1, in Section 1.4). The reactive learning algorithm is described in the following papers, Packer et al. (2008), Packer et al. (2009) and Packer et al. (2010a), and the forgetting algorithm is described in Packer et al. (2010b).

We evaluate the performance of our three evolution algorithms using the RoboCup OWLRescue framework. Specifically, we use the following measures to evaluate the performance of each algorithm: framework success measure, algorithm-specific measures, and ontology complexity. In more detail, we overview why our approaches outperformed the other state-of-the-art approaches:

1. **Framework Success Measure:** We adopted the RoboCup Rescue score vector (see Section 3.5.1) to enable the comparison between different approaches. The score vector is comprised of a number of scores that measure different aspects of

the environment, such as the number of civilians rescued, the number of civilians in a refuge, the number of civilians in low, medium, good and full health and the number of buildings undamaged by fire. In our evaluation we compared the vectors representing the number of civilians saved and the number of buildings undamaged by fire. We used the vector representing the number of civilians saved because it represents the efficiency of the ambulance agents, and likewise we use the vector representing the number of buildings undamaged because it represents the efficiency of the fire brigade agents. We now summarise the results of the score vector measurements for each of our algorithms:

(a) **Reactive Learning Algorithm:** Outperforms the other approaches that do not have perfect foresight, in terms of RoboCup Rescue, by saving on average more civilians and more of the city. This is because it sends fewer messages, maintains a smaller ontology, thus spending the least amount of time acquiring and learning concepts. Agents using our approach saved 21.8% more civilians and 40.6% more of the city than the next best benchmark approach.

(b) **Predictive Algorithm:** Outperforms our reactive learning approach, in terms of RoboCup Rescue, by saving on average more civilians and more of the city. This is because it sends even fewer messages, maintains an even smaller ontology, and thus spends the least amount of time acquiring and learning concepts compared to our reactive learning algorithm. Agents using our approach saved 21% more civilians and 59% more of the city than the next best benchmark approach.

(c) **Forgetting Algorithm:** Outperforms the other approaches that do not have perfect foresight, in terms of RoboCup Rescue, by saving on average 112% more civilians and 78% more of the city than the next best benchmark approach. This is because it sends fewer messages, relearns fewer concepts, maintains a smaller ontology, and thus spends the least amount of time acquiring and learning concepts.

2. **Measures to evaluate an evolving ontology:**

In Section 3.5.2, we describe various measures that can be used to evaluate an evolving ontology, which include the number of messages sent, the size of an ontology, and the time taken to generate messages. We recorded and analysed these measures in our evaluations (see Sections 4.4, 5.5, and 6.4), we now discuss how our algorithms performed:

(a) **Reactive Learning Algorithm:** Reduces the number of messages required to complete tasks, compared with the agents that learn nothing (our approach sends 29.9% fewer messages than the next best approach). In fact, because our approach learns a subset of concepts instead of single concepts or every

concept, it is able to send fewer messages because the vocabulary in the agent's ontology supports the agent in completing tasks faster (because they have augmented their ontology with concepts related to their domain) than the other approaches.

(b) **Predictive Algorithm:** Sends fewer messages than our reactive learning approach, because it was able to predict which concepts would be useful for future tasks (our approach sends 103% fewer messages than the next best approach). Therefore, agents using this approach can request more than one concept at a time, and can prioritise augmenting an ontology with concepts that have a high likelihood of use while also actively avoiding augmenting their ontology with concepts that have never been used before.

(c) **Forgetting Algorithm:** Sends fewer messages than the other approaches because it successfully completes the most attempted tasks (agents using our approach completed 91.8% more tasks successfully than any other approach), and can more accurately remove concepts that are the least recently and frequently used.

3. **Algorithm-Specific Measures:** In Sections 3.5.2.3 and 3.5.2.4, we describe various algorithm-specific measures, such as time spent learning, number of concepts learned and number of axioms augmented into an ontology. These measures are specific to learning algorithms, while the measures of time spent forgetting, number of concepts forgotten, and number of concepts relearned are specific to forgetting algorithms. We recorded and analysed these measures in our evaluations (see Sections 4.3, 5.4, and 6.3) and summarise below our algorithms performed against them:

   (a) **Reactive Learning Algorithm:** The agents using this approach spent the least total time acquiring and learning concepts, because it augments more than one concept at a time.

   (b) **Predictive Algorithm:** The agents using this approach can request more than one concept at a time and they prioritise augmenting an ontology with concepts that have a high likelihood of use and actively avoid augmenting their ontology with concepts that have never been used before. They also spend the least total time acquiring and learning concepts, because they can predict which concepts have a high likelihood of use for future tasks.

   (c) **Forgetting Algorithm:** The agent relearns fewer concepts and is required to send fewer messages (agents using our approach relearned 22.6% fewer concepts and sent 46.4% fewer messages than the next best approach). Spends the least total time forgetting, because it forgets a subset of concepts from an ontology. Thus, agents using this approach can spend more time acting and saving their targets than other forgetting approaches (agents using

our approach spend 83.3% less time forgetting than agents using the closest benchmark approach).

4. **Ontology Complexity:** In order to evaluate the complexity of an ontology, which affects the time taken to infer logical consequences and thus the time to use it, we investigated different state-of-the-art complexity measures (presented in Section 3.4). From this, we concluded that these measures are not effective for measuring and comparing evolving ontologies, however we found that the number of concepts in an ontology can be used to provide a ranking of the ontologies in order of their complexity. We now summarise our three algorithms with regards to the number of concepts in their ontologies:

   (a) **Reactive Learning Algorithm:** Incorporates concepts from domain related ontologies because it successfully completes the highest number of tasks with the smallest ontology, against the other learning approaches. Agents using our approach had 33% fewer concepts in their ontology, and therefore the lowest *estimated complexity*, than the next best approach (excluding the *no-collaboration* approach which does not learn new concepts), and 5.8% more completed tasks than the next best approach.

   (b) **Predictive Algorithm:** Incorporates fewer concepts than our reactive approach, it successfully completes the highest number of tasks with the smallest ontology, against the other learning approaches. Thus, this approach creates a more domain focused ontology than any other approach without perfect foresight. Agents using our approach had 7.7% fewer concepts in their ontology than the next best approach, and therefore the lowest *estimated complexity*, (excluding the *no-collaboration* approach which does not learn new concepts), and 40.5% more completed tasks than the next best approach.

   (c) **Forgetting Algorithm:** Completes the greatest number of tasks and has the smallest ontology, therefore agents using this approach have a more focused domain ontology than any other forgetting approaches, and a lower *estimated complexity* (agents using our approach had 10.8% fewer concepts in their ontology than the next best approach).

By fulfilling Requirement 3, we demonstrate for the first time that learning and forgetting are effective ways for an agent to complete tasks in a specific domain (see Contribution 2, in Section 1.4). This contribution is shown in our empirical evaluations (see Sections 4.4, 5.5, and 6.4), and is described in the following peer-reviewed papers: Packer et al. (2010a) and Packer et al. (2010b). In the next section, we discuss the assumptions and limitations of our algorithms.

7.2 Assumptions

In this thesis, the work we have presented is bound by assumptions about the environment and ontology expressivity, as follows:

1. The set of ontologies in the environment does not contain any inconsistent axioms, therefore our approach does not handle inconsistencies. We use an open environment which has been *a priori* verified that there are no inconsistencies. This would not be possible in an open environment.

2. The agents in our environment use sequential messages to communicate, where message pairs are used to request and receive information and are handled in the order in which they are received. Therefore, we assume that minimising the number of messages is beneficial, whereas in other scenarios this may not be the case.

3. Messages sent within the environment are not affected by network latency or message loss, and therefore this is not a consideration for our work.

4. Our learning and forgetting algorithms can be used with OWL Lite and OWL DL ontologies. Our evolution algorithms will modify the concept subsumption hierarchy of an ontology and do not specifically modify any other semantic components. Thus, there will be no prerequisite of expressive features that an ontology should contain before our algorithms are applied. However, ontology evolution algorithms have not typically been performed with OWL Full ontologies, because they are not fully decidable. Their lack of decidability means that full verification cannot be performed, and thus checking for inconsistencies is not possible.

It should be noted that neither our learning or forgetting approaches guarantee that reasoning from an ontology is sound or complete. However, this is not always a requirement and a "good enough" answer is appropriate in cases where a response is required quickly (Fensel et al., 2008). For example, an agent that rescues casualties requires that it can respond before the patient's condition deteriorates, thus waiting for the optimal answer (where the entire knowledge base available in the environment is consulted) may prevent the casualties being rescued in time, while the "good enough" answer provides a quick enough response so that casualties can be rescued. There is a trade-off between these two cases, where the "good enough" response does not enable the rescue of the casualties, and the optimal answer does not provide a timely response. This trade-off is hard to balance, and "good enough" answers are formed from a semi-formal ontology which does not contain all of the concepts and relationships that describe a domain (see Section 2.1.1), and because the semi-formal ontology does not have complete knowledge of a domain it may produce an incomplete or incorrect answer. For example, an incomplete answer does not describe all possible treatments for an injured civilian, where: a

"good" incomplete answer provides an answer quickly enough to save the patient; and a "bad" incomplete answer provides treatments that are not appropriate for a civilian with allergies, because the ontology does not contain all of the information about which treatments trigger allergic reactions. More specifically, our approach is suited for:

1. Augmenting and pruning concepts into lightweight ontologies that support tasks. In contrast this approach is not suited to augmenting and pruning concepts from a formal ontology, because they are designed to encompass a complete domain and therefore have smaller scope to learn new concepts and little desire to reduce the information in the ontology;
2. Building a domain and task focused ontology on demand, because it uses an online algorithm that is designed to evolve an ontology with incomplete information on future tasks;
3. Augmenting and pruning concepts from ontologies published in any ontology language, because our learning and forgetting approaches use structural elements that are contained in all ontologies (see Section 2.1.1). In particular, these elements are concepts and properties;
4. Situations that require fast decisions in time critical scenarios. This is because our approach reduces costs associated with learning and forgetting concepts in an ontology, by reducing the number of messages, ontology size and concepts relearned.

In contrast to the above situations, our approach is not suitable for:

1. Pruning a formal ontology. A formal ontology is designed to describe all concepts in a domain (see Section 2.1.1) and thus by pruning concepts, a domain will not be fully described;
2. Agents in an open environment (as described in Section 2.2.1). This is because our approach has not been designed to scale the amount of information it learns according to the amount available, and learning from a large number of ontologies could result in the same performance as the *learn-everything* approach (see Section 4.4). In order to enable our algorithm to scale an agent could incorporate heuristics on which specialist agents have supplied the most useful knowledge, and therefore who to trust in the future.
3. Situations that require agents to collaborate. This is because two agents that use our evolution algorithm but do not share common experiences will not have the same concepts and relationships in their ontologies. Therefore the agents may not share a vocabulary, which may hinder their collaboration, unless they agree on a common vocabulary;

In this thesis we have presented our approach using the RoboCup OWLRescue framework, however our algorithms are general purpose approaches to selecting concepts to learn and forget from ontologies, and could therefore be applied outside of our framework to other situations that require lightweight ontologies which support tasks that need fast decisions. In the next section we discuss future work which aims to widen the applicability of algorithms by broadening our assumptions and addressing limitations.

7.3 Future Work

Although we have developed effective online learning and forgetting algorithms, there are several ways that the work in this thesis can be advanced. In order to identify how we can advance our work, we consider: First how to expand our work using the assumptions presented in Section 7.2. Specifically, Assumptions 1, 2 and 3 assume that our algorithms will be used in a closed environment; Second where our algorithms are best suited. This will enable us to identify the advances which would be the most beneficial for our algorithms. Beyond the RoboCup OWLRescue scenario, our algorithms are applicable to scenarios where there is:

1. The potential to learn new knowledge to support tasks. Therefore, an agent using our algorithm requires:

   (a) Tasks that were unforeseen at design time. The greater number of tasks that were unforeseen at design time increases the amount of information that an agent can augment into its ontology;

   (b) Bodies of information to learn from that are domain-related. The larger the set of information, the larger the potential size of an ontology.

   The result of having a large number of tasks and available information both increase the potential size of an ontology, and is dependant on the strategies used to augment information.

2. Limited resources that require agents to consider the efficiency of their actions, including:

   (a) Memory resources. Agents that use large ontologies require more resources to host, manage and use, therefore an agent with limited memory resources will require algorithms that are able to selectively learn and forget concepts from its ontology.

   (b) Time. Agents that have large ontologies require more resources to use than smaller ontologies, and therefore have slower response times.

Agents that have memory and time limitations, require that their ontologies remain small in size (in terms of concepts) while supporting tasks.

Scenarios with both of these constraints will benefit from using an approach such as ours. However, before we can apply this to our work we consider three extensions that make our approach more applicable to scenarios that display the characteristics above:

1. Extend the framework to enable agents to reason over their ontologies. While our algorithms have been used in a scenario where agents are not required to reason over their ontologies, the focus of some applications require the use of reasoners (Holford et al., 2010; Ruttenberg et al., 2009; Tappolet et al., 2010);
2. Extend the framework to send fragments representing a concept using different modularisation algorithms. This extension will allow us to investigate the efficiency of our learning and forgetting algorithms, and result in improving their performance, which is applicable to all applications;
3. Extend our learning and forgetting algorithms so that they can be used in an open environment. This extension increases the number of systems that our approach can be applied to.

First, we propose an extension to our RoboCup OWLRescue framework which enables agents to reason over their ontologies (as identified in Section 2.1.3). Currently, our agents make their decisions from axioms that are explicitly stated in their ontologies. In more detail, an agent can infer implicit facts using a reasoner, without which the facts would have had to be explicitly located and requested from a specialist agent, and then augmented into its ontology. This would enable the agent to reason over the classification of instances, subsumption relationships, and the consistency of concepts and its ontology (see Section 2.1.3). For example, an agent has augmented its ontology with the concepts `fire_truck` and `fire_engine`, these concepts are equivalent, however, this is not explicitly stated in its ontology. A reasoner would be able to identify that these two concepts are equivalent because they have the same property and subsumption relationships. Therefore, when the agent assigns drivers to vehicles, it will be able to assign drivers with certification to drive a `fire_truck` to a vehicle classified as a `fire_engine`, thus increasing the number of fire vehicles attending fires. Inferring information from an agent's ontology negates the need for acquiring some information and therefore would reduce costs associated with acquiring knowledge. Using inference also enables the agents to infer knowledge that may not be acquired from specialist agents, either because it is not explicitly stated, or an ontology has been removed from the environment. There are, however, costs associated with using a reasoner which includes the time and memory costs which vary given the constructs and structure of an ontology. These costs may negate the use of a reasoner and acquiring the required information may incur less cost

to the agent. For our future work, we aim to investigate this trade-off and assess when it is best to infer implicit knowledge or acquire knowledge given a particular task. This may be based on the amount of information or the constructs used to define concepts required for a task. While the use of reasoners offer many benefits to the agents, it is important to investigate the trade-offs in costs between using them and alternative strategies. This extension provides a new scenario to evaluate the performance of our learning and forgetting approaches.

Second, we propose another extension to the framework which enables specialist agents, which disseminate fragments of ontologies, with other fragmentation algorithms presented in Section 2.2.3. Currently our framework uses a single modularisation algorithm to provide our agents with fragments of ontologies to learn from. Each of the described fragmentation algorithms (in Section 2.2.3) use different features to select which concepts to include in a fragment. In the real world, however, not every agent which disseminates fragments of its ontology may use the same fragmentation algorithm because there are different types of algorithms used for producing ontology fragments (as detailed in Section 2.2.3). In order to show that our technique is also successful using alternative algorithms to generate fragments, we propose an empirical evaluation so that we can compare the performance of agents that select concepts from fragments that are generated using different algorithms. From this evaluation, we expect to see a trade-off between the relevance of the information included in the fragment and the time taken to generate the fragment. The benefit of using algorithms that use a reasoner to infer which concepts to include into a fragment is that the agent will be able to select concepts and relationships from all axioms that are used to define the target concept. However, depending on the size and complexity of an ontology, this type of algorithm may require more time and resources than an algorithm which uses structure alone to select concepts included in a fragment.

Third, we propose an extension to our learning and forgetting algorithms which enables them to work in an open environment. Currently, our framework RoboCup OWLRescue is a closed system where agents can learn from a limited number of ontologies. An open system such as the Web contains a plethora of ontologies, enabling an agent to access more vocabularies. The key differences between open and closed world systems are compared in Table 7.1.

As outlined in Table 7.1, the main difference between an open and closed world is that agents have access to more vocabularies and therefore an agent can access more resources in its environment. With access to more resources, an agent can allocate and deliberate about more types of resources, increasing the likelihood of successful task completion. However, if an agent then spends a disproportionate amount of time processing a large number of received fragments, it jeopardises its ability to complete tasks successfully in a short timeframe. Therefore, there are various challenges to overcome before an ontology evolution technique can be applicable to an open system. We discussed this challenge

	Closed World	**Open World**	**Effect**
1.	Fragment sources are limited to environment ontologies.	Fragment sources can be any ontology, provided by any third party entities.	The agent will need to consider from which sources to request fragments, from a potentially massive set.
2.	Domains of environment ontologies are known ahead of time.	Domains of ontologies are unknown until they are queried.	Each source will need to be queried to determine its domain, and therefore its suitability for use for a task.
3.	Ontologies are all individually consistent and the union of all ontologies is consistent.	Ontologies may contain inconsistencies, and ontologies may be inconsistent with each other.	Fragments will need to be consistency checked before use, as well as after merging with the agent's ontology. Inconsistencies will need to be resolved.
4.	Ontologies do not return very large fragments.	Ontologies may return fragments of very large sizes.	Large sized fragments may take extra time to transfer and process, and the agent will therefore have to consider if they can be processed within a timeframe useful for its task.
5.	Ontologies contain correct information.	Ontologies may contain incorrect or untrue information.	Agents cannot blindly trust information from external ontologies, and have to consider if the information they contain is correct.
6.	Fragment sources are reliable.	Fragment sources may become unreliable and stop returning fragments.	Agents must handle failure of fragment sources.
7.	Fragment sources return information immediately.	Fragment sources may be slow to respond to requests.	Agents must handle slow response times from sources, and consider their time constraints.

TABLE 7.1: Differences between Closed and Open environments.

in Section 2.2.1 and consider three of these challenges for future work:

1. **Trust**: It is possible that, in an open environment, information in an ontology may be unreliable, malicious or simply incomplete (Baumeister and Seipel, 2005), as described by (5) in Table 7.1. For the future, we consider evaluating related work into trust on the Semantic Web (such as Schenk (2008), Gao (2010), and Thuraisingham (2009)) and in multi-agent systems (such as de Oliveira et al. (2009), Smith and Desjardins (2009), and Mun et al. (2009)), and evaluate various trust models to test whether they prevent agents from learning incorrect information.

   In our experiments we did not introduce incorrect or incomplete information, since we were using a closed system. However, in an open system, where ontologies may include inaccurate information, agents are at risk of making decisions based on incorrect facts, and thus there is potential to lose utility (Finin and Joshi, 2002) and corrupt their ontology. Thus, agents should evaluate the fragments they receive, because their contents could be malicious or incomplete. Agents may also encounter slow response times from other agents that share their ontologies (as described by (6) and (7) in Table 7.1). In more detail, we describe how we plan to address trust issues with respect to our evolution algorithms:

   (a) Reactive Learning Algorithm: Using our reactive learning algorithm, it is possible to augment concepts that are i) incorrect, or ii) sources that are unreliable. We propose the following two approaches:

      i. We propose using 'tags' to record the source which provided an augmented concept. Therefore, when an agent uses a concept it can evaluate the outcome, if the outcome is negative then the agent can determine that the source is either malicious or contains incomplete information. We propose investigating two possible approaches:

         - Remove all concepts provided by the source;
         - Evaluate statements provided by the source against other trusted sources to check whether they have a common consensus.

      Existing work has shown that ontologies can be augmented with additional axioms to denote the authors of classes and predicates. Specifically, a framework has been presented Kotis et al. (2010) which utilises these trust features so that queries to the ontology can be limited by the trust value of the contributors;

      ii. We propose that agents use 'time outs' when requesting information from other agents. However, this approach would prioritise augmenting fragments from agents that respond the quickest. This may not be the best approach because more reliable and trusted sources, or have a larger more complete ontology, may be slower to respond because of the demand for

their services and size of their ontology, compared to other agents that respond quicker. Therefore, we propose an investigation into using variable time outs according to the quality and trust of a source.

(b) Proactive Learning Algorithm: In order to build upon the proposed trust model in our reactive learning algorithm, we propose tagging each Markov node with the sources used to acquire it in the past. This approach would require agents to regularly analyse whether they trust the source because it is possible that the reliability of a source changes over time. Thus, using tags in both the agent's ontology and Markov model would provide more accurate information than just one set of tags, for this evaluation.

(c) Forgetting Algorithm: The forgetting algorithm enables an agent to remove concepts that were provided by untrustworthy sources. We propose investigating measures that rank trustworthiness of sources, so that we can use it to evaluate which concepts to forget. Simplistically, we propose using such a measure to evaluate each concept in an ontology (similar to our presented forgetting algorithm in Section 6.2) and use it with our CFV rating:

$$CFV = LRFU + AC + TRUST\_RATING \tag{7.1}$$

2. **Scale**: Currently the RoboCup OWLRescue framework is a closed environment, where agents learn fragments from the environment ontologies. In an open environment such as the web, there is an abundance of published ontologies (Oren et al., 2009) which would require more time to acquire knowledge and time to process ontology fragments (see (1) and (4) in Table 7.1) than a closed environment. While it is costly in terms of time to retrieve and process ontology fragments in an open environment, it is also possible to augment an ontology with hundreds of concepts which in turn increases the time to use the ontology. In order to make our ontology evolution algorithms scalable to open systems, agents need to identify which ontologies to learn from so that they can manage the trade-off between learning more information and spending too much time learning. In more detail, we describe how we plan to address scale issues with respect to our evolution algorithms:

(a) Reactive Learning Algorithm: Using our reactive learning algorithm in an open environment, it would be possible to augment hundreds of concepts per learning operation. In order to allow our algorithms to scale to open environments, we propose that agents should i) share their ontologies and ii) evaluate the usefulness of fragments.

i. We propose that agents with the same domain share their ontologies. For example, in RoboCup OWLRescue the fire brigade agents have their own ontologies but share the same domain and purpose, therefore they could

share their ontologies. Such a system could be built around Electronic Institutions (Esteva et al., 2001) to enable ontology sharing among agents. Specifically, Electronic Institutions are interaction conventions for agents who can establish commitments on an open environment, in our case agents that have the same domain interest would share an ontology and evolve it according to their needs. Thus, agents could collaboratively build an ontology, rather than our current approach where each agent has its own ontology. In our framework there are many ontologies that contain the same information, this is because many agents require the same knowledge to collaboratively handle tasks. It is therefore possible to reduce the cost of learning by sharing an ontology because it reduces the number of messages and time spent learning. However, if collaboratively sharing tasks is infrequent and each task requires a large amount of knowledge which reduces response times, sharing an ontology may hinder an agent's performance unnecessarily. We aim to investigate this trade-off by identifying the scenarios where sharing an ontology is beneficial.

ii. In order to determine the usefulness of fragments, an agent could use an evaluation stage that enables the it to decide on the potential usefulness of received fragments. This enables the agent to decide which subset of ontologies to learn from when completing tasks and will reduce the risk of requesting fragments from ontologies with different domain interests. There are two potential approaches to evaluate the usefulness of an ontology:

- Evaluate the domains of two ontologies using similarity measures, as described by Maedche and Staab (2002) and Euzenat and Valtchev (2004).
- Evaluate how well each ontology helps to complete domain tasks (Porzel and Malaka, 2004), and to rank the ontologies on their performance on each task.

We plan to build upon the work in the area of ontology ranking (Alani et al., 2006a) and collaborative recommendation of ontologies (Romero et al., 2010), to aid agents in the selection of either highly ranked or highly recommended ontologies, respectively.

(b) Proactive Learning Algorithm: In order to enable our proactive learning algorithm to be more scalable, we propose that agents that share the same domain should share their Markov models. Currently agents each have their own Markov models, and a shared Markov model for a domain would enable the agents to benefit from other agents' experiences. However, a Markov model may grow very large and could be inefficient to use. Therefore, we will investigate the effect of sharing Markov models between a set of agents.

(c) Forgetting Algorithm: In order to enable our forgetting algorithm to be more

scalable, we propose using a proactive forgetting algorithm. Our proactive learning algorithm saw benefits over our reactive learning algorithm by using a prediction model because it was able to maintain a smaller ontology while supporting more completed tasks. We were able to achieve this by learning fewer concepts per operation, and we hypothesise that we can remove more concepts per operation using a predictive model while supporting the same amount of future tasks. Similar to our proactive learning algorithm, we propose that a proactive forgetting algorithm will use a Markov model, but in this case it will represent the usefulness of concepts. In our forgetting algorithm we currently wait for a concept's CFV to degrade, but using a Markov model to predict which concepts to forget enables an agent to remove more unnecessary concepts sooner. This results in a smaller ontology and reduces the costs associated with using it.

3. **Consistency**: Effectively resolving ontology inconsistencies automatically is a hard problem, which stands as one of the key issues within Semantic Web research (Plessers and De Troyer, 2006; Haase and Stojanovic, 2005b; Sirin and Parsia, 2004). In our RoboCup OWLRescue framework, the union of the environment ontologies contained no inconsistencies (see Section 4.2.1). We proposed Algorithm 7 in Section 4.2.1 to enable an agent to remove inconsistent axioms from a fragment so that it could select concepts to learn without introducing inconsistencies. This algorithm was not implemented in our framework and is therefore not evaluated in this thesis. Such an algorithm is more important in an open environment because they contain many ontologies which are developed independently and can contain contradictory axioms about the same concept, as described by (3) in Table 7.1. For future work, we plan to evaluate and develop techniques to guarantee the consistency of an ontology, such as our proposed Algorithm 7 presented in Section 4.2.1. In more detail, we describe how we plan to address consistency issues with respect to our evolution algorithms:

   (a) Reactive Learning Algorithm: Using our reactive learning algorithm does not prevent an agent from introducing inconsistencies into its ontology. We propose that an agent should i) blacklist sources that repeatedly contain inconsistent information, ii) consider how to resolve inconsistencies and iii) check whether augmenting a fragment will introduce consistencies into its ontology. In more detail:

      i. We propose that agents do not learn from other agents' ontologies if they are repeatedly found to contain inconsistent axioms. This will potentially reduce the number of inconsistencies introduced into the agent's ontology.

      ii. An agent can deal with inconsistencies in the following ways:

         - Do not augment inconsistent axioms into the agent's ontology.

- Evaluate the inconsistent axioms and if the inconsistent axioms in the agent's EO have a low LRFU value then remove them before augmenting the received axioms.

We also propose to evaluate the effectiveness of these two approaches.

iii. We plan to evaluate the effectiveness of Algorithm 7 presented in Section 4.2.1. This algorithm has the order of $O(n^n)$, and thus in the worst case can be very time costly. Therefore, we propose that if a suitable set of fragments cannot be located within a timeframe, the agent should then select the first fragment that does not contain any inconsistencies with the agent's ontology. This will reduce the time spent finding a consistent fragment. We will evaluate this approach against selecting the first random single fragment with no inconsistencies. This will enable us to evaluate the effect of an agent's performance using a fragment from a single source or many sources.

(b) Proactive Learning Algorithm: In order to enable our Markov model to support our agents in selecting information without inconsistencies, we propose that they should contain a list of sources that have been found to be inconsistent in the past. This would enable agents to reduce the potential number of inconsistencies to resolve.

(c) Forgetting Algorithm: The forgetting algorithm enables an agent to remove concepts that are inconsistent with concepts that will be augmented into an agent's ontology. We propose extending our forgetting algorithm so that it can evaluate which concepts to forget based on how many times they were found to be inconsistent with other sources. In more detail, we propose extending our CFV rating to include the variable $INCONSISTENT\_COUNT$ which a ranked number of times a concept has been found to be inconsistent.

$$CFV = LRFU + AC + INCONSISTENT\_COUNT \tag{7.2}$$

Removing concepts that have been known to be inconsistent with many other sources has the potential to increase the number of useful concepts into an ontology.

These three challenges have started to be addressed by the Semantic Web community, however, there are limited conclusions. It is important to tackle these challenges in order to enable the use of an open environment as a source of knowledge for autonomous agents. By using an open environment, agents can interact with a greater number of resources and services, and adapt the knowledge they store in their ontology to reflect any changes in the environment and its users' needs. This is important because in real life situations users' needs change and so do environments. For example, medical science is continually changing with the creation of new drugs and treatments, and recommendations for how

to treat patients, and agents that learn such information can achieve more success than those that use out of date information. Using an open environment also enables agents to expand their knowledge dynamically without the need for manually editing ontologies or adding new ontologies into a closed environment. In order to advance the state-of-the-art, our proposed future work will:

1. Contribute the first semantic extension to RCR that enables its agents to perform reasoning tasks over their ontology. This would extend our first contribution (see Section 1.4) by enabling agents to perform reasoning tasks, and uses the first part of our future work as a requirement.

2. Develop the first fully automated technique to enable an agent to learn and forget concepts from its ontology in an open environment. This contribution would extend our second contribution (see Section 1.4) by enabling our algorithms to perform in an open environment and uses the second and third parts of our future work as requirements. In more detail, we consider:

   (a) Developing learning algorithms that select concepts to learn from fragments created using different modularisation techniques (as described in part two of our future work).

   (b) Developing automated algorithms that can handle scaling problems, trust issues and that guarantee consistency of an evolving ontology (as described in part three of our future work).

3. Demonstrate for the first time that ontology evolution in an open environment, in terms of an agent learning and forgetting concepts from its ontology, is an effective way for an agent to complete tasks in a specific domain in a changing environment. This contribution extends our third contribution (see Section 1.4) by evaluating online learning and forgetting algorithms in an open environment in contrast to a closed environment. This contribution evaluates all three parts of our future work.

Appendix A

Additional Results from our Evaluation of the Reactive Learning Algorithm

In this appendix, we present additional results from our reactive learning experiments, Experiments 1 and 2 (mentioned in Chapter 4). The parameters used for both of these experiments are listed in Table A.1.

	Experiment 1	**Experiment 2**
Environment ontologies	10	10
Number of agents	20	20
Number of fire brigades	10	0
Number of police	10	10
Number of ambulance	0	10
Number of time steps	2000	2000
Map	Kobe	Kobe
Benchmark approaches	Learn-repeated, Learn-everything, Learn-concept, No-collaboration	Learn-repeated, Learn-everything, Learn-concept, No-collaboration
Number of civilians	115	155
Iterations	250	250

Table A.1: The parameters for Experiments 1 and 2.

The next two sections presents the results from Experiment 1 and 2, respectively.

A.1 Experiment 1: Fire Brigade Agents

In this section we present the results of Experiment 1, which focuses on the performance of the fire brigade agents in the RoboCup OWLRescue framework using our reactive learning approach. The purpose of this experiment is to show how augmenting select concepts into an agent's ontology benefits the fire brigade agents. We use the experimental setup in Table A.1. From the results, it shows that the fire brigade agents using our approach send fewer messages per attempted task, compared to the agents using other benchmark approaches (see Figure A.6 and Table A.2). This is because our approach learns more than one concept at a time. Agents using our approach complete the greatest number of tasks with the smallest ontology, compared with the other learning approaches (see Figure A.5 and Table A.4). Also, agents that use our approach spend the least time learning and deliberating, and more time acting compared with the other approaches. In terms of RoboCup Rescue, our approach outperforms the other approaches that do not have perfect foresight by saving on average more civilians (see Figure A.2) and more of the city (see Figure A.3).

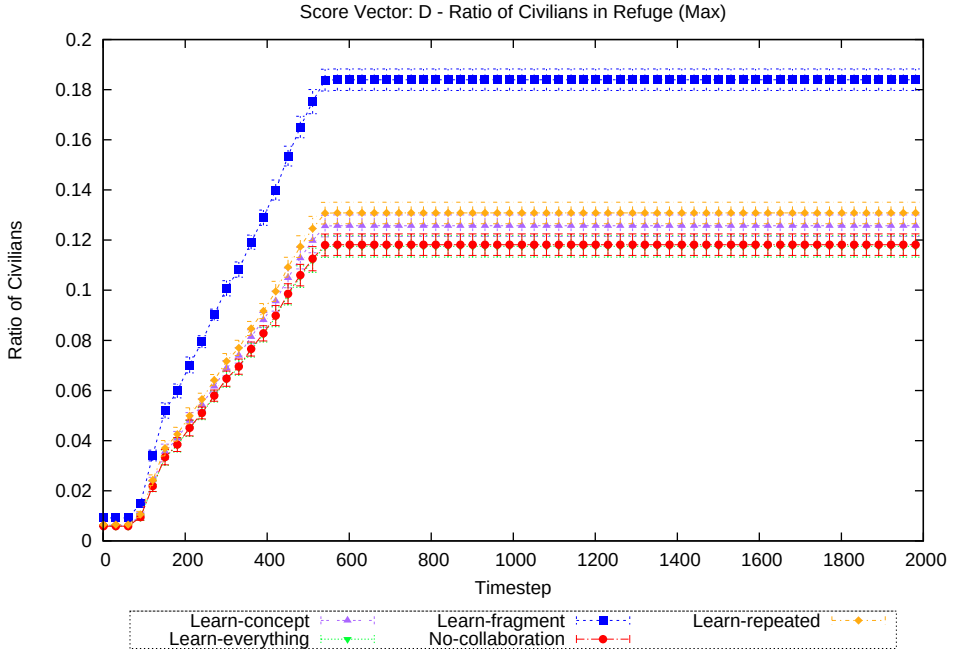


FIGURE A.1: A graph showing the RCR score vector D, showing the average ratio of civilians in the refuge.

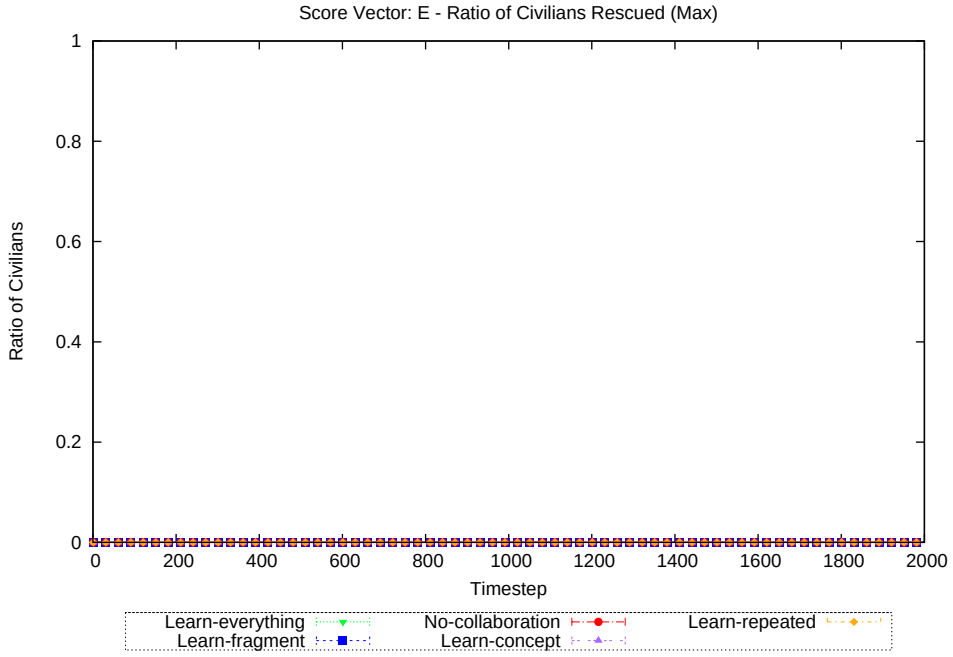


FIGURE A.2: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

Technique	Average final value
Learn-everything	94.37
Learn-repeated	106.82
Learn-concept	232.00
Learn-fragment	64.44
No-collaboration	0

TABLE A.2: Statistics for the number of messages sent per agent.

Technique	Average final value
Learn-everything	2550.47
Learn-repeated	2887.09
Learn-concept	3135.41
Learn-fragment	2087.90
No-collaboration	0

TABLE A.3: Statistics for the number of concepts in each agent's ontology.

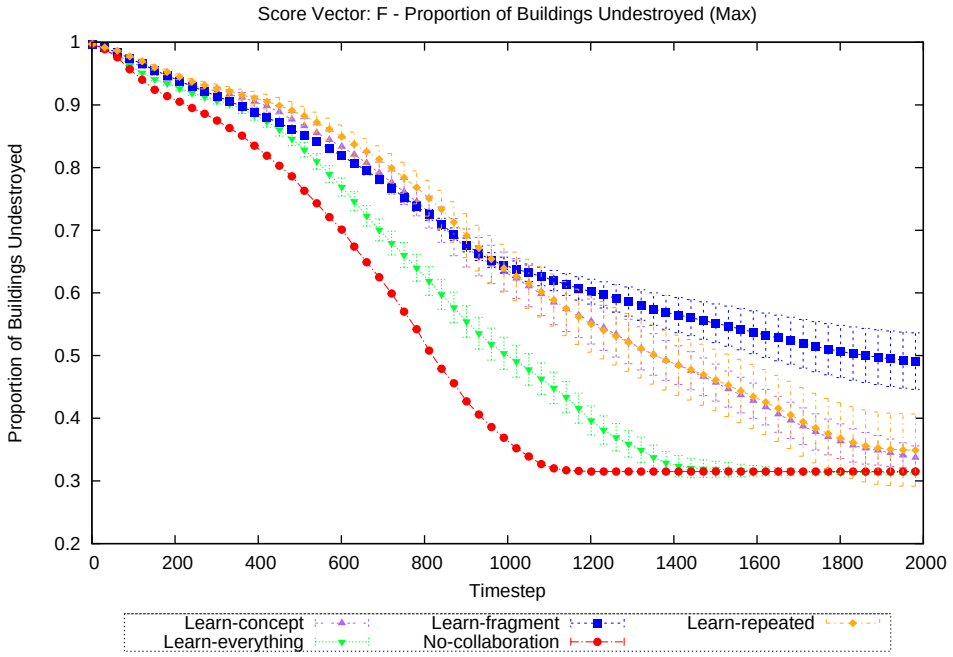


FIGURE A.3: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

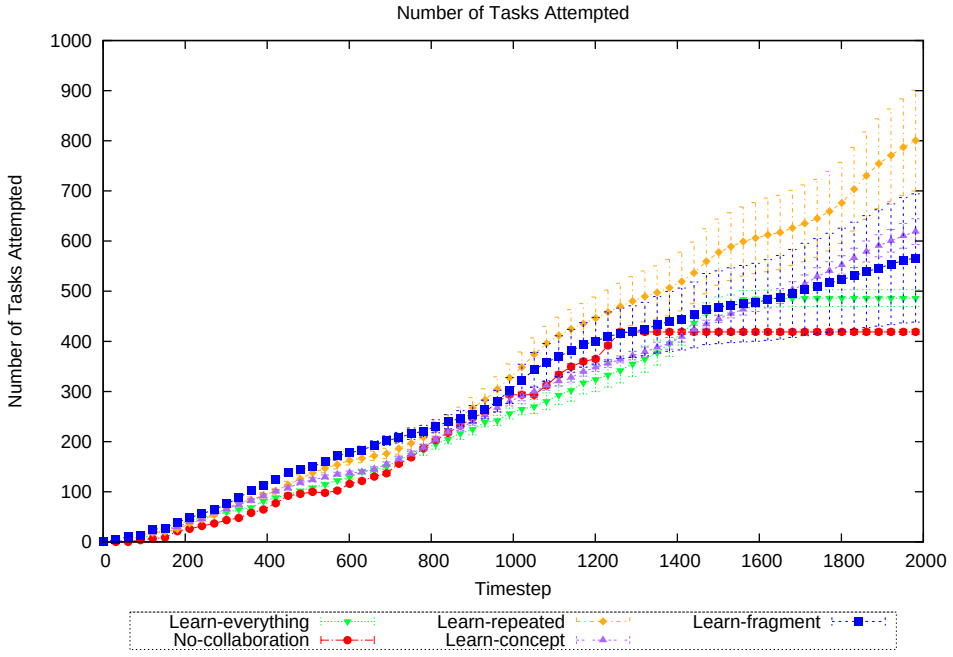


FIGURE A.4: A graph showing the average number of tasks attempted per agent.

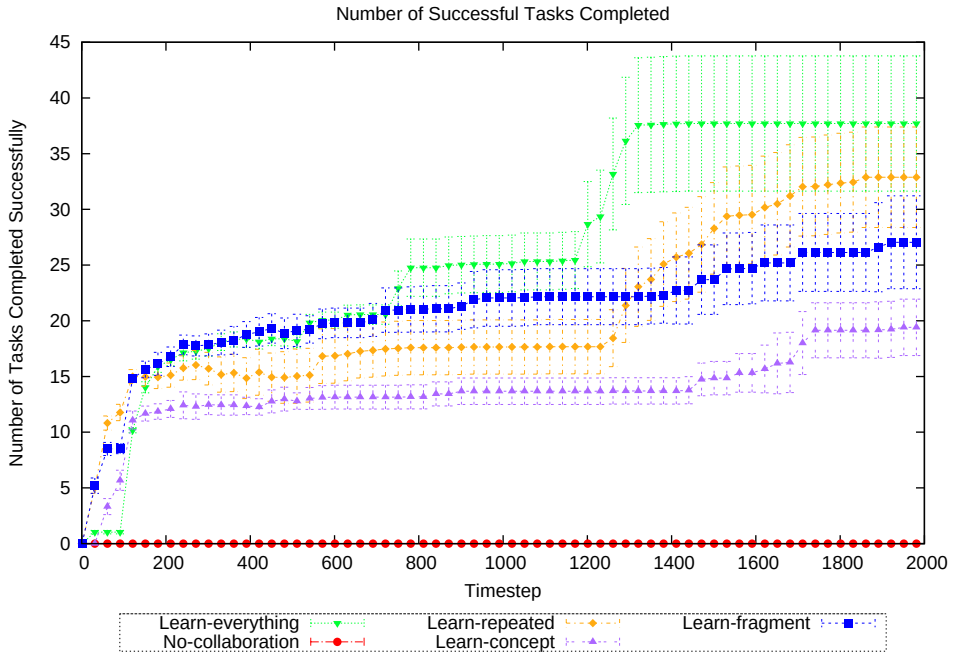


FIGURE A.5: A graph showing the average number of successfully completed tasks per agent.

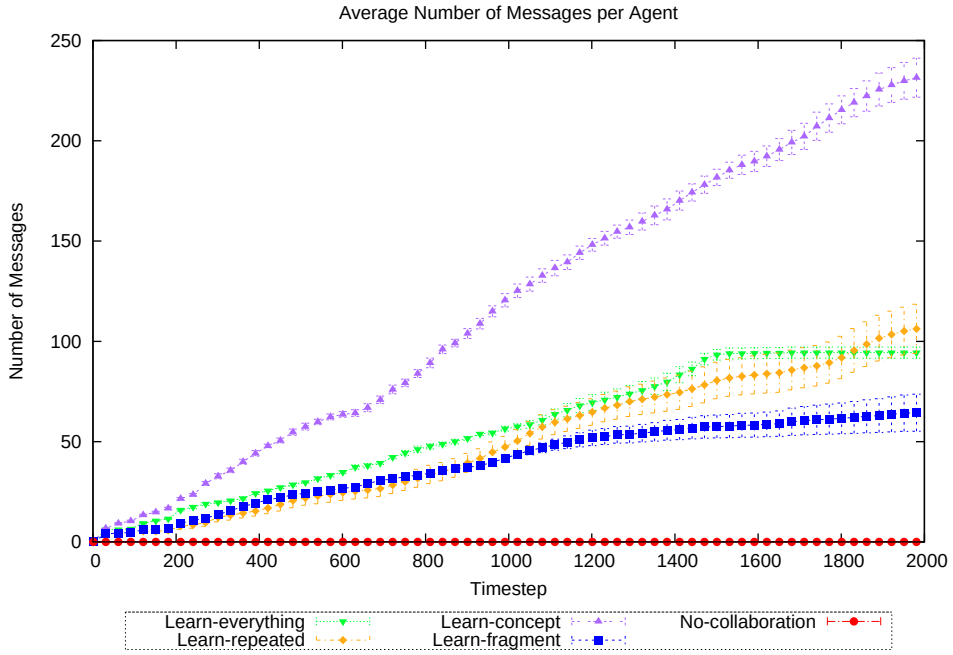


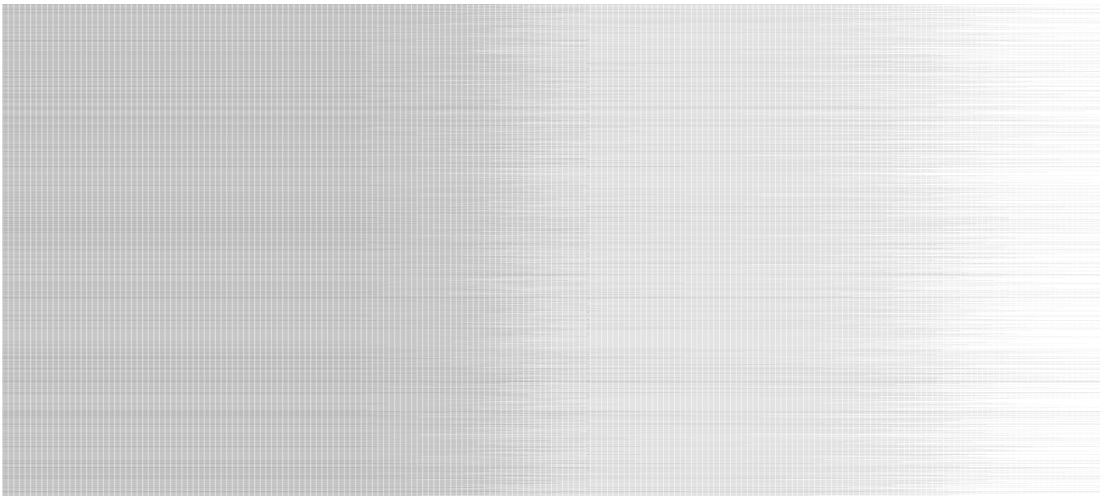
FIGURE A.6: A graph showing the average number of messages sent per agent.

Technique	**Average final value**
Learn-everything	37.71
Learn-repeated	32.88
Learn-concept	19.41
Learn-fragment	27.05
No-collaboration	0

TABLE A.4: Statistics for the average number of successful tasks an agent performs.

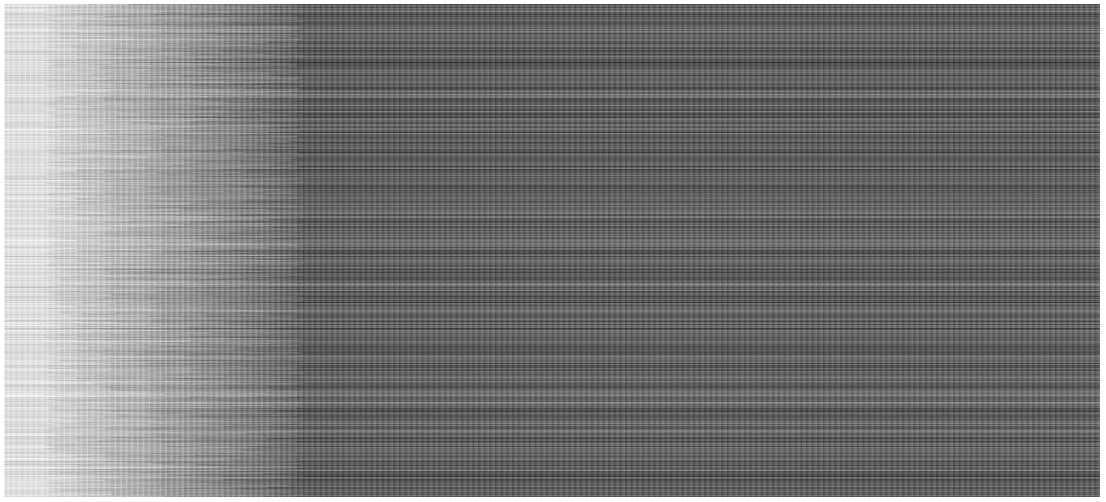
(a) Pixel-plot showing the time spent learning for the *learn-everything* approach.

(b) Pixel-plot showing the time spent deliberating for the *learn-everything* approach.

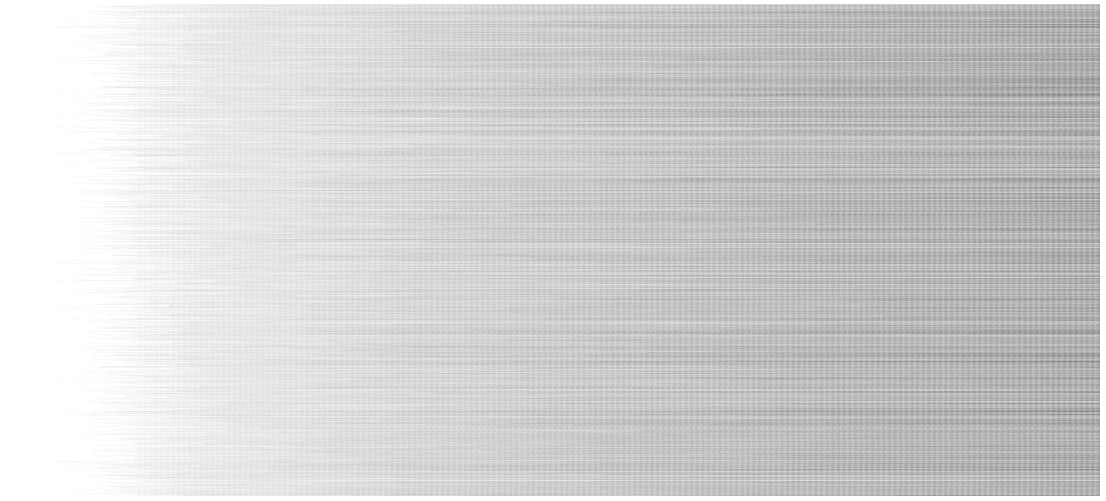


(c) Pixel-plot showing the time spent acting for the *learn-everything* approach.

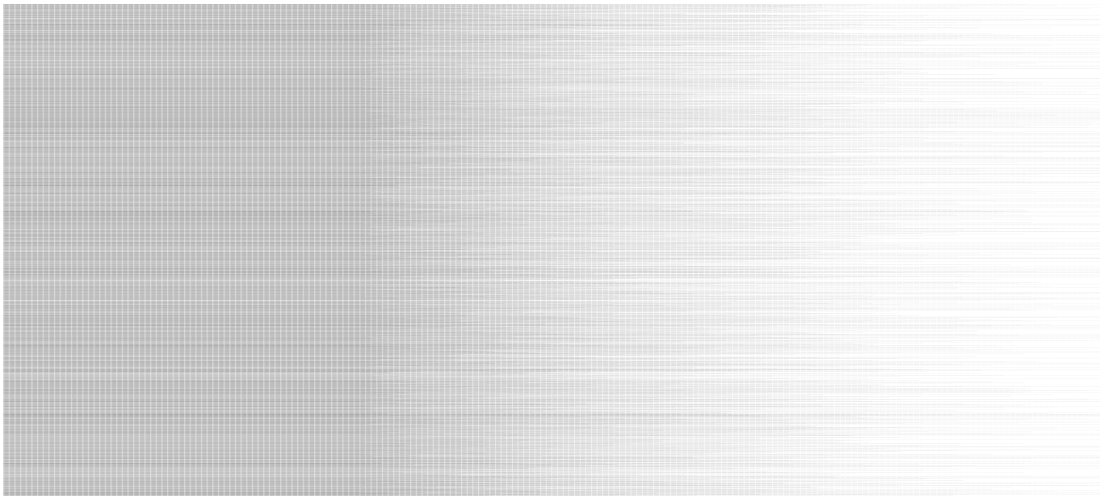
FIGURE A.7: Pixel plots for the *learn-everything* approach



(a) Pixel-plot showing the time spent learning for the *learn-concept* approach.

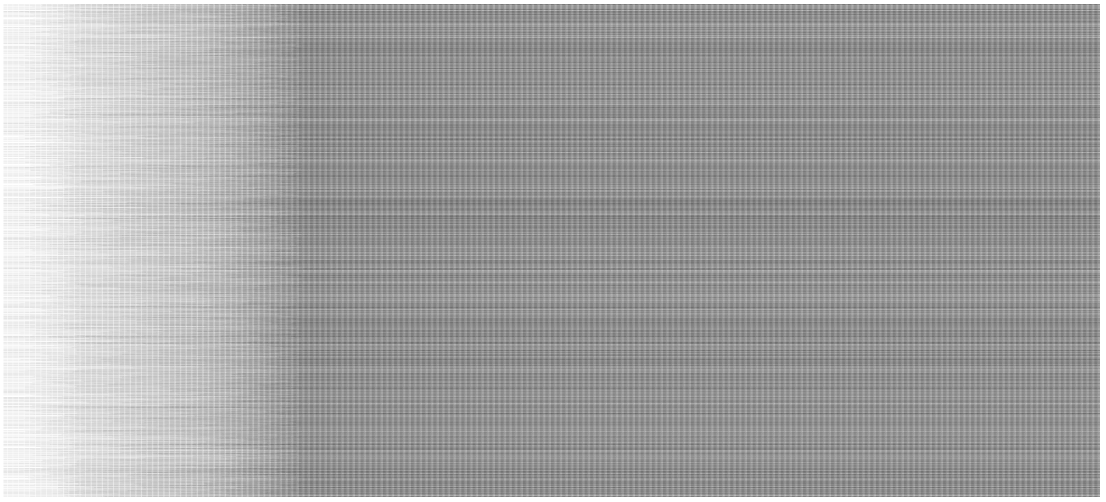


(b) Pixel-plot showing the time spent deliberating for the *learn-concept* approach.

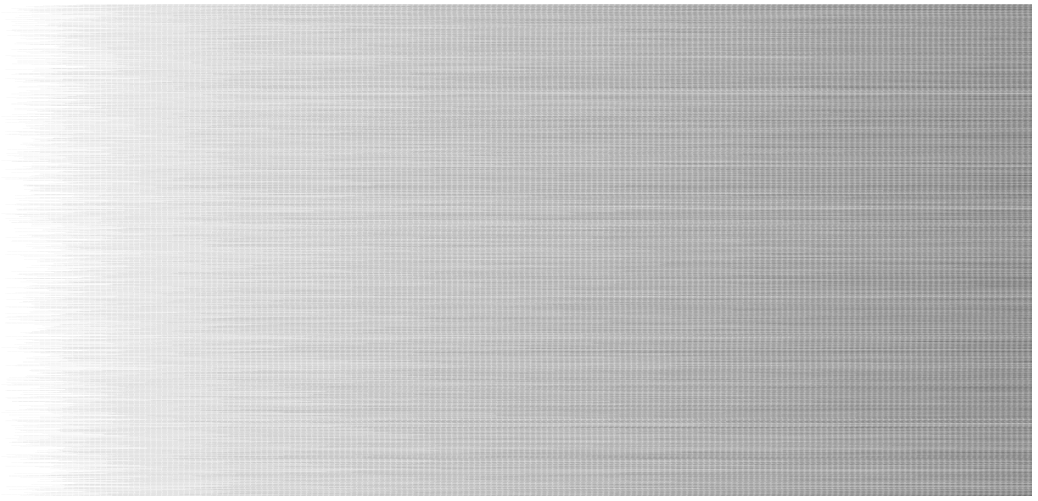


(c) Pixel-plot showing the time spent acting for the *learn-concept* approach.

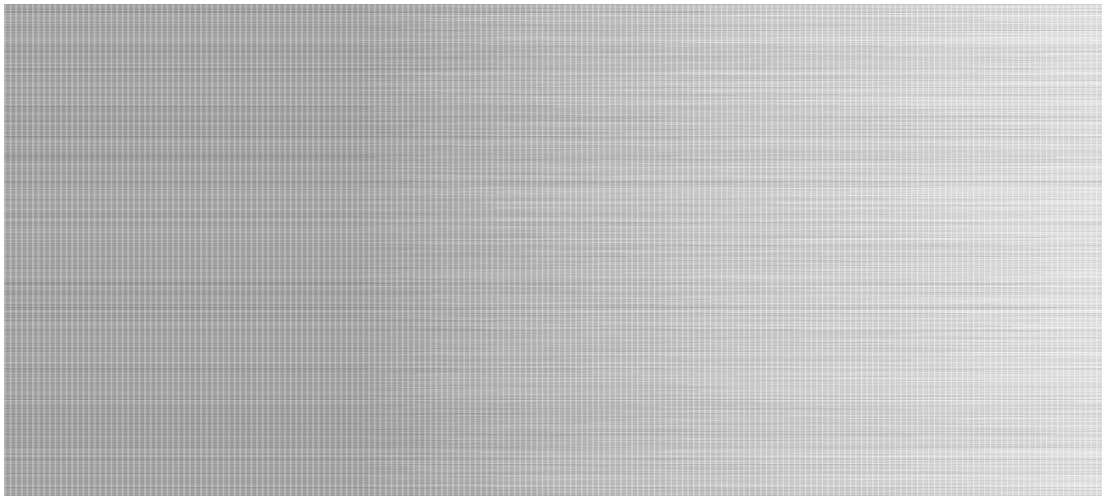
FIGURE A.8: Pixel plots for the *learn-concept* approach



(a) Pixel-plot showing the time spent learning for the *learn-repeated* approach.

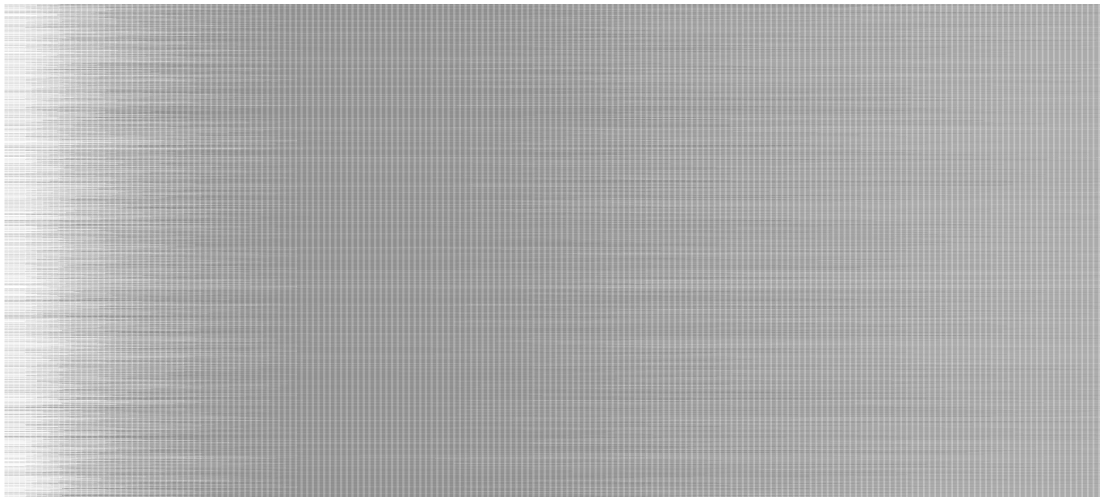


(b) Pixel-plot showing the time spent deliberating for the *learn-repeated* approach.

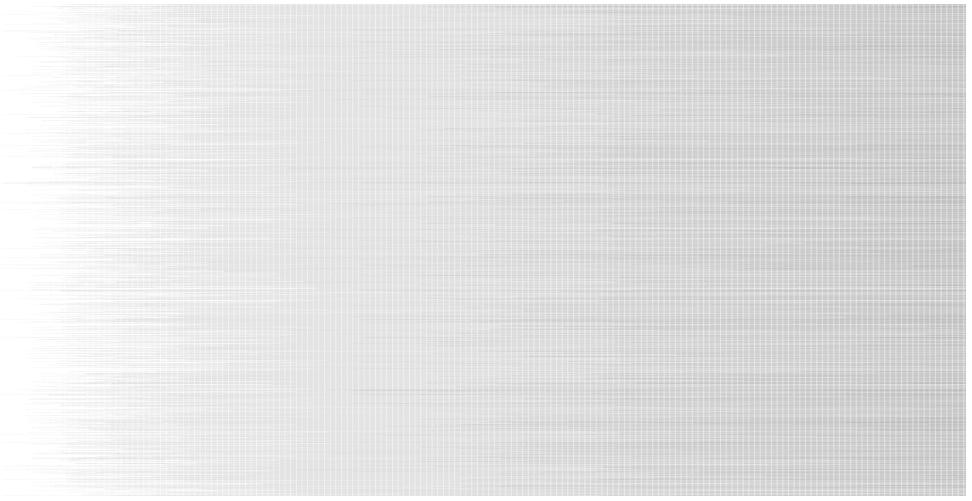


(c) Pixel-plot showing the time spent acting for the *learn-repeated* approach.

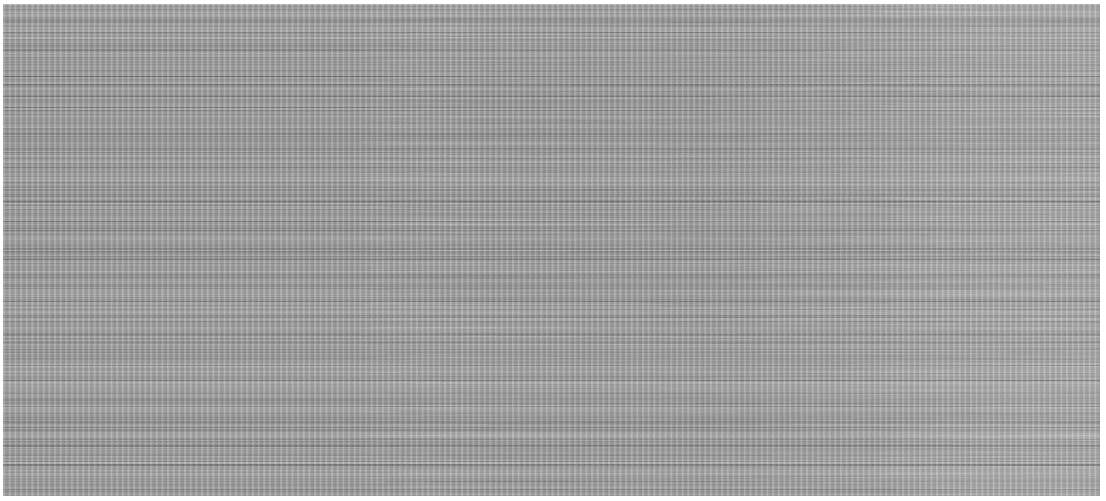
FIGURE A.9: Pixel plots for the *learn-repeated* approach



(a) Pixel-plot showing the time spent learning for the *learn-fragment* approach.



(b) Pixel-plot showing the time spent deliberating for the *learn-fragment* approach.



(c) Pixel-plot showing the time spent acting for the *learn-fragment* approach.

FIGURE A.10: Pixel plots for the *learn-fragment* approach

A.2 Experiment 2: Ambulance Agents

In this section we now present Experiment 2, which focuses on providing results to analyse the performance of the ambulance agents within the RoboCup OWLRescue framework using our reactive learning algorithm. We aim to show that our approach benefits the ambulance agents, so that they can perform better than the benchmark approaches. We use the experimental setup in Table A.1. From the results, it shows that the ambulance agents using our approach send fewer messages per attempted task, compared to the agents using other benchmark approaches (see Figure A.15 and Table A.7). This is because the ambulance agents using our approach learn more than one concept at a time. Agents using our approach complete the greatest number of tasks with the smallest ontology, compared with the other learning approaches (see Figure A.14 and Table A.6). Also, agents that use our approach spend the least time learning and deliberating, and more time acting compared with the other approaches. In terms of RoboCup Rescue, agents using our approach outperform the other approaches that do not have perfect foresight by saving on average more civilians (see Figure A.11) even though the fires spread faster because they are not controlled by the fire brigade agents.

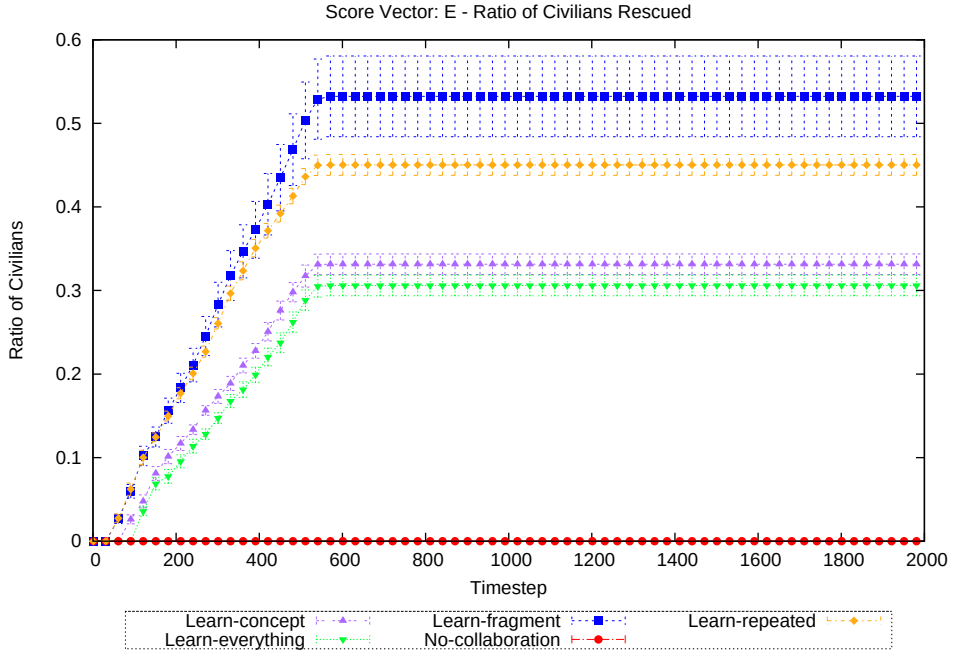


FIGURE A.11: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

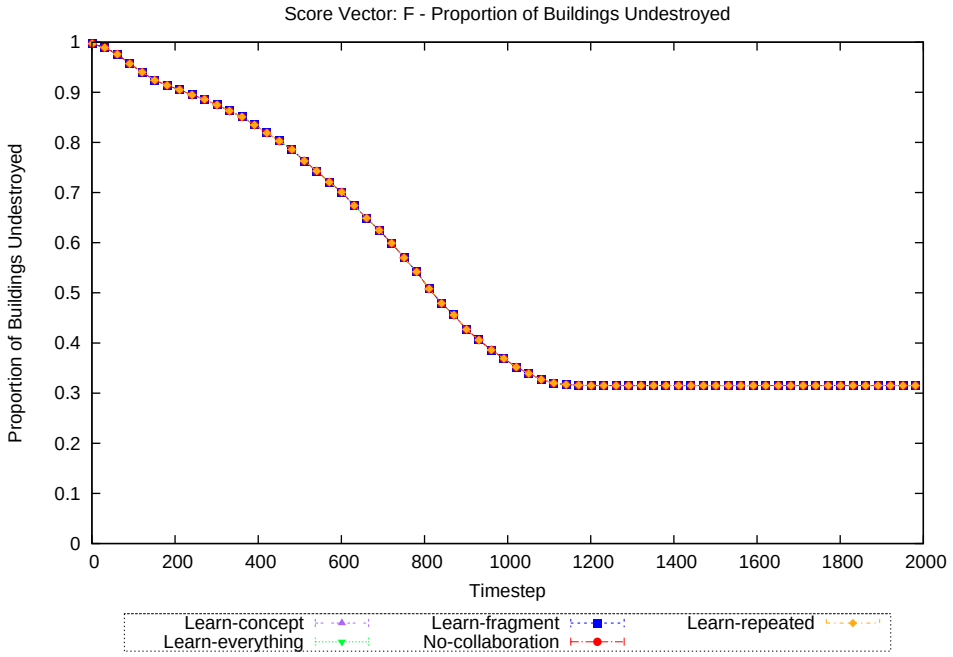


FIGURE A.12: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

Technique	Average final number
Learn-everything	35.21
Learn-repeated	51.79
Learn-concept	38.12
Learn-fragment	63.06

TABLE A.5: Statistics for the average number of attempted tasks an agent performs.

Approach	Average final value	Average minimum value	Average maximum value
Learn-everything	34.79	27	40
Learn-repeated	51.68	47	58
Learn-concept	37.91	28	43
Learn-fragment	62.42	59	67

TABLE A.6: Statistics for the average number of successful tasks an agent performs.

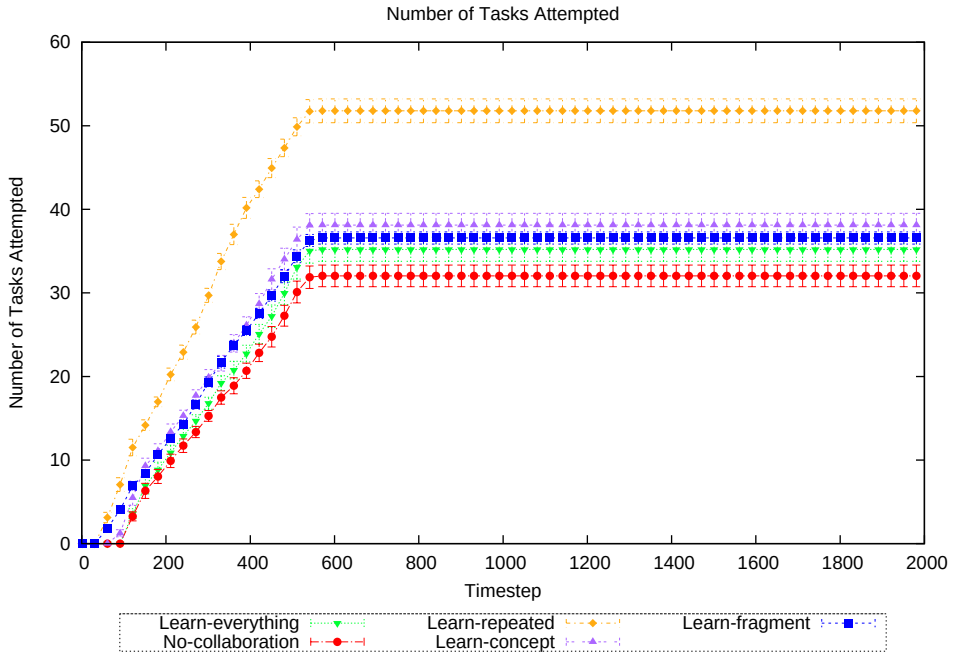


FIGURE A.13: A graph showing the average number of tasks attempted per agent.

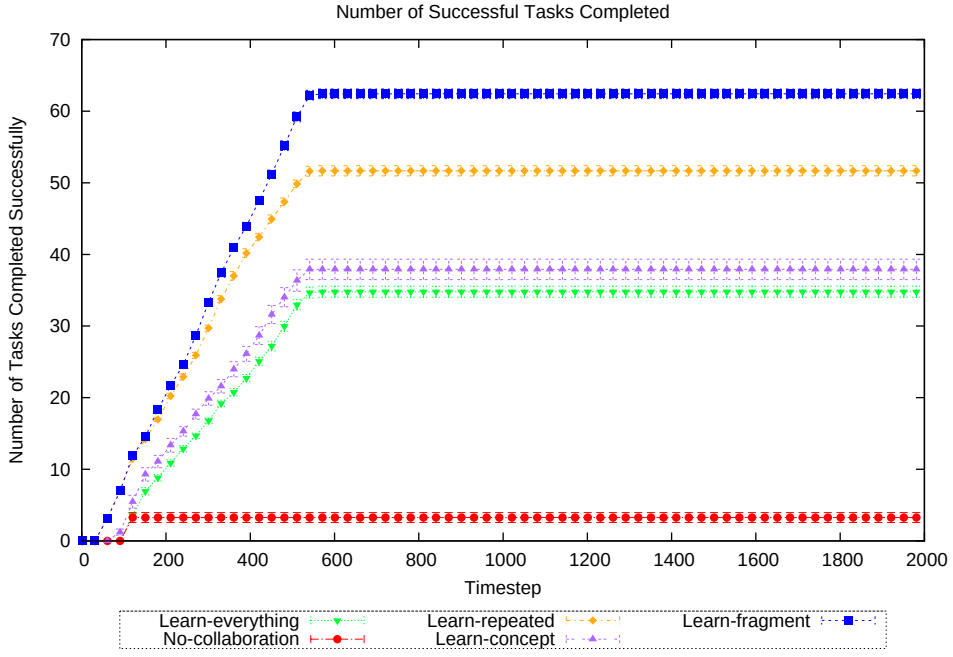


FIGURE A.14: A graph showing the average number of successful completed tasks per agent.

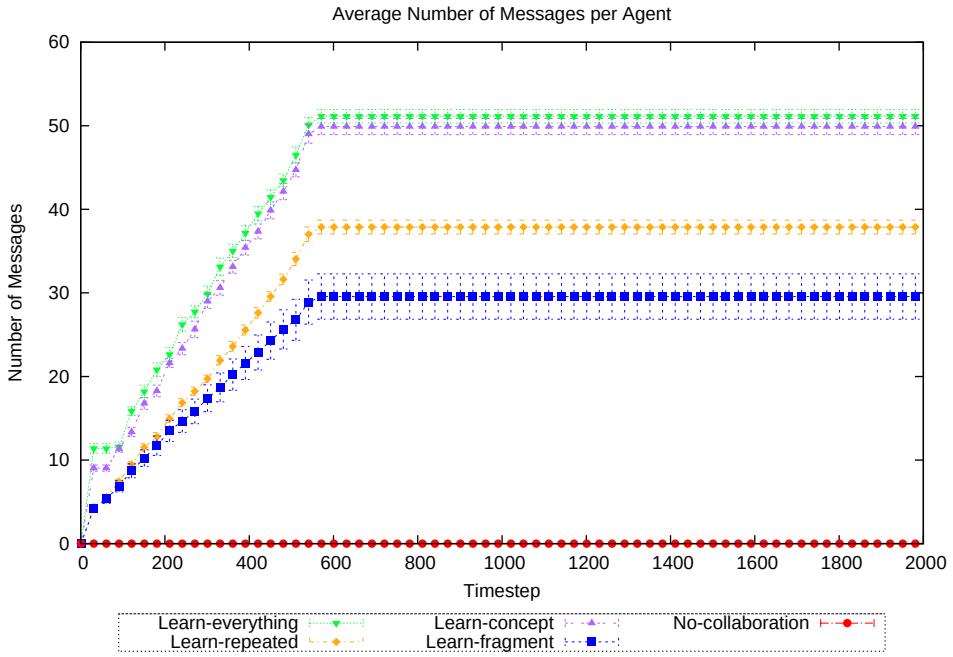


FIGURE A.15: A graph showing the average number messages sent per agent.

Approach	Average number of timesteps with no message sent	Average final value
Learn-everything	3	51.15
Learn-repeated	3	37.88
Learn-concept	3	49.91
Learn-fragment	61.74	29.56
No-collaboration	2000	

TABLE A.7: Statistics for the average number of messages sent by an agent.

Approach	Average final value	Average minimum value	Average maximum value
Learn-everything	1382.38	1282	1465
Learn-repeated	1023.82	947	1104
Learn-concept	674.5	624	730
Learn-fragment	823.18	739	896
No-collaboration	0	0	0

TABLE A.8: Statistics for the average number of concepts in an agent's ontology.

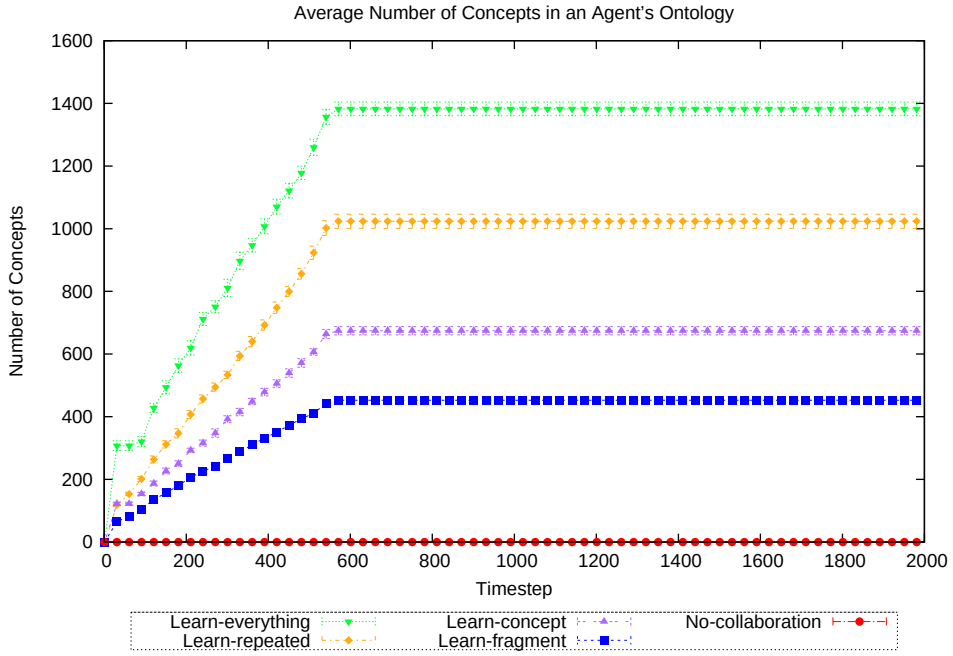
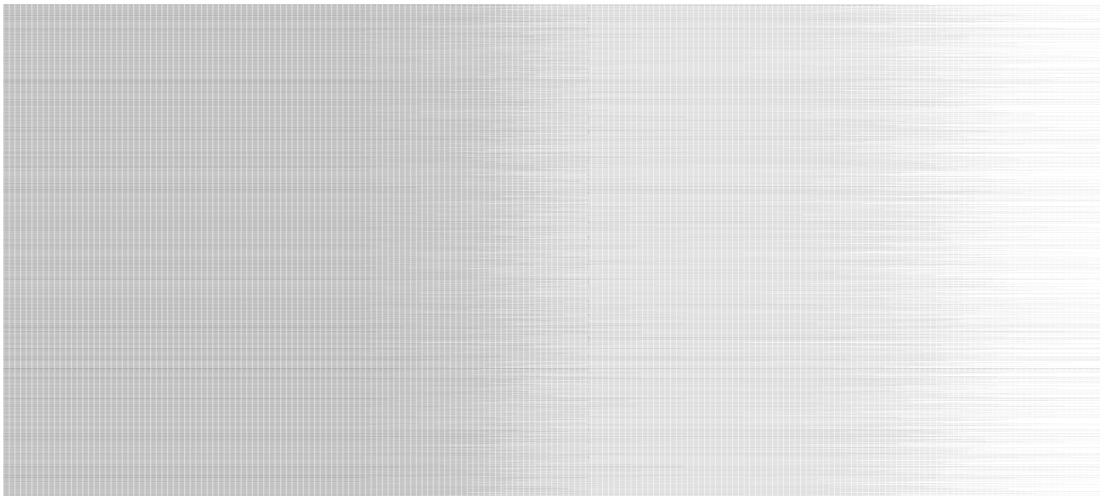


FIGURE A.16: A graph showing the average number of concepts in an agent's ontology.

(a) Pixel-plot showing the time spent learning for the *learn-everything* approach.

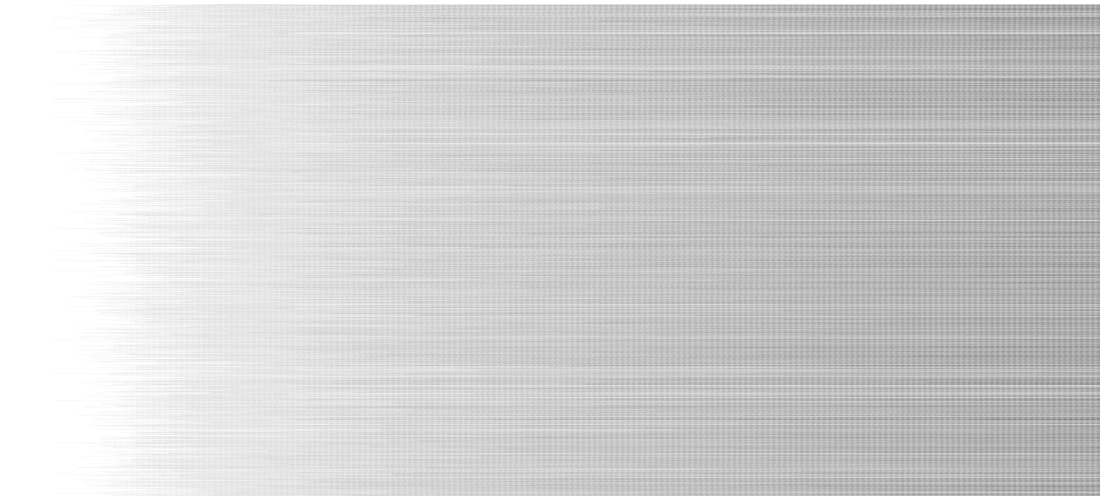
(b) Pixel-plot showing the time spent deliberating for the *learn-everything* approach.



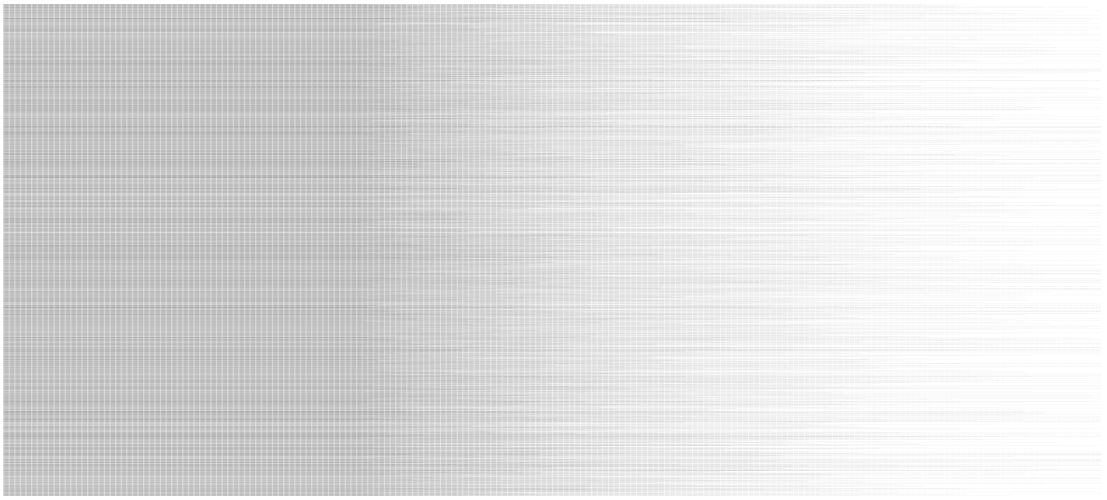
(c) Pixel-plot showing the time spent acting for the *learn-everything* approach.

FIGURE A.17: Pixel plots for the *learn-everything* approach

(a) Pixel-plot showing the time spent learning for the *learn-concept* approach.

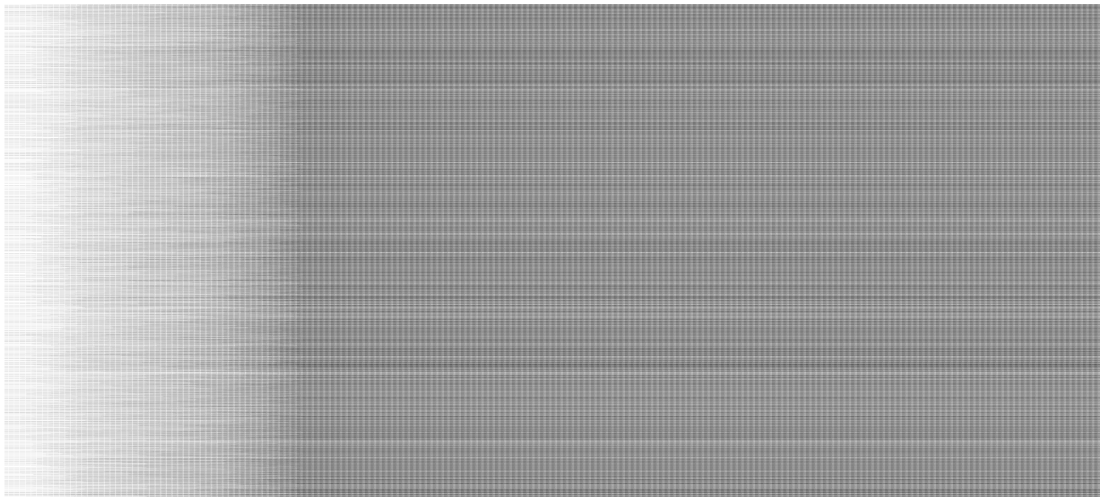


(b) Pixel-plot showing the time spent deliberating for the *learn-concept* approach.

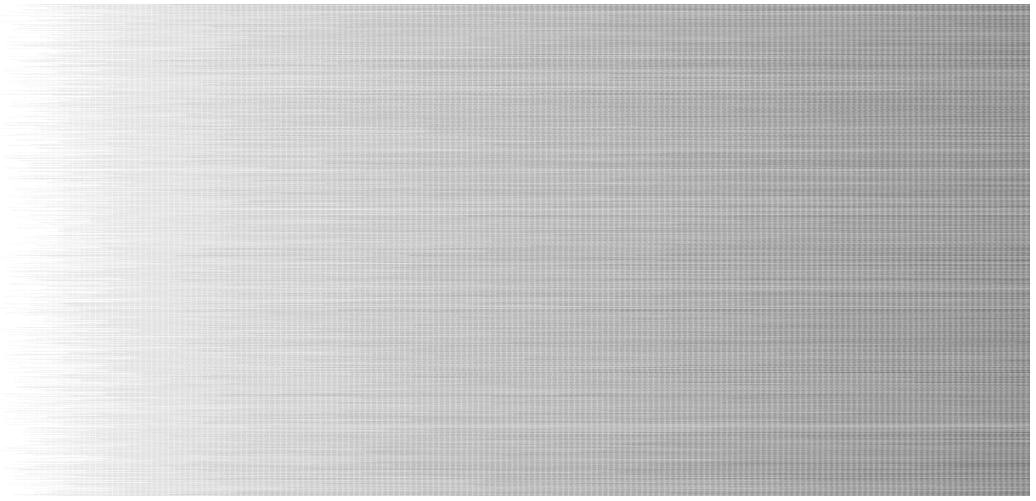


(c) Pixel-plot showing the time spent acting for the *learn-concept* approach.

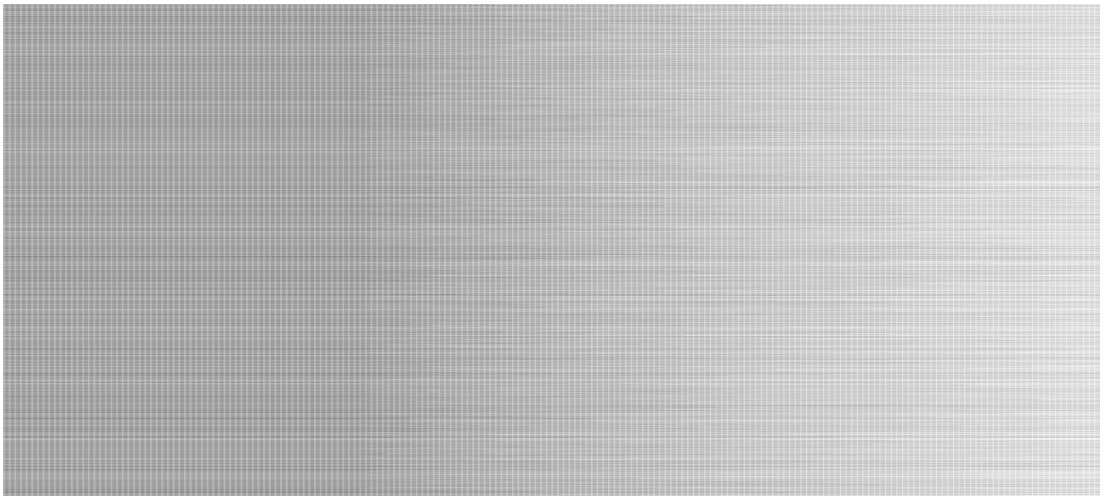
FIGURE A.18: Pixel plots for the *learn-concept* approach



(a) Pixel-plot showing the time spent learning for the *learn-repeated* approach.



(b) Pixel-plot showing the time spent deliberating for the *learn-repeated* approach.

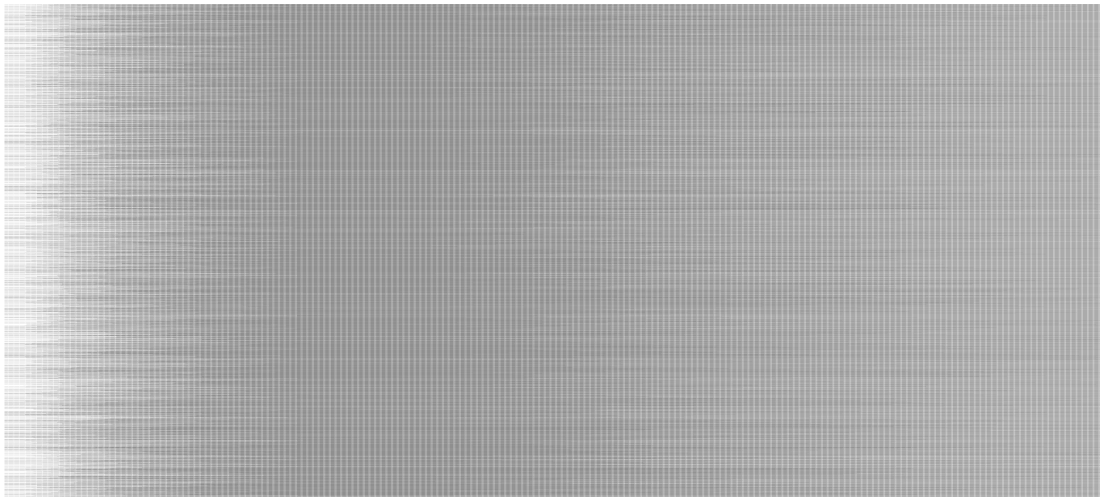


(c) Pixel-plot showing the time spent acting for the *learn-repeated* approach.

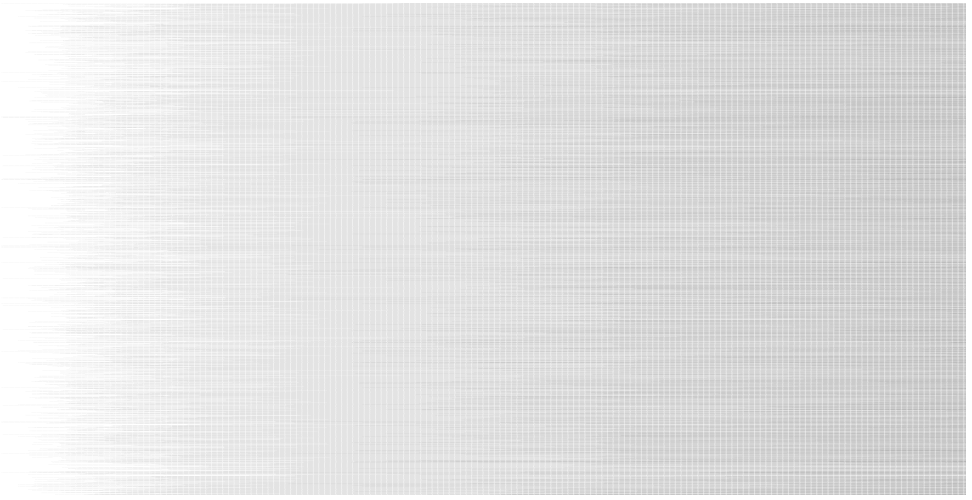
FIGURE A.19: Pixel plots for the *learn-repeated* approach

A.3 Summary

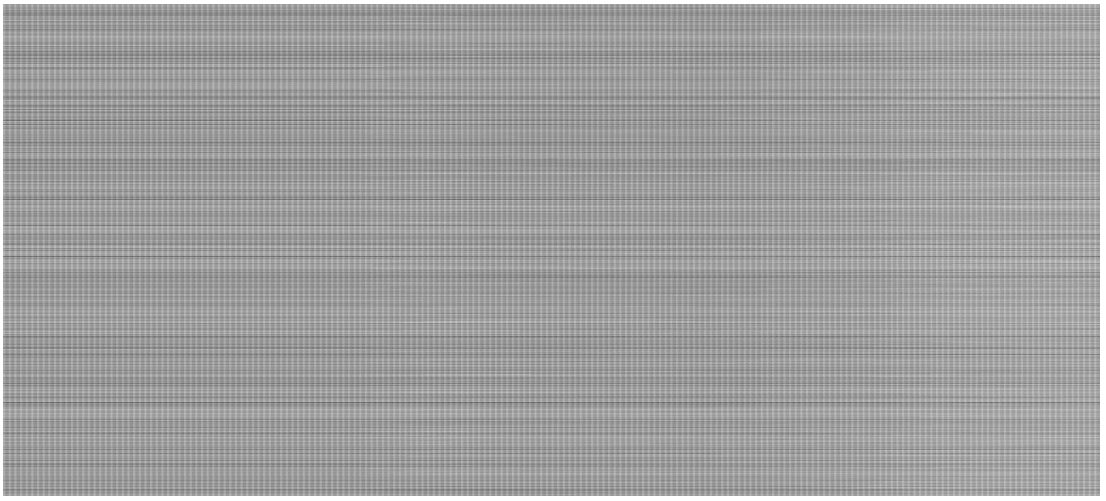
In summary, both the results from Experiment 1 and 2 show that both the fire brigade and ambulance agents benefit from using our reactive learning approach. In comparison with our Experiment 3 where we can see how all of the three types of agent perform together, Experiments 1 and 2 show how they affect them individually. While both the fire brigade and ambulance agents using the comparative approaches exhibit the same trends, the fire brigade is more effective because its actions prevent the fire from spending thus reducing the number of civilian casualties.



(a) Pixel-plot showing the time spent learning for the *learn-fragment* approach.



(b) Pixel-plot showing the time spent deliberating for the *learn-fragment* approach.



(c) Pixel-plot showing the time spent acting for the *learn-fragment* approach.

FIGURE A.20: Pixel plots for the *learn-fragment* approach

Appendix B

Additional Results from our Evaluation of the Proactive Learning Algorithm

In this appendix, we present additional results from our proactive learning experiments, Experiments 1 and 2 (mentioned in Chapter 5). The parameters used for both of these experiments are listed in Table B.1.

	Experiment 1	**Experiment 2**
Environment ontologies	10	10
Number of agents	20	20
Number of fire brigades	10	0
Number of police	10	10
Number of ambulance	0	10
Number of time steps	2000	2000
Map	Kobe	Kobe
Benchmark approaches	Learn-repeated, Learn-fragment, Learn-everything, No-collaboration	Learn-repeated, Learn-fragment, Learn-everything, No-collaboration
Number of civilians	115	115
Iterations	250	250

Table B.1: The parameters for Experiments 1 and 2.

The next two sections presents the results from Experiment 1 and 2, respectively.

B.1 Experiment 1: Fire Brigade Agents

In this section we present the results of Experiment 2, which focuses on the performance of the fire brigade agents in the RoboCup OWLRescue framework using our proactive learning approach. The purpose of this experiment is to show how using a Markov model to predict concepts that will be used in the future can benefit the fire brigade agents, so that they can perform better than the benchmark approaches. We use the experimental setup in Table B.1. From the results, it shows using a Markov model enabled the agents to outperform our reactive learning approach because it: sent fewer messages per attempted task, compared to the agents using other benchmark approaches (see Figure B.6 and Table B.2); completed the greatest number of tasks with the smallest ontology compared with the other learning approaches (see Figure B.5 and Table B.4); spend the least time learning and deliberating, and more time acting compared with the other approaches; saved on average more civilians (see Figure B.2) and more of the city (see Figure B.3).

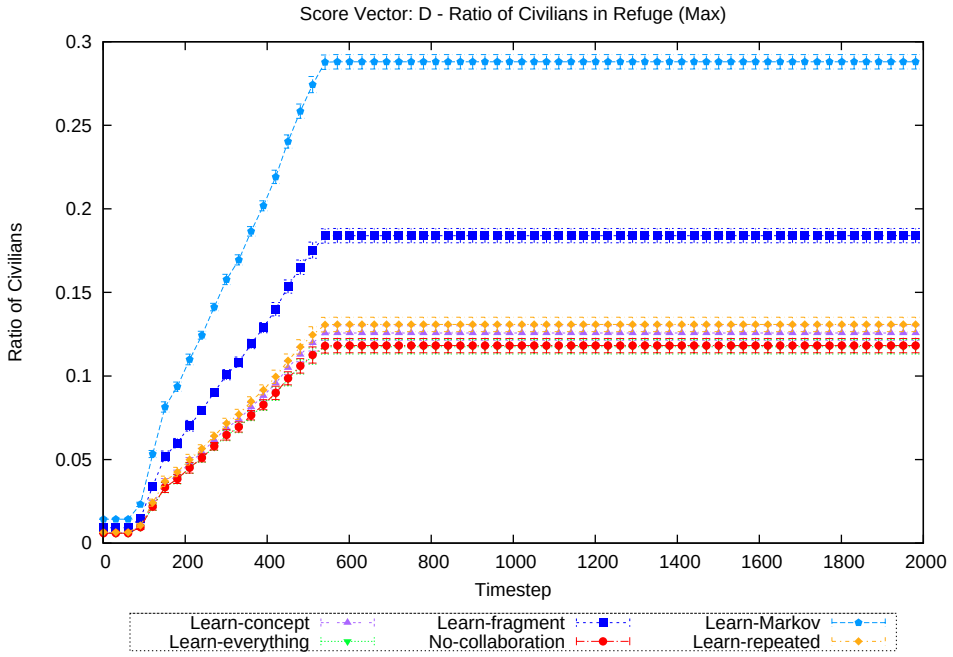


FIGURE B.1: A graph showing the RCR score vector D, showing the average ratio of civilians in the refuge.

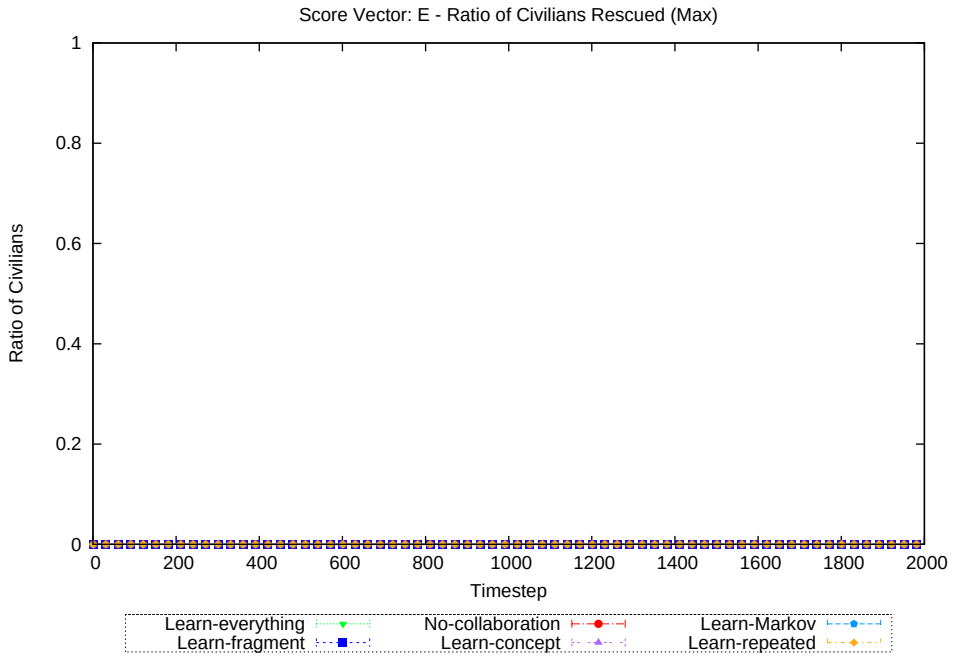


FIGURE B.2: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

Technique	Average final value
Learn-everything	94.37
Learn-repeated	106.82
Learn-concept	232.00
Learn-fragment	64.44
No-collaboration	0
Learn-Markov	38.34

TABLE B.2: Statistics for the number of messages sent per agent.

Technique	Average final value
Learn-everything	2550.47
Learn-repeated	2887.09
Learn-concept	3135.41
Learn-fragment	2087.90
No-collaboration	0
Learn-Markov	1036.30

TABLE B.3: Statistics for the number of concepts in each agent's ontology.

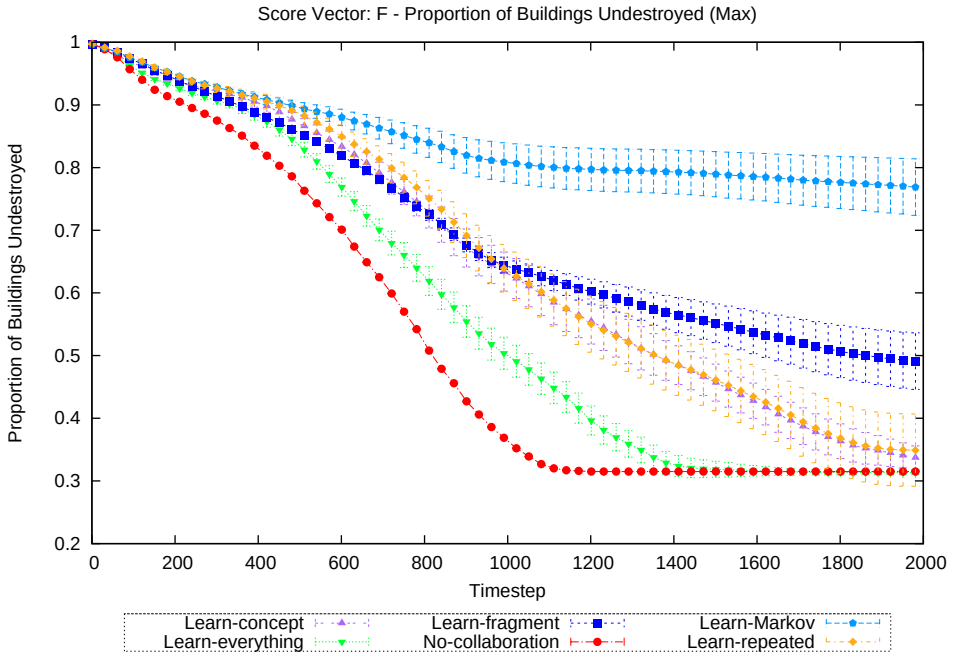


FIGURE B.3: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

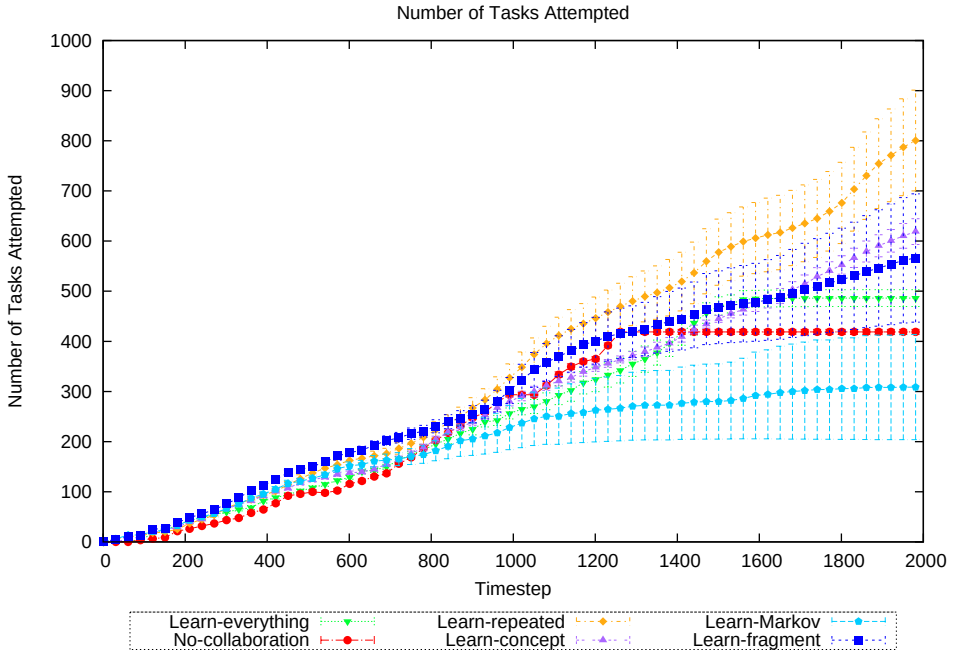


FIGURE B.4: A graph showing the average number of tasks attempted per agent.

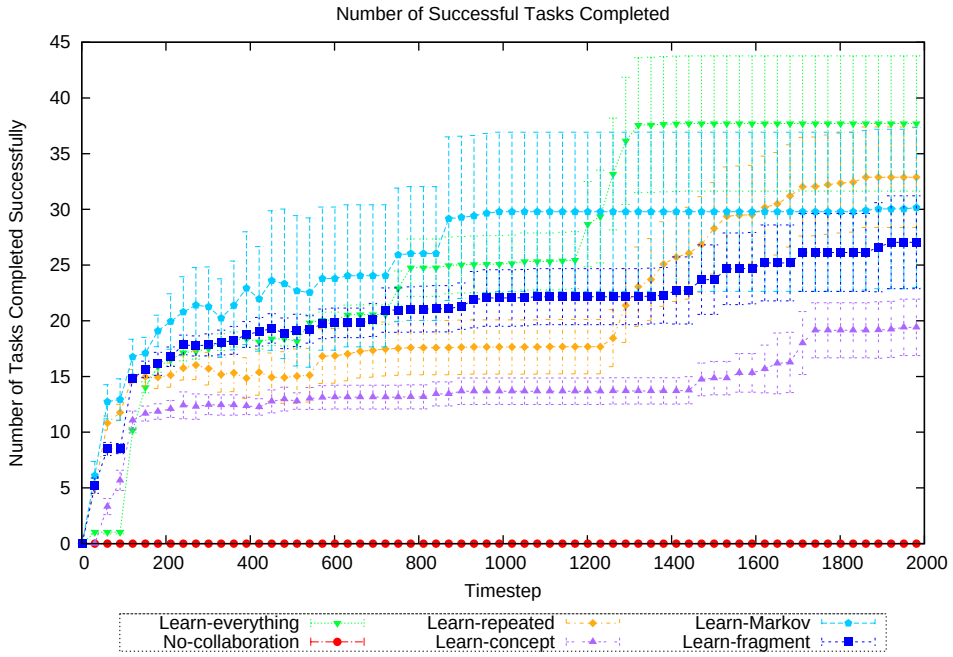


FIGURE B.5: A graph showing the average number of successfully completed tasks per agent.

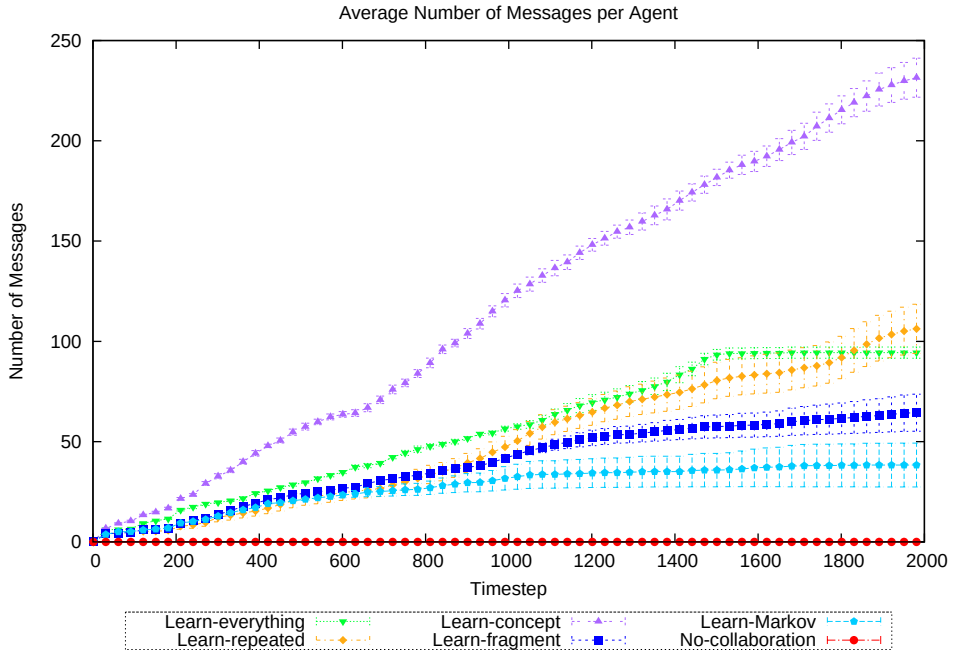
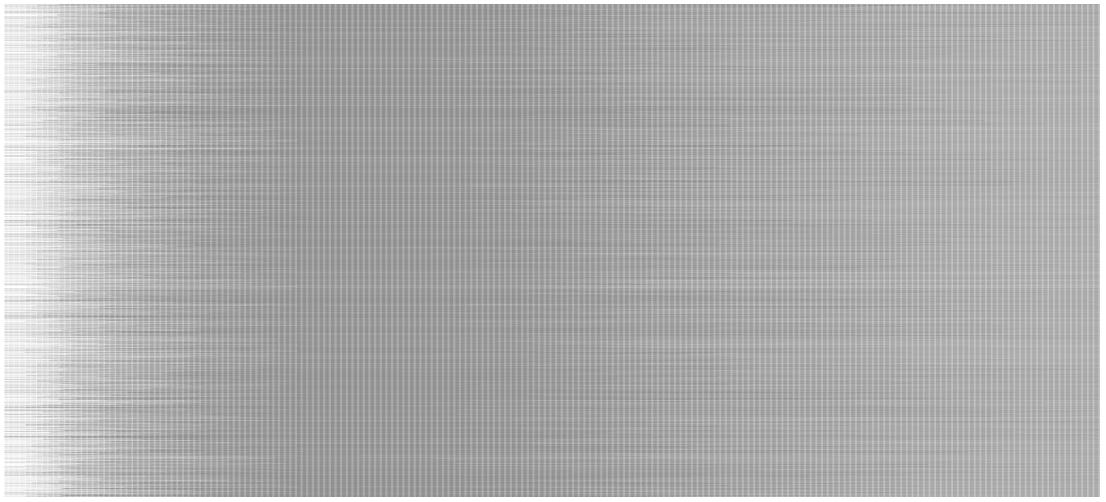
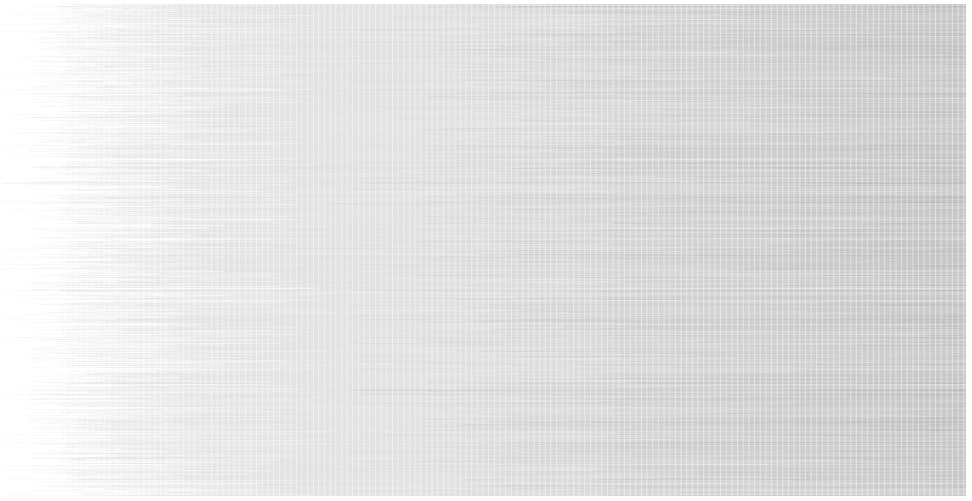


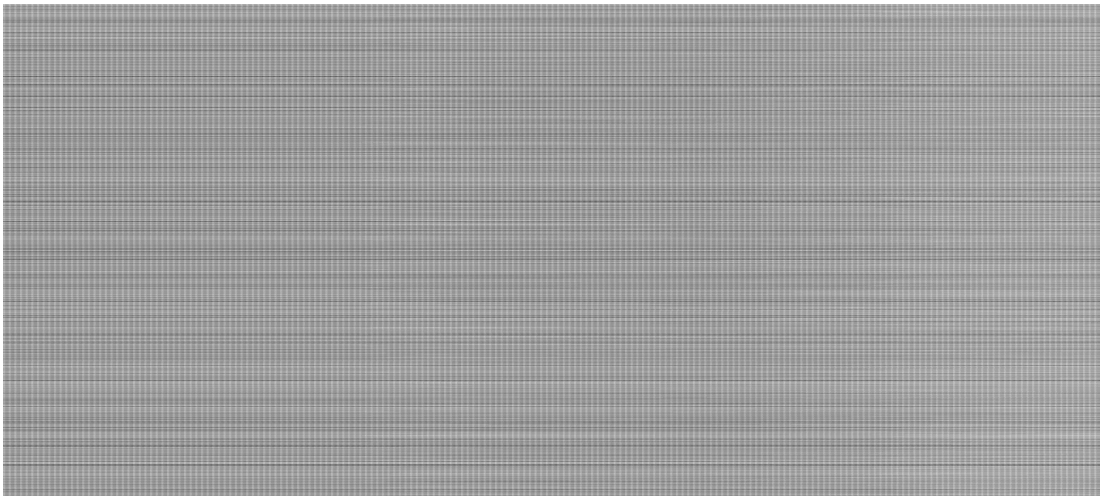
FIGURE B.6: A graph showing the average number of messages send per agent.



(a) Pixel-plot showing the time spent learning for the *learn-fragment* approach.



(b) Pixel-plot showing the time spent deliberating for the *learn-fragment* approach.

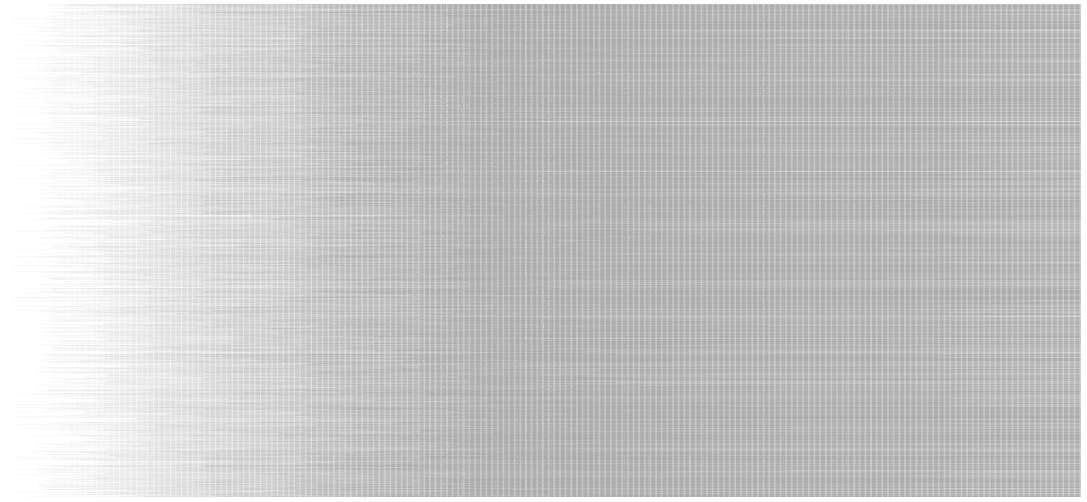


(c) Pixel-plot showing the time spent acting for the *learn-fragment approach.*

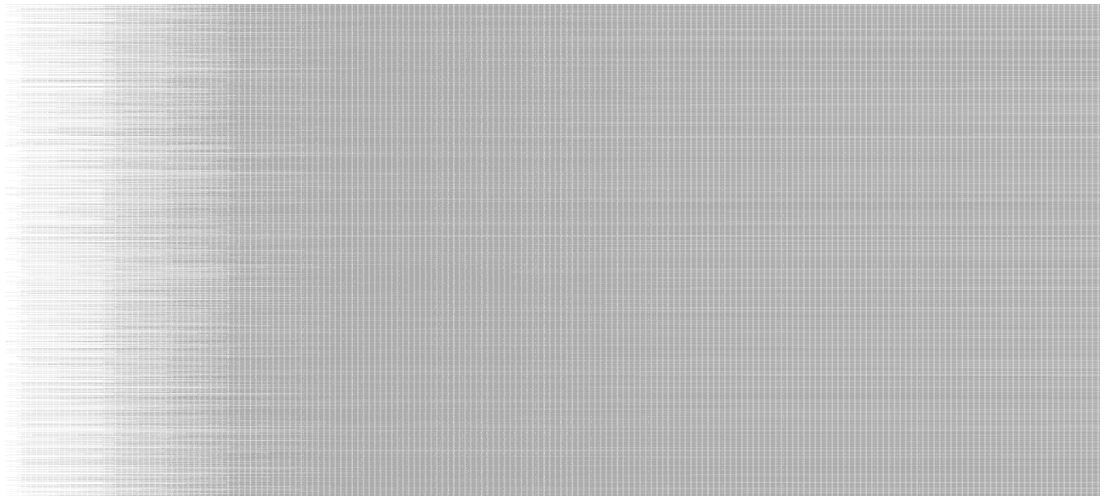
FIGURE B.7: Pixel plots for the *learn-fragment* approach

Technique	**Average final value**
Learn-everything	37.71
Learn-repeated	32.88
Learn-concept	19.41
Learn-fragment	27.05
No-collaboration	0
Learn-Markov	30.16

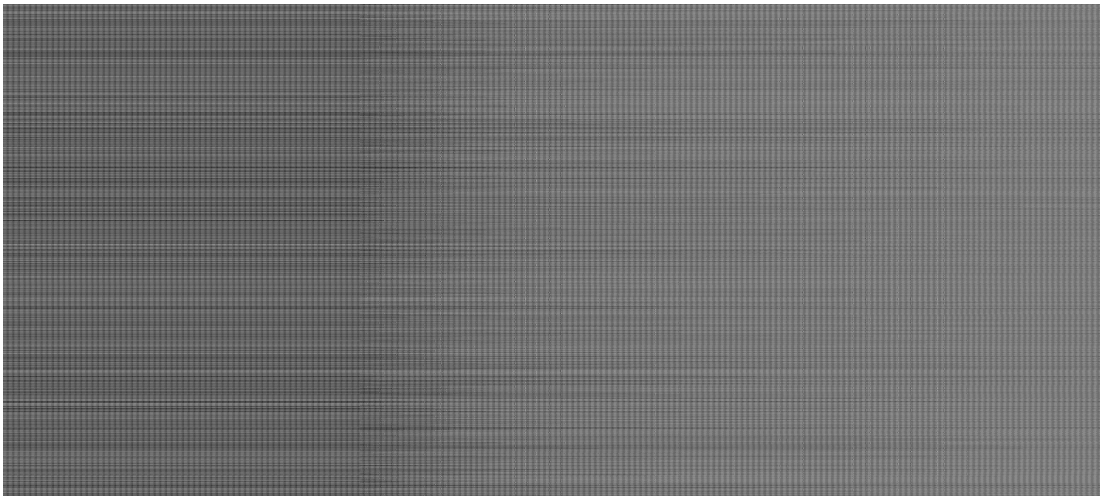
TABLE B.4: Statistics for the average number of successful tasks an agent performs.



(a) Pixel-plot showing the time spent learning for the *learn-Markov approach.*

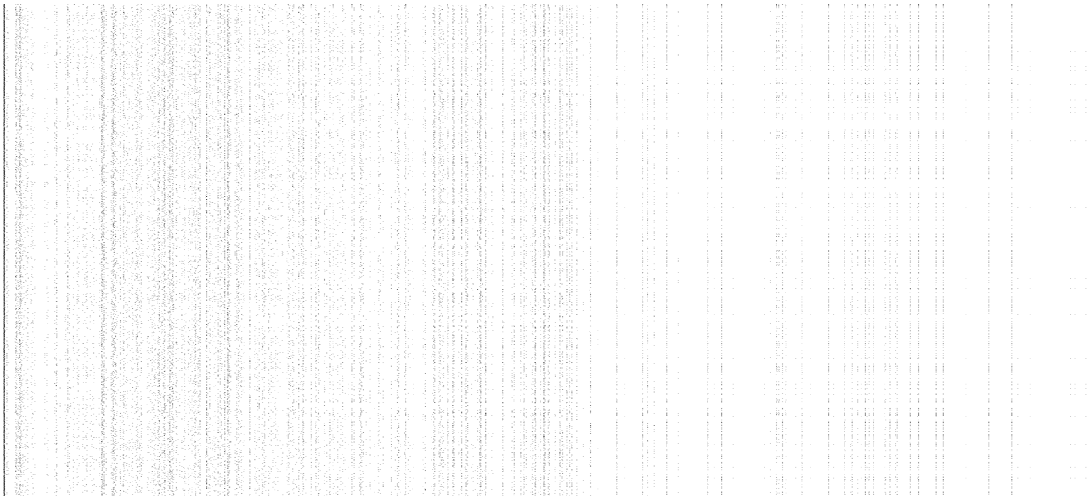


(b) Pixel-plot showing the time spent deliberating for the *learn-Markov* approach.



(c) Pixel-plot showing the time spent acting for the *learn-Markov* approach.

FIGURE B.8: Pixel plots for the *learn-Markov* approach



(d) Pixel-plot showing the time spent using the Markov model for the *learn-Markov* approach.

FIGURE B.8: Pixel plots for the *learn-Markov* approach

B.2 Experiment 2: Ambulance Agents

In this section we now present Experiment 2, which focuses on the performance of the ambulance agents within the RoboCup OWLRescue framework. We aim to show that our approach benefits the ambulance agents, so that they can perform better than the benchmark approaches. We use the experimental setup in Table B.1. From the results, it shows using a Markov model enabled the agents to outperform our reactive learning approach because it: sent fewer messages per attempted task, compared to the agents using other benchmark approaches (see Figure B.13 and Table B.7); completed the greatest number of tasks with the smallest ontology, compared with the other learning approaches (see Figure B.12 and Table B.6); spend the least time learning and deliberating, and more time acting compared with the other approaches; saved on average more civilians (see Figure B.9) and more of the city (see Figure B.10).

TABLE B.5: Statistics for the average number of attempted tasks an agent performs.

Technique	**Average final number**
Learn-everything	35.21
Learn-repeated	51.79
Learn-concept	38.12
Learn-fragment	63.06
Learn-Markov	96.32

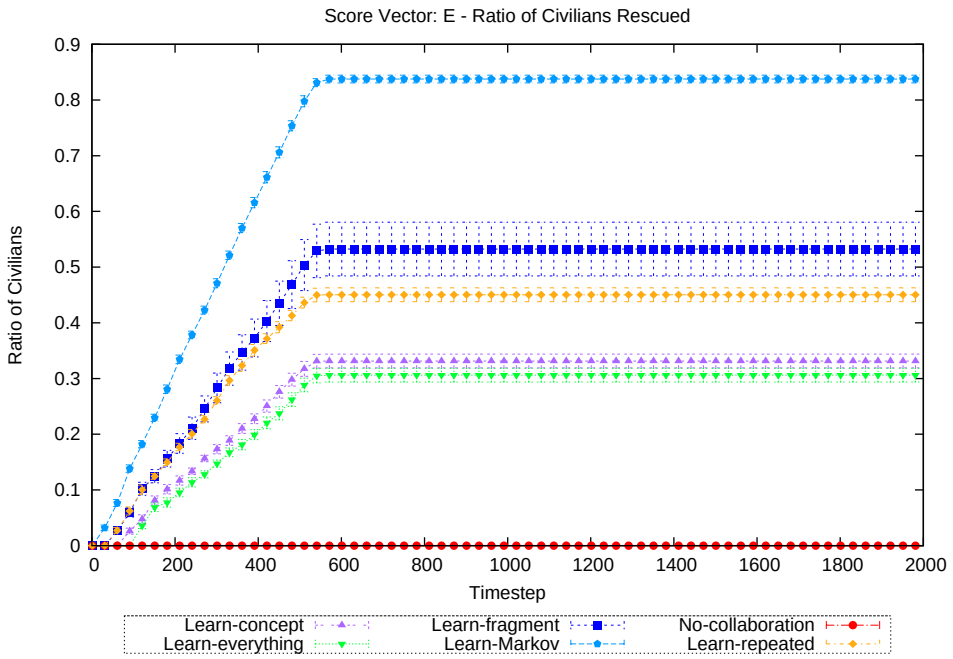


FIGURE B.9: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

TABLE B.6: Statistics for the average number of successful tasks an agent performs.

Approach	Average final value	Average minimum value	Average maximum value
Learn-everything	34.79	27	40
Learn-repeated	51.68	47	58
Learn-concept	37.91	28	43
Learn-fragment	62.42	59	67
Learn-Markov	95.59	91	99

TABLE B.7: Statistics for the average number of messages sent by an agent.

Approach	Average number of timesteps with no message sent	Average final value
Learn-everything	3	51.15
Learn-repeated	3	37.88
Learn-concept	3	49.91
Learn-fragment	61.74	29.56
No-collaboration	2000	
Learn-Markov	3	6.36

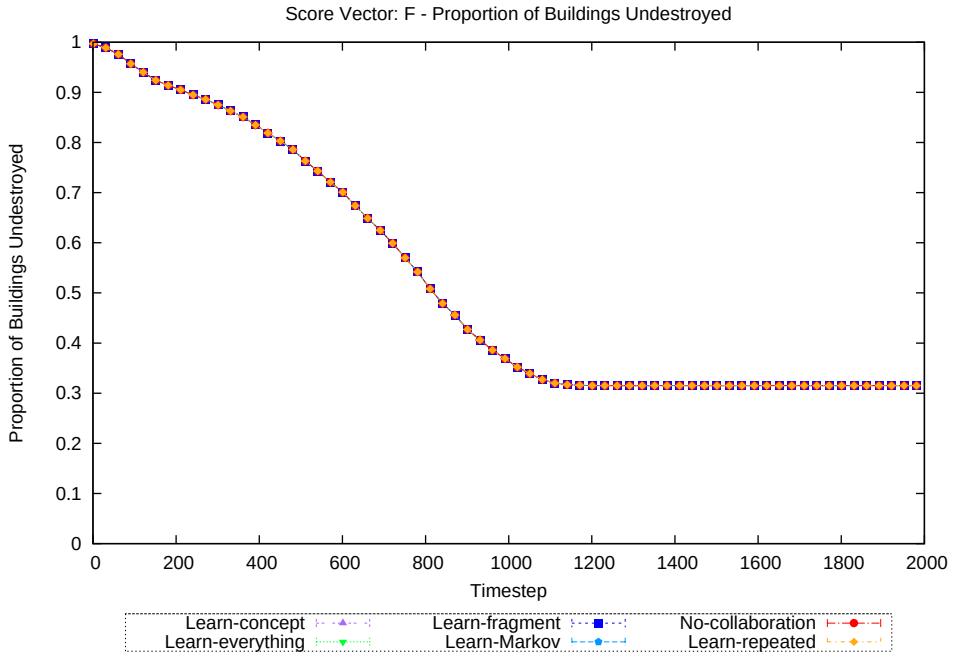


FIGURE B.10: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

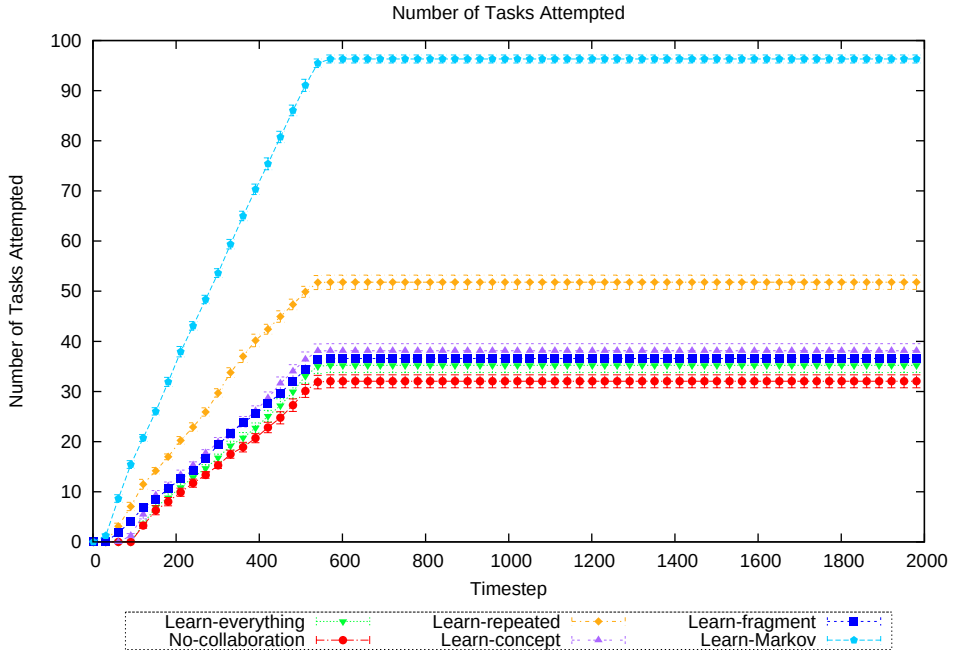


FIGURE B.11: A graph showing the average number of tasks attempted per agent.

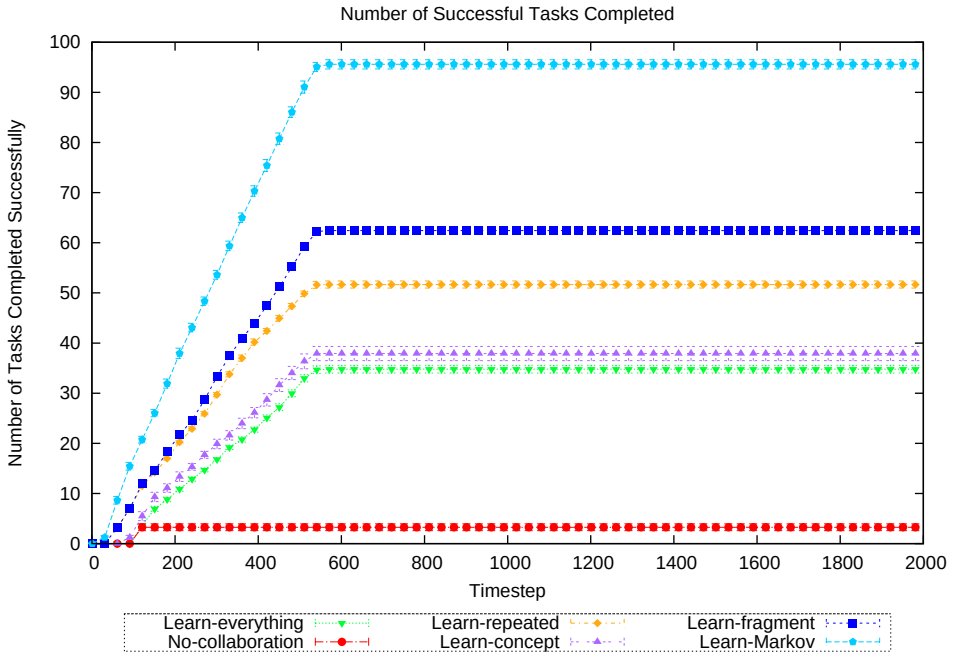


FIGURE B.12: A graph showing the average number of successfully completed tasks per agent.

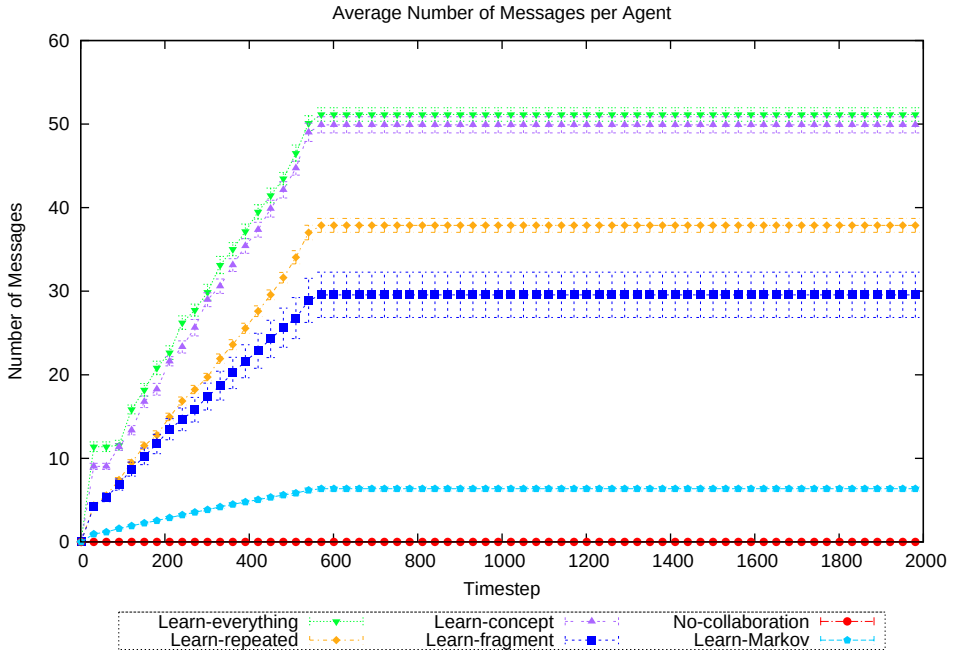


FIGURE B.13: A graph showing the average number of number of messages.

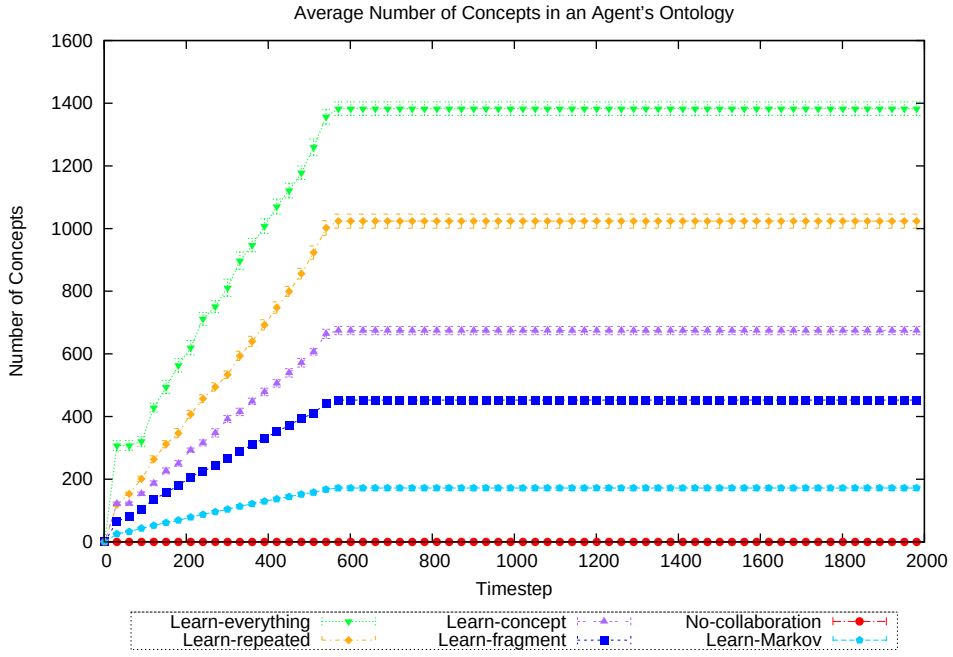
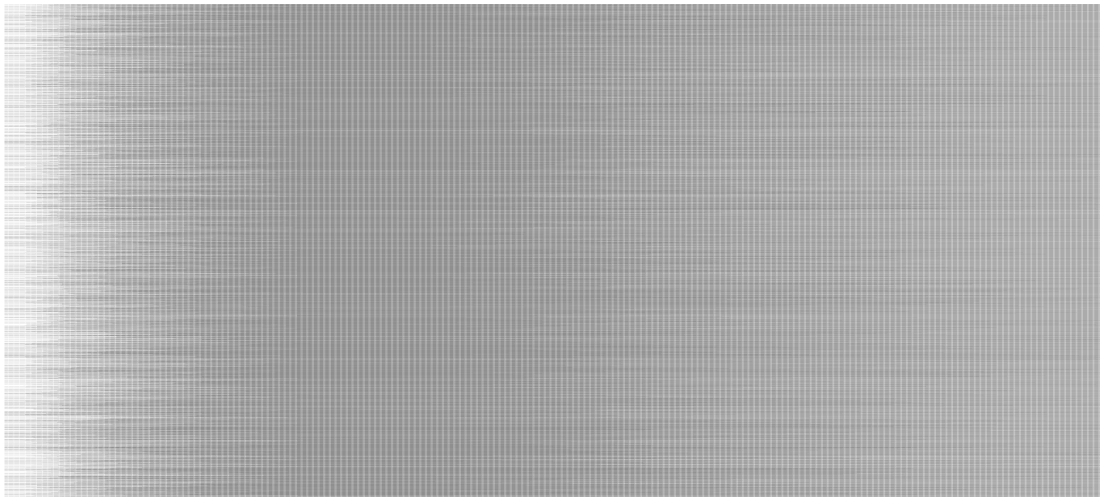


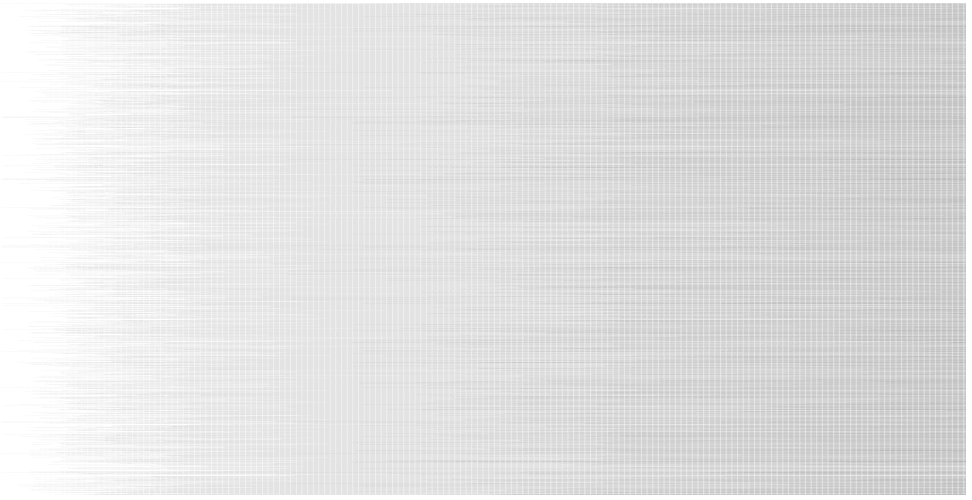
FIGURE B.14: A graph showing the average number of concepts in an agent's ontology.

Approach	**Average final value**	**Average minimum value**	**Average maximum value**
Learn-everything	1382.38	1282	1465
Learn-repeated	1023.82	947	1104
Learn-concept	674.5	624	730
Learn-fragment	823.18	739	896
No-collaboration	0	0	0
Learn-Markov	171.94	148	206

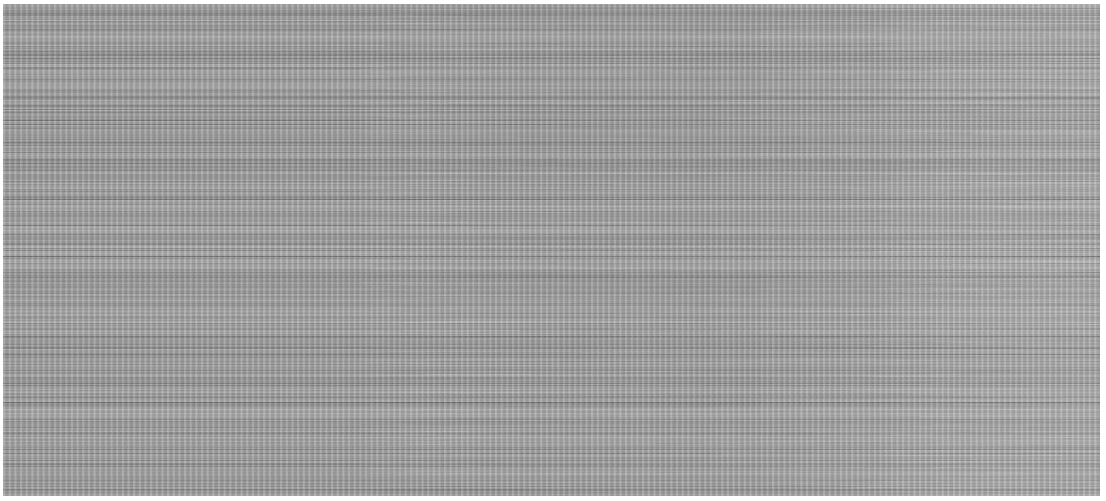
TABLE B.8: Statistics for the average number of concepts in an agent's ontology.



(a) Pixel-plot showing the time spent learning for the *learn-fragment* approach.

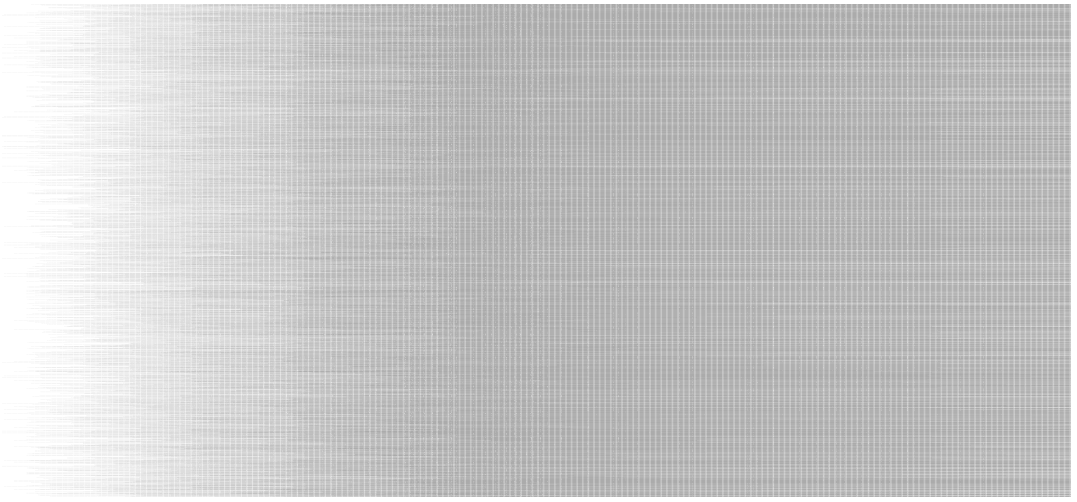


(b) Pixel-plot showing the time spent deliberating for the *learn-fragment* approach.

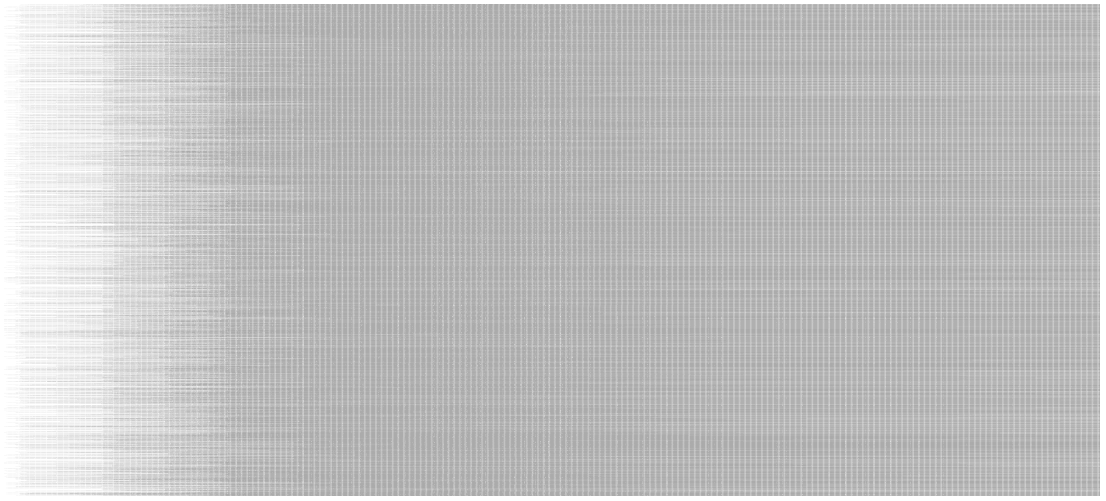


(c) Pixel-plot showing the time spent acting for the *learn-fragment approach.*

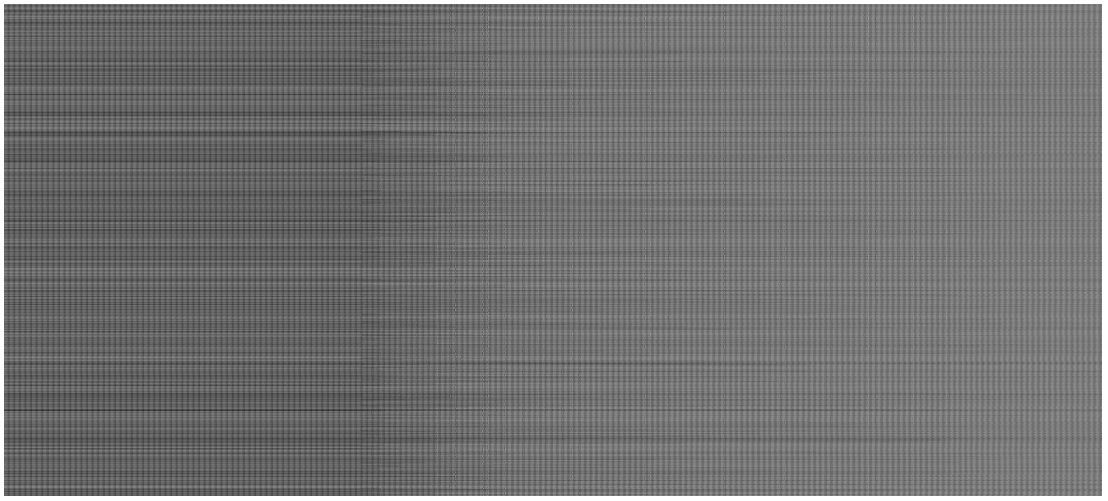
FIGURE B.15: Pixel plots for the *learn-fragment* approach



(a) Pixel-plot showing the time spent learning for the *learn-Markov approach.*

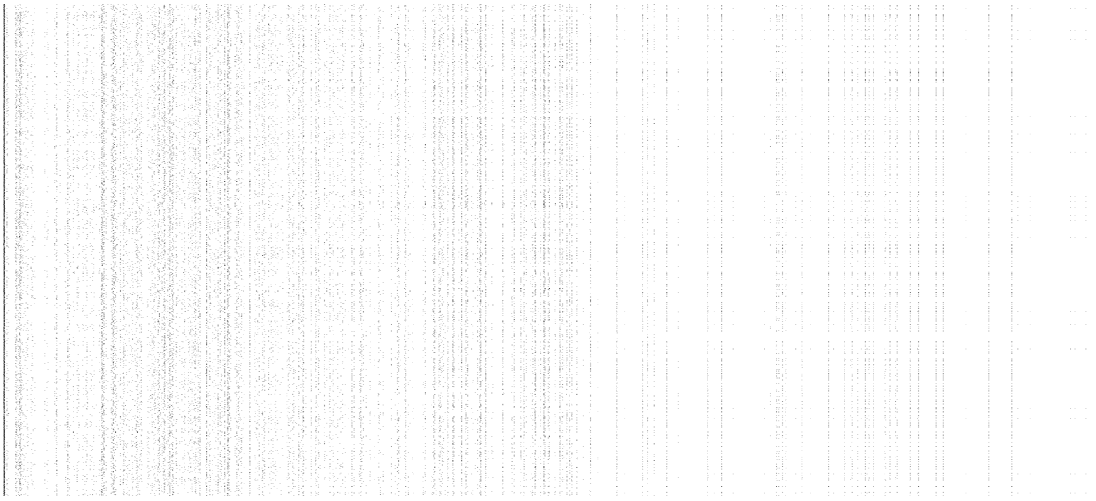


(b) Pixel-plot showing the time spent deliberating for the *learn-Markov* approach.



(c) Pixel-plot showing the time spent acting for the *learn-Markov* approach.

FIGURE B.16: Pixel plots for the *learn-Markov* approach



(d) Pixel-plot showing the time spent using the Markov model for the *learn-Markov* approach.

FIGURE B.16: Pixel plots for the *learn-Markov* approach

B.3 Summary

In summary, both the results from Experiment 1 and 2 show that both the fire brigade and ambulance agents benefit from using our proactive learning algorithm and outperforms agents using our reactive learning algorithm. While both the fire brigade and ambulance agents using the comparative approaches exhibit the same trends compared to Experiment 3 which utilises all types of agent, the fire brigade (shown in Experiment 1) is more effective than the ambulance agents (shown in Experiment 2) because its actions prevent the fire from spending thus reducing the number of civilian casualties.

Appendix C

Additional Results from our Evaluation of the Forgetting Algorithm

In this appendix, we present additional results from our forgetting experiments, Experiments 1 and 2 (mentioned in Chapter 6). The parameters used for both of these experiments are listed in Table C.1.

	Experiment 1	**Experiment 2**
Environment ontologies	10	10
Number of agents	20	20
Number of fire brigades	10	0
Number of police	10	10
Number of ambulance	0	10
Number of time steps	2000	2000
Map	Kobe	Kobe
Benchmark approaches	Forget-concept, Forget-tree, Forget-redundant, Forget-nothing, No-collaboration	Forget-concept, Forget-tree, Forget-redundant, Forget-nothing, No-collaboration
Number of civilians	115	115
EO capacity	500	500
Iterations	250	250

Table C.1: The parameters for Experiments 1 and 2.

The next two sections presents the results from Experiment 1 and 2, respectively.

C.1 Experiment 1: Fire Brigade Agents

In this section we present the results of Experiment 1, which focuses on the performance of the ambulance agents within the RoboCup OWLRescue framework. We aim to show that our approach benefits the ambulance agents, so that they can perform better than the benchmark approaches. We use the experimental setup in Table C.1. From the results, we show that the fire brigade agents using our approach outperforms the other approaches that do not have perfect foresight, in terms of RoboCup Rescue, by saving on average more civilians (see Figure C.1) and more of the city (see Figure C.2). This is because our approach removes more than one concept at a time from an agent's ontology and thus spends the least total time forgetting (see Figure C.8). They were also able to complete the greatest number of tasks (see Figure C.4 and Table C.3), while maintaining an optimal sized ontology with the lowest estimated complexity (see Figure C.6).

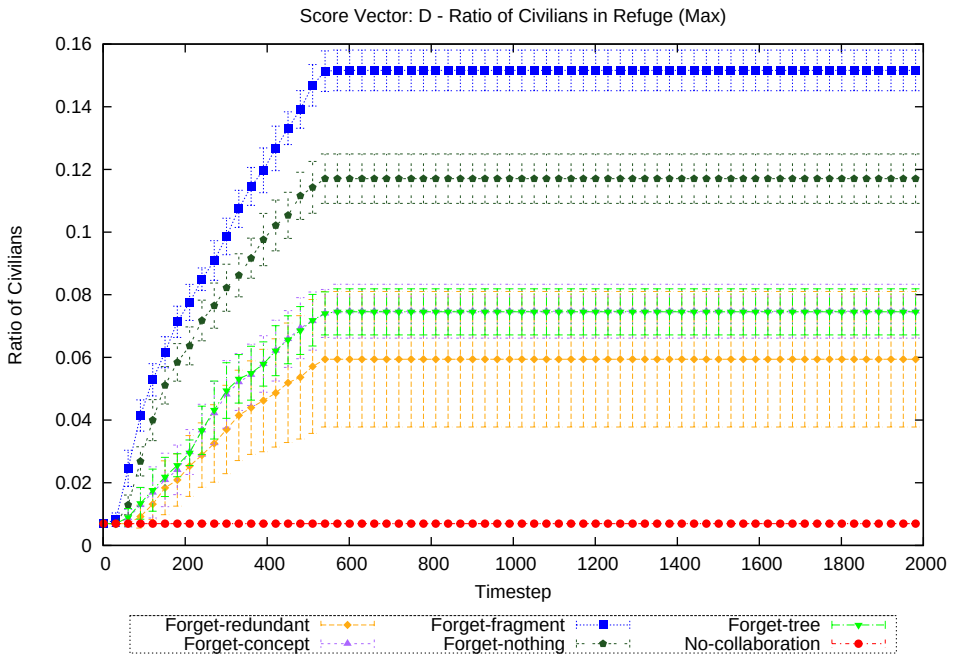


FIGURE C.1: A graph showing the RCR score vector D, showing the average ratio of civilians in the refuge.

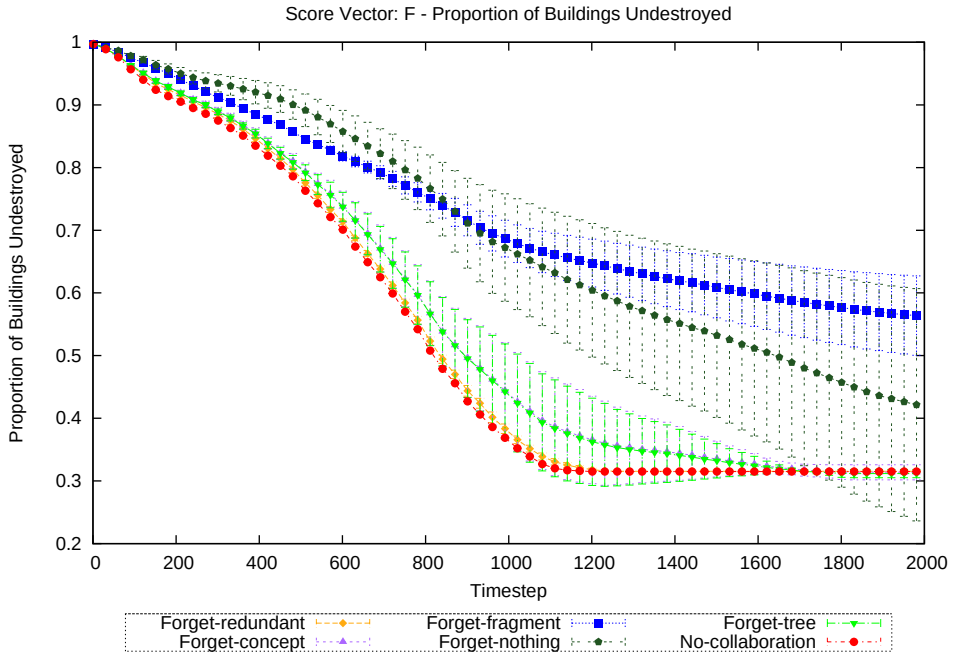


FIGURE C.2: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

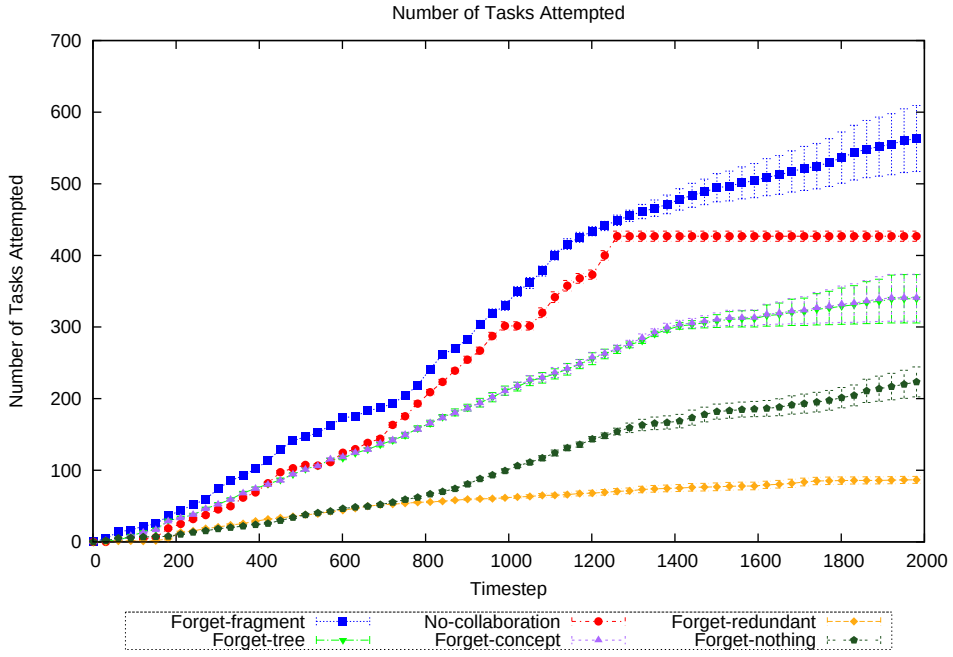


FIGURE C.3: A graph showing the average number of tasks attempted per agent.

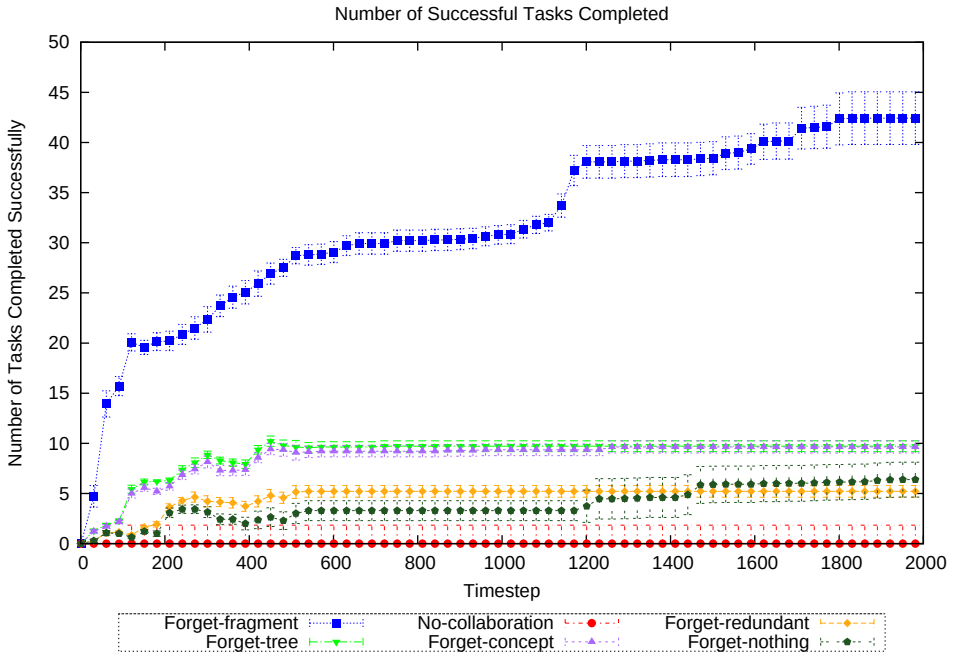


FIGURE C.4: A graph showing the average number of successfully completed tasks per agent.

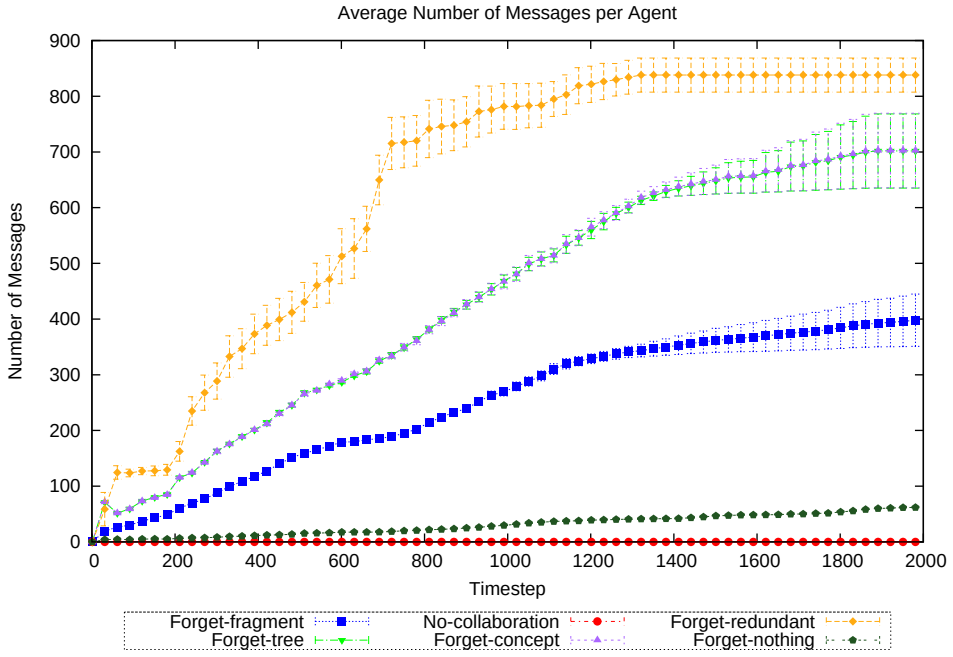


FIGURE C.5: A graph showing the average number of messages sent per agent.

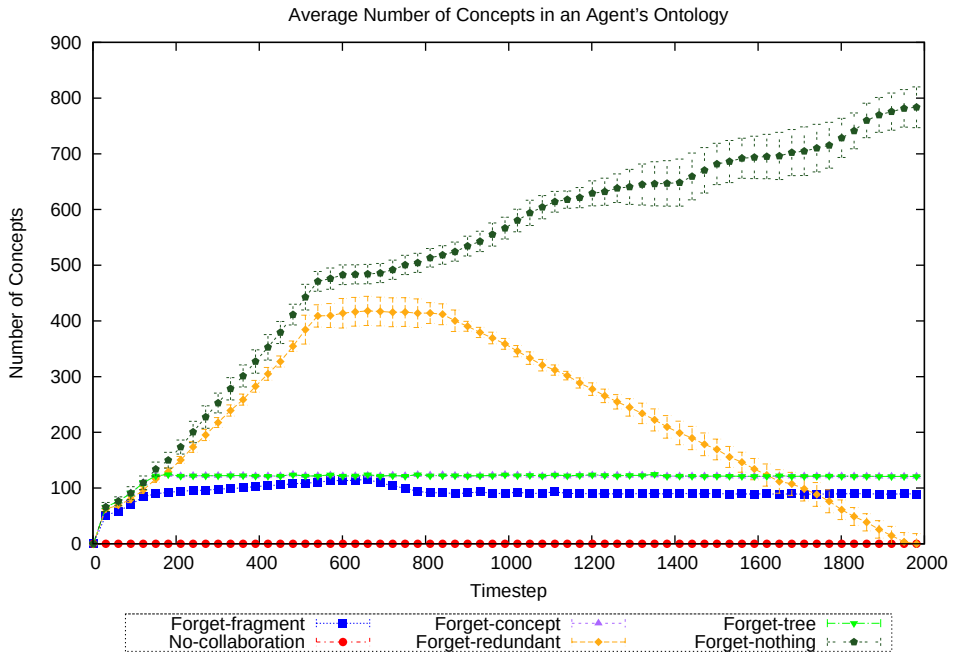


FIGURE C.6: A graph showing the average number of concepts in an agent's ontology.

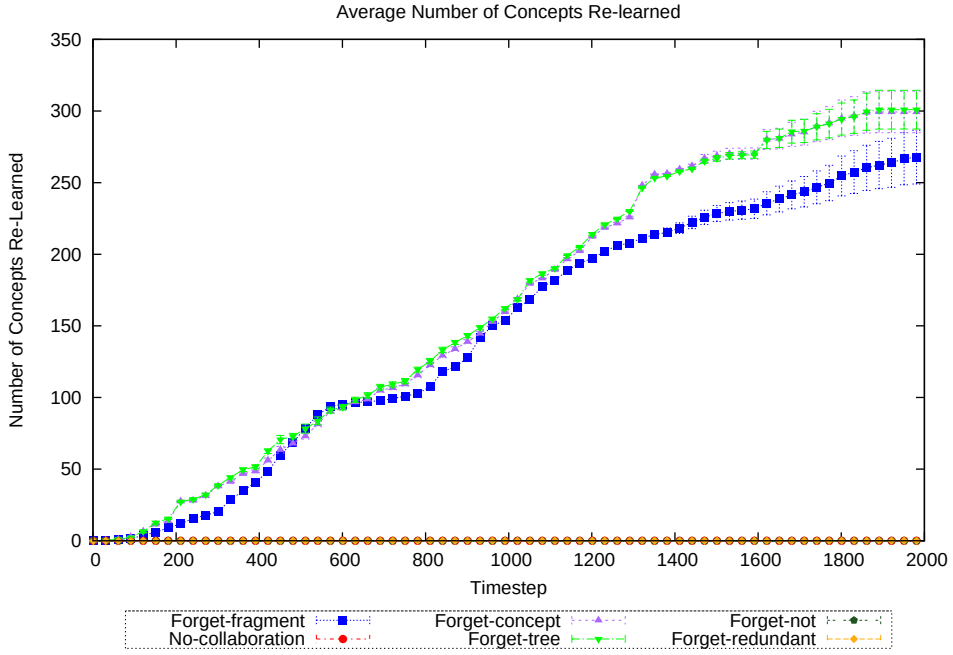


FIGURE C.7: A graph showing the average number of concepts that are relearned.

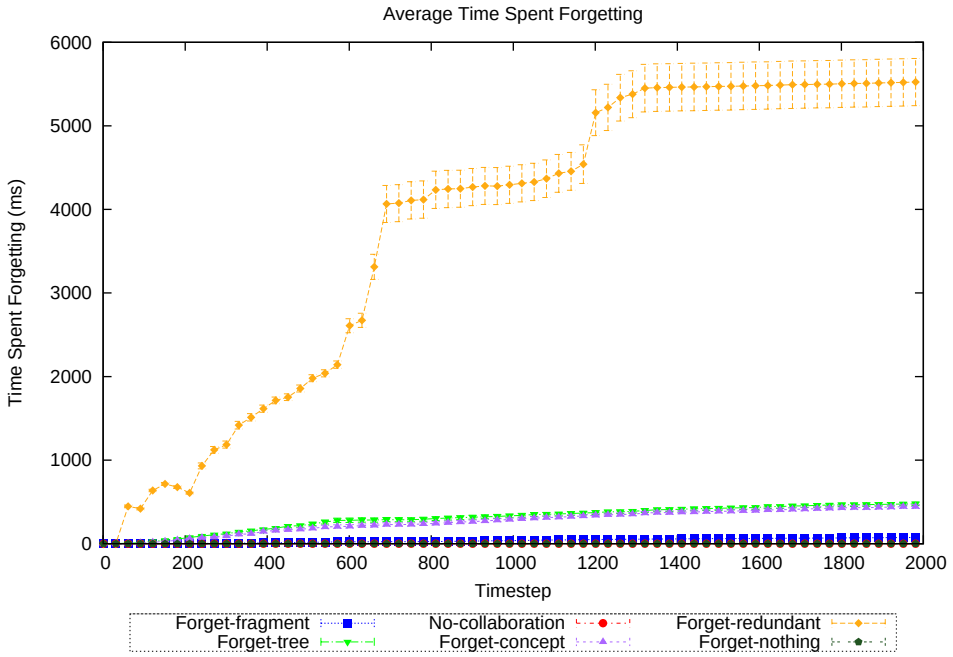


FIGURE C.8: A graph showing the average time an agent spends forgetting.

Approach	Average final number
Forget-tree	339.50
Forget-fragment	566.43
Forget-concept	340.93
Forget-repeated	86.71
Forget-nothing	225.31

TABLE C.2: Statistics for the average number of attempted tasks an agent performs.

Approach	Average final value
Forget-tree	9.71
Forget-fragment	42.50
Forget-concept	9.64
Forget-repeated	5.21
Forget-nothing	6.38

TABLE C.3: Statistics for the average number of successful tasks an agent performs.

Approach	**Average final value**
Forget-tree	701.74
Forget-fragment	399.33
No-collaboration	0
Forget-concept	702.53
Forget-repeated	838.00
Forget-nothing	62.26

TABLE C.4: Statistics for the average number of messages sent by an agent.

Approach	**Average final number of concepts**
Forget-tree	121.02
Forget-fragment	89.70
No-collaboration	0
Forget-concept	121.02
Forget-repeated	0
Forget-nothing	785.92

TABLE C.5: Statistics for the average number of concepts in an agent's ontology.

Approach	**Average final value**
Forget-tree	300.93
Forget-fragment	269.21
Forget-concept	299.36
Forget-repeated	0

TABLE C.6: Statistics for the average number of concepts relearned.

Approach	**Average final value**
Forget-tree	478.77
Forget-fragment	74.44
No-collaboration	0
Forget-concept	446.19
Forget-repeated	5527.21
Forget-nothing	0

TABLE C.7: Statistics for the average time spent forgetting.

C.2 Experiment 2: Ambulance Agents

In this section we now present Experiment 2, which focuses on the performance of the ambulance agents within the RoboCup OWLRescue framework. We aim to show that our approach benefits the ambulance agents, so that they can perform better than the benchmark approaches. We use the experimental setup in Table C.1. From the results, we show that the ambulance agents using our approach outperforms the other approaches that do not have perfect foresight, in terms of RoboCup Rescue, by saving on average more civilians (see Figure C.9) and more of the city (see Figure C.10). This is because our approach removes more than one concept at a time from an agent's ontology and thus spends the least total time forgetting (see Figure C.16 and Table C.13). They were also able to complete the greatest number of tasks, while maintaining an optimal sized ontology with the lowest estimated complexity (see Figure C.11).

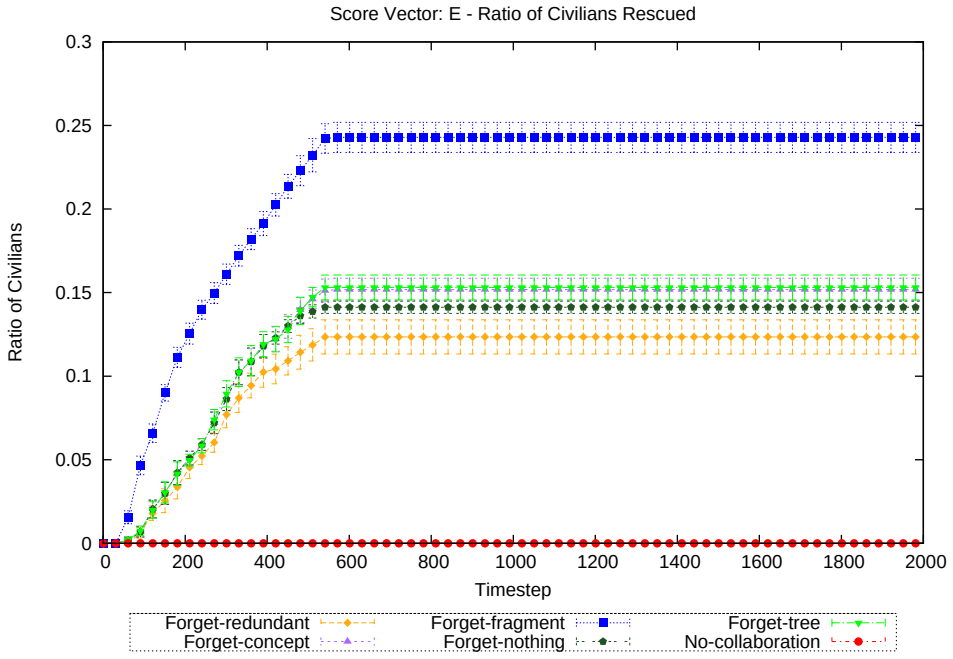


FIGURE C.9: A graph showing the RCR score vector E, showing the average ratio of civilians saved.

C.3 Summary

In summary, both the results from Experiment 1 and 2 show that both the fire brigade and ambulance agents benefit from using our forgetting algorithm and outperform the other approaches. While both the fire brigade and ambulance agents using the com-

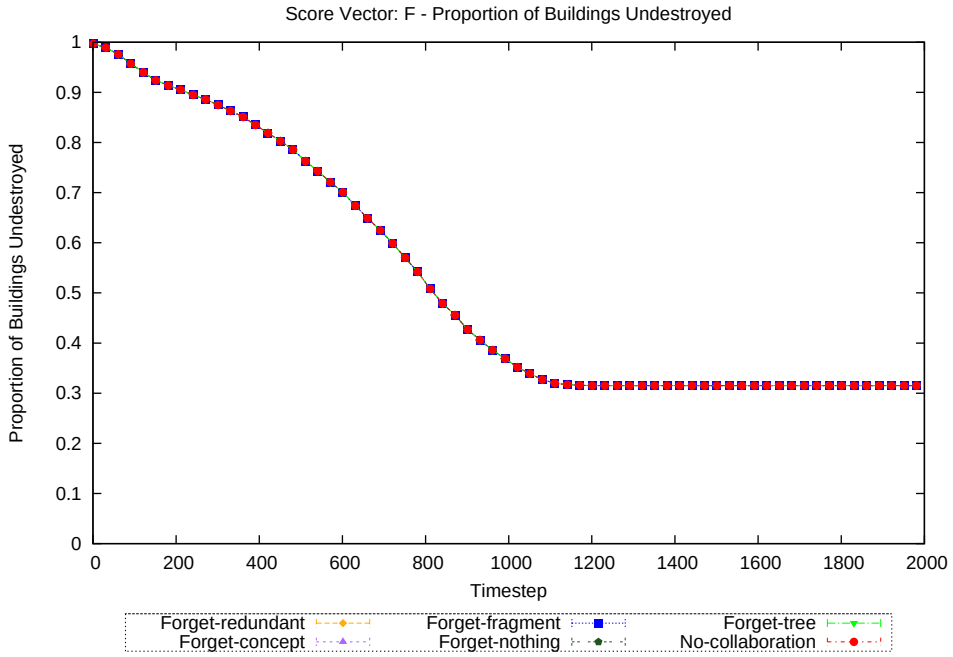


FIGURE C.10: A graph showing the RCR score vector F, showing the average proportion of buildings unburned.

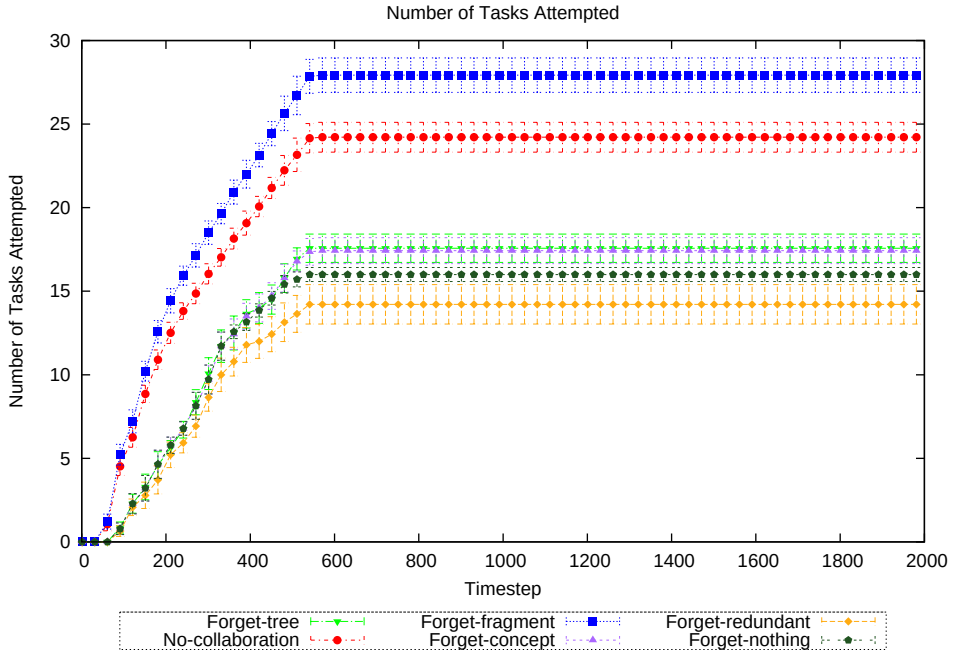


FIGURE C.11: A graph showing the average number of tasks attempted per agent.

Approach	Average final number
Forget-tree	17.57
Forget-fragment	27.93
Forget-concept	17.43
Forget-repeated	14.21
Forget-nothing	15.98

TABLE C.8: Statistics for the average number of attempted tasks an agent performs.

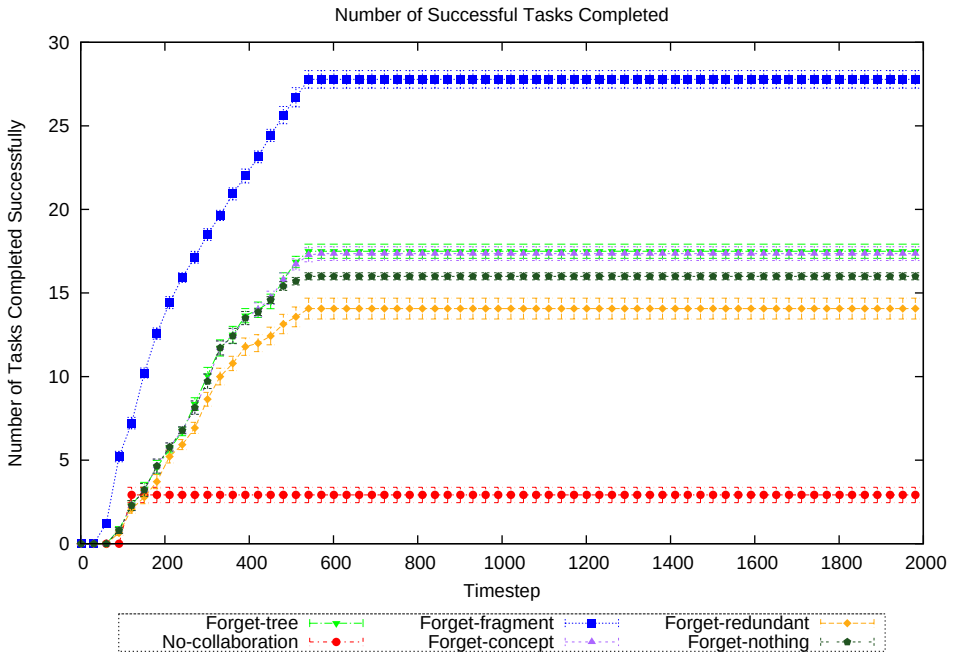


FIGURE C.12: A graph showing the average number of successfully completed tasks per agent.

Approach	Average final value	Average minimum value	Average maximum value
Forget-tree	17.5	14	21
Forget-fragment	27.79	24	31
Forget-concept	17.36	14	20
Forget-repeated	14.07	8	19
Forget-nothing	15.89	14	18

TABLE C.9: Statistics for the average number of successful tasks an agent performs.

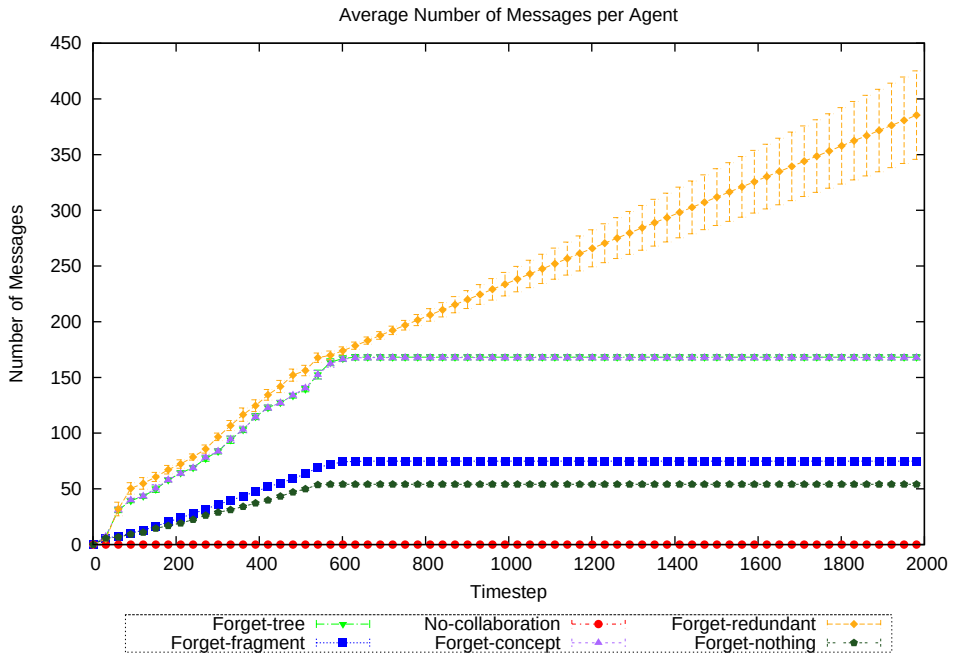


FIGURE C.13: A graph showing the average number of messages sent per agent.

Approach	**Average number of timesteps with no message sent**	**Average final value**
Forget-tree	3	168.15
Forget-fragment	3	75.88
No-collaboration	2000	
Forget-concept	3	167.88
Forget-repeated	3	388.35
Forget-nothing	3	54.05

TABLE C.10: Statistics for the average number of messages sent by an agent.

Approach	**Average final number of concepts**	**Average maximum number of concepts**
Forget-tree	381.97	126.75
Forget-fragment	959.53	59.90
No-collaboration	0	
Forget-concept	382.6	124.35
Forget-repeated	3	291.26
Forget-nothing	3	41.87

TABLE C.11: Statistics for the average number of concepts in an agent's ontology.

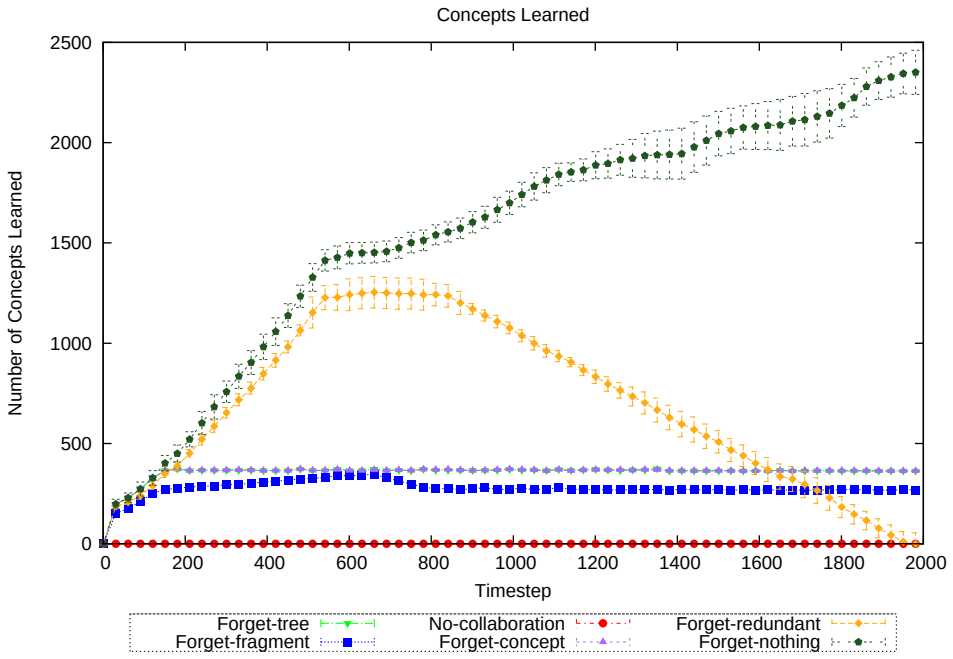


FIGURE C.14: A graph showing the average number of concepts in an agent's ontology.

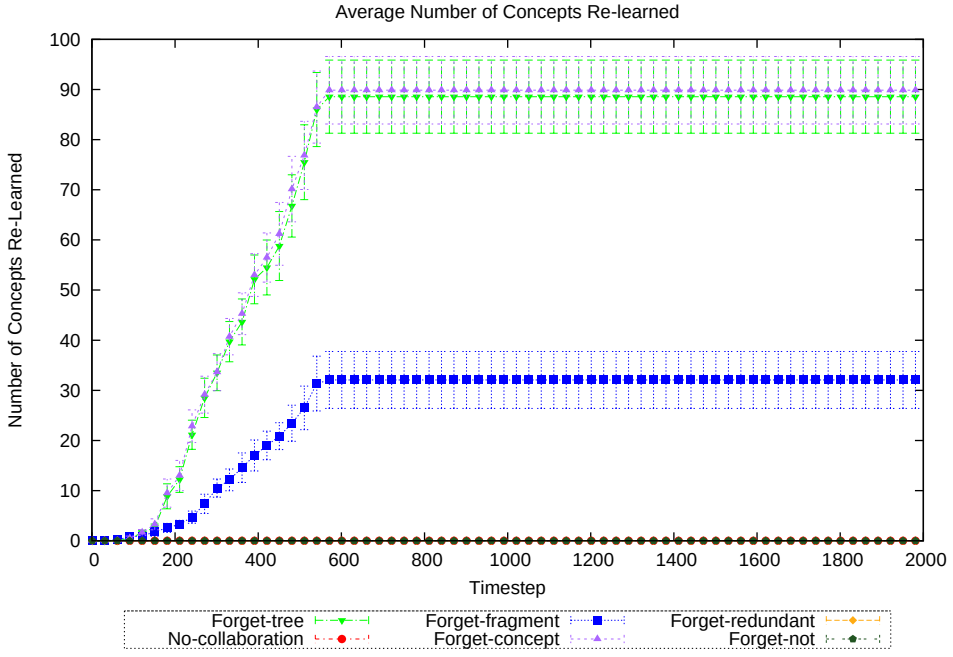


FIGURE C.15: A graph showing the average number of concepts relearned.

Approach	Average final value	Average minimum value	Average maximum value
Forget-tree	88.57	67	115
Forget-fragment	32.07	21	60
Forget-concept	89.86	68	115
Forget-repeated	0	0	0

TABLE C.12: Statistics for the average number of concepts relearned.

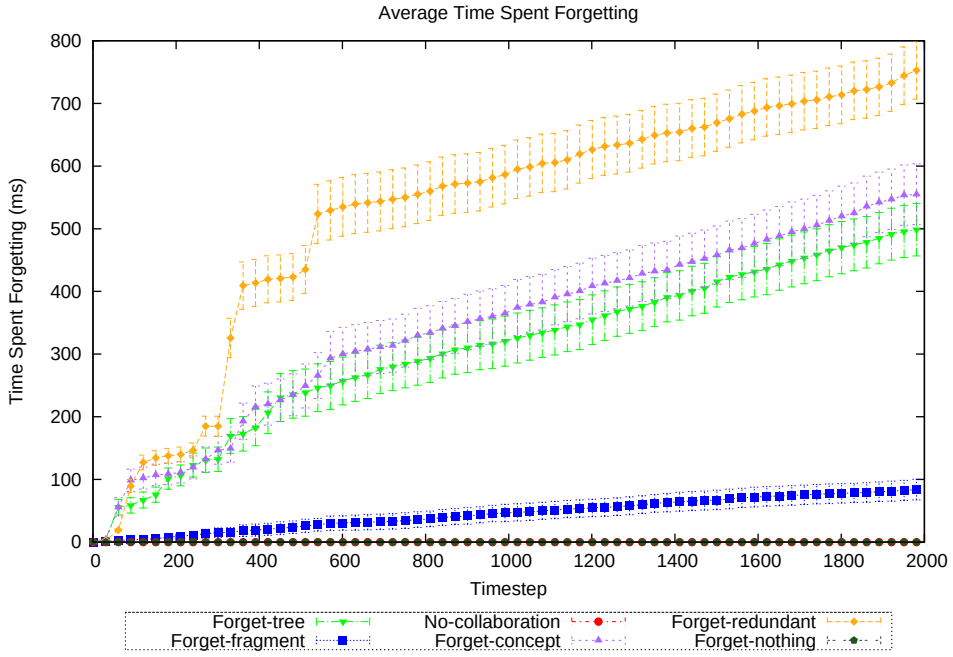


FIGURE C.16: A graph showing the average time an agent spends forgetting.

Approach	Average final value	Average minimum value	Average maximum value
Forget-tree	502.86	210	1190
Forget-fragment	367.86	180	660
No-collaboration	0	0	0
Forget-concept	559.29	280	1390
Forget-repeated	753.57	210	3760
Forget-nothing	0	0	0

TABLE C.13: Statistics for the average time spent forgetting.

parative approaches exhibit the same trends compared to Experiment 3 which utilises all types of agent, the fire brigade (shown in Experiment 1) is more effective than the ambulance agents (shown in Experiment 2) because its actions prevent the fire from spending thus reducing the number of civilian casualties.

Appendix D

Treatment Ontology

In this chapter we have serialised 60 out of 1854 concepts from the treatment ontology (as described in Section 3.2.4), in N3 format. This fragment contains 3.24% (rounded to the nearest 2^{nd} significant figure) of the concepts from the original ontology.

```
@prefix : <http://ontologies.com/treatment#> .
@prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#> .
@prefix Ambulance_Patient-Care:
    <http://ontologies.com/treatment#Ambulance_Patient-Care/> .
@prefix symptomsAndTreatment: <http://ontologies.com/treatment#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix bag-valve_mask_resuscitator:
    <http://ontologies.com/treatment#bag-valve_mask_resuscitator,> .
@prefix stretcher: <http://ontologies.com/treatment#stretcher,> .
@prefix suction_unit: <http://ontologies.com/treatment#suction_unit,> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix multi-purpose: <http://ontologies.com/treatment#multi-purpose/> .
@base <http://ontologies.com/treatment> .

<http://ontologies.com/treatment> rdf:type owl:Ontology ;
    rdfs:comment "information from http://www.facs.org/trauma/
    publications/ambulance.pdf" .

:hasEquipment rdf:type owl:ObjectProperty .

:hasQuantity rdf:type owl:ObjectProperty .

:hasStandard rdf:type owl:ObjectProperty .

:hasSymptom rdf:type owl:ObjectProperty .

:Ambulance_Accessory_Equipment rdf:type owl:Class ;
    rdfs:subClassOf :ambulances .
```

```
Ambulance_Patient-Care:Transportation_Equipment rdf:type owl:Class ;
    rdfs:subClassOf :ambulances ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :toilet_tissue_roll
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :burn_kit
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :blankets
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom bag-valve_mask_resuscitator:_adult
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom suction_unit:_vehicle_mounted
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom suction_unit:_hand_operated
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom multi-purpose:malleable_splint
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :safety_helmet
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom suction_unit:_disposable_collection_bag
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom stretcher:_scoop
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :eye_pads
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom suction_unit:_tubing
        ] ,
        [ rdf:type owl:Restriction ;
```

```
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :abdominal_pads
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :spinal_immobilization_extrication_device
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :rescue_blankets_foil_type
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :spinal_board
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :traction_splint
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :urinal_plastic
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :box_disposable_tissues
        ] .

:B rdf:type owl:Class ;
    rdfs:subClassOf :priority ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :Massive_brain_injuries
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :Cardiac_arrest
        ] .

:Cardiac_arrest rdf:type owl:Class ;
    rdfs:subClassOf :symptoms ;
    rdfs:comment "Cardiac arrest (especially resulting from trauma or
    blood loss). Respiratory arrest when not due to drugs or upper airway
    obstruction."@en .

:Circulatory_shock rdf:type owl:Class ;
    owl:equivalentClass :cirulatory_shock ;
    rdfs:subClassOf :symptoms .

:Coma rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .
```

```
:Emergency_Response_Vehicle_Equipment rdf:type owl:Class ;
    rdfs:subClassOf :ambulances ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom multi-purpose:malleable_splint
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :burn_kit
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :traction_splint
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasEquipment ;
          owl:allValuesFrom :spinal_immobilization_extrication_device
        ] .

:G rdf:type owl:Class ;
    rdfs:subClassOf :priority ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :fingertip_amputations
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :nosebleed
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :first_degree_burn
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :uncomplicated_clean_lacerations
        ] .

:L2-001 rdf:type owl:Class ;
    rdfs:subClassOf :standard .

:L2-005 rdf:type owl:Class ;
    rdfs:subClassOf :standard .

:L2-035 rdf:type owl:Class ;
    rdfs:subClassOf :standard .

:L2-040 rdf:type owl:Class ;
    rdfs:subClassOf :standard .
```

```
:L2-110 rdf:type owl:Class ;
    rdfs:subClassOf :standard .

:L2-145 rdf:type owl:Class ;
    rdfs:subClassOf :standard .

:L2-240 rdf:type owl:Class ;
    rdfs:subClassOf :standard .

:Land_Ambulance_Advanced_Life_Support_Equipment_and_Drug_List rdf:type
    owl:Class ;
    rdfs:subClassOf :ambulances .

:Life-threatening_bleeding rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:Massive_brain_injuries rdf:type owl:Class ;
    rdfs:subClassOf :symptoms ;
    rdfs:comment "Massive brain injuries (indicated by massive head
    trauma, dilated-fixed pupils, absence of all reflexes, extrusion of
    brain matter)."@en .

:R rdf:type owl:Class ;
    rdfs:subClassOf :priority ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :Coma
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :Upper_airway_obstruction
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :Circulatory_shock
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :Life-threatening_bleeding
        ] .

:Upper_airway_obstruction rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:Y rdf:type owl:Class ;
    rdfs:subClassOf :priority ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :penetrating_wound_of_eyeball
        ] ,
        [ rdf:type owl:Restriction ;
```

```
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :head_injury
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasSymptom ;
          owl:someValuesFrom :amputation_of_extremity
        ] .

:abdominal_pads rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] .

:ambulances rdf:type owl:Class .

:amputation_of_extremity rdf:type owl:Class ;
    owl:equivalentClass :fingertip_amputations ;
    rdfs:subClassOf :symptoms .

bag-valve_mask_resuscitator:_adult rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .

:blankets rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q4
        ] .

:box_disposable_tissues rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .

:burn_kit rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasStandard ;
          owl:allValuesFrom :L2-035
```

```
        ] .

:cirulatory_shock rdf:type owl:Class ;
    rdfs:subClassOf :symptoms ;
    rdfs:comment "Circulatory shock which has responded adequately to
    initial treatment with one liter of IV fluid."@en .

:equipment rdf:type owl:Class .

:eye_pads rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q10
        ] .

:fingertip_amputations rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:first_degree_burn rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:head_injury rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

multi-purpose:malleable_splint rdf:type owl:Class ;
    rdfs:subClassOf :splints ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q6
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q4
        ] .

:nosebleed rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:penetrating_wound_of_eyeball rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:priority rdf:type owl:Class .

:q1 rdf:type owl:Class ;
    rdfs:subClassOf :quantity .

:q10 rdf:type owl:Class ;
    rdfs:subClassOf :quantity .

:q2 rdf:type owl:Class ;
```

```
    rdfs:subClassOf :quantity .

:q4 rdf:type owl:Class ;
    rdfs:subClassOf :quantity .

:q6 rdf:type owl:Class ;
    rdfs:subClassOf :quantity .

:quantity rdf:type owl:Class .

:rescue_blankets_foil_type rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] .

:safety_helmet rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] .

:spinal_board rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] .

:spinal_immobilization_extrication_device rdf:type owl:Class ;
    rdfs:subClassOf :splints ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .

:splints rdf:type owl:Class ;
    rdfs:subClassOf :equipment .

:standard rdf:type owl:Class .

stretcher:_scoop rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
```

```
          owl:allValuesFrom :q1
        ] .

suction_unit:_disposable_collection_bag rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q4
        ] .

suction_unit:_hand_operated rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .

suction_unit:_tubing rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] .

suction_unit:_vehicle_mounted rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .

:symptoms rdf:type owl:Class .

:toilet_tissue_roll rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .

:traction_splint rdf:type owl:Class ;
    rdfs:subClassOf :splints ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q2
        ] .
```

```
:uncomplicated_clean_lacerations rdf:type owl:Class ;
    rdfs:subClassOf :symptoms .

:urinal_plastic rdf:type owl:Class ;
    rdfs:subClassOf :equipment ,
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasQuantity ;
          owl:allValuesFrom :q1
        ] .
```

Appendix E

Ontology Fragments based on the Concept `bromine`

In this chapter we have serialised two ontology fragments that are generated from the environment ontologies Chemical Sampling Information (CSI) and treatment ontology (see Section 3.2.4). These fragments represent the concept `bromine`. To this end, the following two sections contain the two fragments.

E.1 Fragment from the Chemical Sampling Information (CSI) ontology

In this section we have serialised a fragment about the concept `bromine` from the CSI ontology.

```
@prefix : <http://ontologies.com/fragment/CSI/bromine#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://ontologies.com/fragment/CSI/bromine#> rdf:type owl:Ontology .

:7726-95-6 rdf:type owl:Class;
    rdfs:subClassOf :CAS_Number .

:Bromine rdf:type owl:Class;
    rdfs:label "Bromine";
    rdfs:subClassOf :Chemical,
    [ rdf:type owl:Restriction;
      owl:onProperty :hasSymptom;
      owl:someValuesFrom :pneumonitis
    ],
    [ rdf:type owl:Restriction;
      owl:onProperty :hasSymptom;
```

```
    owl:someValuesFrom :eye_irritation_lacrimation
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :affectsOrgan;
    owl:someValuesFrom :Respiratory_system
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :pulmonary_edema
  ],
  [ rdf:type owl:Restriction;
    owl:allValuesFrom :7726-95-6;
    owl:onProperty :hasCASNumber
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :epistaxis
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :affectsOrgan;
    owl:someValuesFrom :CNS
  ],
  [ rdf:type owl:Restriction;
    owl:allValuesFrom :HE11;
    owl:onProperty :hasHealthEffect
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :abdominal_pain
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :diarrhea
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :skin_burns
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :feeling_of_oppression
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :affectsOrgan;
    owl:someValuesFrom :skin
  ],
  [ rdf:type owl:Restriction;
    owl:onProperty :hasSymptom;
    owl:someValuesFrom :Dizziness_headaches
  ],
  [ rdf:type owl:Restriction;
```

```
      owl:onProperty :hasSymptom;
      owl:someValuesFrom :reactive_airways_dysfunction_syndrome
    ],
    [ rdf:type owl:Restriction;
      owl:onProperty :hasSymptom;
      owl:someValuesFrom :measle-like_eruptions
    ],
    [ rdf:type owl:Restriction;
      owl:onProperty :hasSymptom;
      owl:someValuesFrom :
   bronchospasm_shortness_of_breath_coughing_chest_pain_or_tightness
    ],
    [ rdf:type owl:Restriction;
      owl:onProperty :affectsOrgan;
      owl:someValuesFrom :eyes
    ],
    [ rdf:type owl:Restriction;
      owl:allValuesFrom :HE14;
      owl:onProperty :hasHealthEffect
    ] .

:CAS_Number rdf:type owl:Class .

:CNS rdf:type owl:Class;
    rdfs:subClassOf :Organs_Affected .

:Chemical rdf:type owl:Class .

:Dizziness_headaches rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:HE11 rdf:type owl:Class;
    rdfs:subClassOf :Health_Effect_Code .

:HE14 rdf:type owl:Class;
    rdfs:subClassOf :Health_Effect_Code .

:Health_Effect_Code rdf:type owl:Class;
    rdfs:subClassOf :Health_Effects .

:Health_Effects rdf:type owl:Class .

:Organs_Affected rdf:type owl:Class;
    rdfs:subClassOf :Health_Effects .

:Respiratory_system rdf:type owl:Class;
    rdfs:subClassOf :Organs_Affected .

:Symptom rdf:type owl:Class;
    rdfs:subClassOf :Health_Effects .
```

```
:abdominal_pain rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:affectsOrgan rdf:type owl:ObjectProperty .

:bronchospasm_shortness_of_breath_coughing_chest_pain_or_tightness rdf:
    type owl:Class;
    rdfs:subClassOf :Symptom .

:diarrhea rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:epistaxis rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:eye_irritation_lacrimation rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:eyes rdf:type owl:Class;
    rdfs:subClassOf :Organs_Affected,
            :Symptom .

:feeling_of_oppression rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:hasCASNumber rdf:type owl:ObjectProperty .

:hasHealthEffect rdf:type owl:ObjectProperty .

:hasSymptom rdf:type owl:ObjectProperty .

:measle-like_eruptions rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:pneumonitis rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:pulmonary_edema rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:reactive_airways_dysfunction_syndrome rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

:skin rdf:type owl:Class;
    rdfs:subClassOf :Organs_Affected,
            :Symptom .

:skin_burns rdf:type owl:Class;
    rdfs:subClassOf :Symptom .

rdfs:label rdf:type owl:AnnotationProperty .
```

E.2 Fragment from the treatment ontology

In this section we have serialised a fragment about the concept `bromine` from the treatment ontology.

```
@prefix : <http://ontologies.com/fragment/treatment/bromine#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://ontologies.com/fragment/treatment/bromine#> rdf:type owl:Ontology
     .

:Thing rdf:type owl:Class .

:blisters rdf:type owl:Class;
    rdfs:subClassOf :symptom .

:breathing_100percent_oxygen rdf:type owl:Class;
    rdfs:subClassOf :treatment .

:bromine rdf:type owl:Class;
    rdfs:subClassOf :chemical_exposure,
    [ a owl:Restriction;
      owl:onProperty :hasTreatment;
      owl:someValuesFrom :breathing_100percent_oxygen
    ],
    [ a owl:Restriction;
      owl:onProperty :hasTreatment;
      owl:someValuesFrom :remove_contaminated_clothing
    ],
    [ a owl:Restriction;
      owl:onProperty :hasSymptom;
      owl:someValuesFrom :respiratory_distress
    ],
    [ a owl:Restriction;
      owl:onProperty :hasTreatment;
      owl:someValuesFrom :rinse_exposed_hair
    ],
    [ a owl:Restriction;
      owl:onProperty :hasSymptom;
      owl:someValuesFrom :blisters
    ] .

:chemical_exposure rdf:type owl:Class .

:hasSymptom rdf:type owl:ObjectProperty .

:hasTreatment rdf:type owl:ObjectProperty .

:remove_contaminated_clothing rdf:type owl:Class;
    rdfs:subClassOf :treatment .
```

```
:respiratory_distress rdf:type owl:Class;
    rdfs:subClassOf :symptom .

:rinse_exposed_hair rdf:type owl:Class;
    rdfs:subClassOf :treatment .

:symptom rdf:type owl:Class;
    rdfs:subClassOf :Thing .

:treatment rdf:type owl:Class .
```

Glossary

A_s The set of all specialist agents in an environment.

A_t The set of all task agents in an environment.

A The set of all agents in an environment.

C^h The concepts selected from the hierarchical selection, where h denotes that the set of concepts C were selected by the **h**ierarchical algorithm.

C^r The concepts selected from the relational selection algorithm, where r denotes that the set of concepts C were selected by the **r**elational algorithm.

C_r The set of concepts which are removed from the merged fragment, where $C_r = C^r \backslash concepts(F_d)$.

C A set of concepts.

DO Domain Ontology (DO), a static ontology that contains all the axioms that an agent is initiated with.

EO Evolving Ontology (EO), a dynamic ontology that contains all the axioms that have been augmented into an agent's ontology.

F The set of fragments received by a task agent.

HP Health Points, these belong to RCR agents, and RCR civilians.

K A set of axioms.

O A set of ontologies.

P_c The set of concepts that represent the path from concept c to the root concept.

P_o The number of paths in an ontology o.

Q_f The set of future queries.

R The finite set of types of relationships.

T A set of tasks.

λ_i The longest path length of concept c_i to the root node.

μ The average number of relationships per concept.

$\overline{\lambda_i}$ The average path length of concept c_i to the root node.

ρ The average paths per concept in an ontology.

σ The ratio of maximum path length to average path length of an ontology.

$a_{s,j}$ A specialist agent that refers to ontology o_s, in our use case o_s would be an environment ontology.

$a_{t,i}$ A task agent which has an ontology o_t. Where t denotes that this agent is a task agent, and i is a unique identifier for this task agent.

abf The average branching factor.

bf_c The branching factor of the concept c.

c_j A concept from the contained in the ontology o_j. This notation is used in Chapter 2.

c_t A target concept or a concept which is required to complete a task, the task agent that receives this concept desires to incorporate axioms that represent this concept into its ontology.

$commonConcepts(o_i, o_j)$ A Boolean function which denotes that both of the ontologies o_i and o_j contain one or more of the same concepts..

$concepts(o_i)$ A function that returns the set of concepts which are referred to in the ontology o_i.

d_c The average of cost of a single translation.

d_f The cost of generating a fragment.

d_s Cost of sending a message.

$depth(concepts(x))$ A function that return a number which is the depth of an ontology, or fragment of an ontology x.

eo^+ This notation is the result of adding one or more concepts from an evolving ontology (eo).

eo^- This notation is the result of removing one or more concepts from an evolving ontology (eo).

f_m A fragment that has been merged after the merged fragment process.

$f_{1,firefighting\_motorcycle}$ An example fragment that represents the concept `firefighting motorcycle`, where 1 specifies the unique identifier of the fragment, and *firefighting_motorcycle* is a concept.

$f_{2,firefighting\_motorcycle}$ An example fragment that represents the concept `firefighting motorcycle`, where 2 specifies the unique identifier of the fragment, and *firefighting_motorcycle* is a concept.

f_{c_t} A fragment that represents the target concept c_t.

l The capacity limit of concepts that a task agent's ontology can contain.

$nDOR$ The number of times a specific concept relates to a concept contained in the DO.

nDO The number of times a specific concept is referred to by axioms contained in the DO.

$nEOR$ The number of times a specific concept relates to a concept contained in the EO.

nEO The number of times a specific concept is referred to by axioms contained in the EO.

n_b The cost to end a byte of data over a network.

o'_i An ontology which contains the concepts in o_i and the concept c_j. This notation is used in Chapter 2.

o_i An ontology which does not contain the concept c_j. This notation is used in Chapter 2.

o_i An ontology that belongs to agent a_i..

o_j An ontology which is used to provide a description of a concept (c_j) which is not contained in the ontology o_i. This notation is used in Chapter 2.

o_j An ontology that belongs to agent $a_{s,j}$.

o_t A task agent's ontology, where t denotes that it belongs to a task agent.

o Is an ontology.

r_c The number of concepts that related to one another relationships.

$refers\_to(k_i)$ A function that denotes the set of concepts referred to in axiom k_i.

t A threshold which determines the number of relationships are used to select concepts to incorporate into an agent's ontology by the relational selection process.

$weight(c)$ A function that returns the concept rating of a concept.

w A weighting for concepts that are contained in the DO.

$|F|$ The number of fragments in a set of fragments F.

$|b|$ The size of a message in bytes.

$|f|$ The number of axioms in a fragment f.

concept rating The rating given to a concept contained in f_m by the hierarchal selection algorithm.

A-Box Assertional Box, a colloquial term that describes all the assertional axioms in an ontology.

ADIT-LN Average Depth of Inheritance Tree of all Leaf Nodes.

NOL Number Of Leaves.

NORC Number Of Root Classes.

RCR RoboCup Rescue.

T-Box Terminological Box, a colloquial term that describes all the terminological axioms in an ontology.

Bibliography

M. Afsharchi, B.H. Far, and J. Denzinger. Ontology-Guided Learning to Improve Communication between Groups of Agents. *In the Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan*, pages 923–930, 2006.

H. Alani, C. Brewster, and N. Shadbolt. Ranking Ontologies with AKTiveRank. *In the Proceedings of the Fifth International Semantic Web Conference (ISWC 2006), Athens, Georgia, USA*, 4273:1–15, 2006a.

H. Alani, S. Harris, and B. O'Neil. Winnowing Ontologies based on Application Use. *In the Proceedings of the Third European Semantic Web Conference (ESWC 2006), Budva, Montenegro*, 2006b.

M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, et al. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*, 25(1):25–29, 2000.

M. Aslani and V. Haarslev. TBox Classification in Parallel: Design and First Evaluation. *In the Proceedings of the Twenty Third International Workshop on Description Logics (DL 2010), Waterloo, Canada*, page 336, 2010.

F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN 0521781760.

S.C. Bailin. Software Reuse as Ontology Negotiation. *Software Reuse: Methods, Techniques and Tools*, pages 242–253, 2004.

S.C. Bailin and W. Truszkowski. Ontology Negotiation between Intelligent Information Agents. *The Knowledge Engineering Review*, 17(01):7–19, 2002.

J. Bao and V. Honavar. Collaborative package-based ontology building and usage. *Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources workshop in the Fifth IEEE International Conference on Data Mining (ICDM 2005), Houston, Texas*, pages 35–44, 2005.

L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

J. Baumeister and D. Seipel. Smelly Owls–Design Anomalies in Ontologies. *In the Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2005), Florida, USA*, pages 215–220, 2005.

A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press Cambridge, 1998. ISBN 0894484532.

W. N. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, Universiteit Twente, Enschede, September 1997.

J. Bosak and T. Bray. XML and the Second-Generation Web. *Scientific American*, 280 (5):89–93, 1999. ISSN 0036-8733.

J. Burbea and CR Rao. Entropy Differential Metric, Distance and Divergence Measures in Probability Spaces- A Unified Approach. *NTIS, Springfield, Virginia, 1980, 38*, 1980.

J. Cardoso. The Semantic Web Vision: Where Are We? *IEEE Intelligent Systems*, 22 (5):84–88, 2007.

I.H. Chou and C.F. Fan. An Agent-based National Radioactive Waste Management Framework design. *Progress in Nuclear Energy*, 52(5):470–480, 2010. ISSN 0149-1970.

Ó. Corcho and A. Gómez-Pérez. A Roadmap to Ontology Specification Languages. *In the Proceedings of the Twelfth European Workshop on Knowledge Acquisition, Modeling and Management (EKAW 2000), Juan-les-Pins, France*, pages 80–96, 2000.

P.C.G. Costa and K.B. Laskey. PR-OWL: A Framework for Probabilistic Ontologies. *In the Proceedings of the Fourth International Conference, Formal Ontology in Information Systems (FOIS 2006), Baltimore, Maryland, USA*, page 237, 2006.

B. Cuenca Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and Web Ontologies. *In the Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District, UK*, 6:198–209, 2006.

F.C. da Silva, W. Vasconcelos, and D. Robertson. Cooperation between Knowledge Based Systems. *In the Proceedings of the Fourth World Congress on Expert Systems (WCES 1998), Mexico City, Mexico, USA*, pages 819–825, 1998.

M. de Oliveira, V. Furtado, S. Cranefield, and M. Purvis. Open collaborative systems as institutions of agents. *In the Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, (WI-IAT 2008), Sydney, Australia*, 2:100–103, 2009.

Y. Dodge. *The Oxford Dictionary of Statistical Terms*. Oxford University Press, USA, 2006. ISBN 0198509944.

P. Doherty, W. Łukaszewicz, and A. Szałas. Approximative Query Techniques for Agents with Heterogeneous Ontologies and Perceptive Capabilities. *In the Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), Whistler, Canada*, 2004.

P. Doran, V. Tamma, and L. Iannone. Ontology module extraction for ontology reuse: An ontology engineering perspective. *In the Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM 2007), Lisbon, Portugal*, pages 61–70, 2007.

S. Edelkamp and R.E. Korf. The Branching Factor of Regular Search Spaces. *In the Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 1998), Workshop on AI and Information Integration, Madison, Wisconsin, USA*, pages 299–304, 1998.

T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. *In the Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland*, 19:90, 2005.

T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, and K. Wang. Forgetting in Managing Rules and Ontologies. *In the Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006), HKCEC, Hong Kong, China*, pages 411–419, 2006.

T. Eiter and K. Wang. Forgetting and Conflict Resolving in Disjunctive Logic Programming. *In the Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006), Boston, Massachusetts*, page 238, 2006.

M. Esteva, J.A. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the Formal Specification of Electronic Institutions. *Agent Mediated Electronic Commerce*, pages 126–147, 2001.

J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2007. ISBN 3540496114.

J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): proceedings*, page 333. Ios Pr Inc, 2004. ISBN 1586034529.

D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. Della Valle, F. Fischer, Z. Huang, A. Kiryakov, T.K. Lee, et al. Towards LarKC: A Platform for

Web-Scale Reasoning. *In the Proceedings of the 2008 IEEE International Conference on Semantic Computing (ICSC 2008), Santa Clara, California, USA*, pages 524–529, 2008.

T. Finin and A. Joshi. Agents, Trust, and Information Access on the Semantic Web. *ACM SIGMOD Record*, 31(4):30–35, 2002. ISSN 0163-5808.

G.D. Forney Jr. The Viterbi Algorithm (Viterbi Algorithm for Recursive Optimal Estimation of State Sequence of Discrete Time Finite State Markov Process Observed in Memoryless Noise. *In the Proceedings of the IEEE*, 61(3):268–278, 1973.

Q. Gao. Towards Trust in Web Content Using Semantic Web Technologies. *In the Proceedings of the Seventh Extended Semantic Web Conference (ESWC 2010), Heraklion, Greece*, pages 457–461, 2010.

M.R. Genesereth and S.P. Ketchpel. Software Agents. *Communications of the ACM*, 37 (7):48–53, 1994.

M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1987. ISBN 0934613311.

B.C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the Right Amount: Extracting Modules from Ontologies. *In the Proceedings of the Sixteenth International Conference on World Wide Web (WWW 2007), Banff, Alberta, Canada*, page 726, 2007.

P. Grenon, B. Smith, and L. Goldberg. Biodynamic Ontology: Applying BFO in the Biomedical Domain. *Ontologies in Medicine, Amsterdam, The Netherlands*, pages 20–38, 2004.

R. Grossmann. *The Existence of the World: An Introduction to Ontology (The Problems of Philosophy)*. Routledge, 1992. ISBN 0415107725.

T.R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human Computer Studies*, 43(5):907–928, 1995.

T.R. Gruber. It is What it Does: The Pragmatics of Ontology. *Invited Talk at Sharing the Knowledge – the International Committee for Documentation of the International Council of Museums Conceptual Reference Mode Symposium, Washington, DC.*, 2003.

T.R. Gruber. Every Ontology is a Treaty–a Social Agreement–Among People with Some Common Motive in Sharing. *AIS SIGSEMIS Bulletin*, 1(3):4–8, 2005.

Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3 (2-3):158–182, 2005.

P. Haase and L. Stojanovic. Consistent Evolution of OWL Ontologies. *In the Proceedings of the Second European Semantic Web Conference (ESWC 2005), Heraklion, Greece*, pages 182–197, 2005a.

P. Haase and L. Stojanovic. Consistent Evolution of OWL Ontologies. *The Semantic Web: Research and Applications*, pages 182–197, 2005b.

P. Haase and J. Völker. Ontology Learning and Reasoning–Dealing with Uncertainty and Inconsistency. *In the Proceedings of the Seventh International Conference for the Semantic Web (ISWC 2008), in the Fourth Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2008), Karlsruhe, Germany*, pages 366–384, 2008.

S. Hartmann, H. Köhler, and J. Wang. Ontology Consolidation in Bioinformatics. *In the Proceedings of the Seventh Asia-Pacific Conference on Conceptual Modelling (APCCM 2010), Brisbane, Australia*, 100:15–22, 2010.

J. Heflin, J. Hendler, and S. Luke. Reading between the Lines: Using SHOE to Discover Implicit Knowledge from the Web. *In the Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 1998), Workshop on AI and Information Integration, Madison, Wisconsin, USA*, 297, 1998.

J. Heflin, J. Hendler, and S. Luke. Coping with Changing Ontologies in a Distributed Environment. *Association for the Advancement of Artificial Intelligence (AAAI 1999) Conference Ontology Management Workshop, Orlando, Florida, USA*, pages 74–79, 1999.

M. Holford, J. McCusker, K. Cheung, and M. Krauthammer. Analysis Of Cancer Omics Data In A Semantic Web Framework. In *3rd International Workshop on Semantic Web Applications and Tools for the Life Sciences, Berlin, Germany*, 2010.

I. Horrocks, P.F. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7–26, 2003.

I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of IGPL*, 8(3):239, 2000. ISSN 1367-0751.

F.V. Jensen. *An Introduction to Bayesian Networks*. Springer, 1997. ISBN 0387915028.

Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, 18(01):1–31, 2003.

D. Kang, B. Xu, J. Lu, and W. Chu. A Complexity Measure for Ontology Based on UML. *In the Proceedings of the IEEE Tenth International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004), Suzhou, China*, pages 222–228, 2004.

S. Kaushik, D. Wijesekera, and P. Ammann. Policy-Based Dissemination of Partial Web-Ontologies. *In the Proceedings of the 2005 Workshop on Secure Web Services (SWS 2005), Fairfax, Virginia, USA*, pages 43–52, 2005.

H. Kitano and S. Tadokoro. Robocup rescue: A Grand Challenge for Multi-Agent and Intelligent Systems. *AI Magazine*, 22(1):39, 2001.

M. Klein and D. Fensel. Ontology Versioning on the Semantic Web. *In the Proceedings of the International Semantic Web Working Symposium (SWWS 2001), Stanford University, California, USA*, pages 75–91, 2001.

Konstantinos Kotis, Panos Alexopoulos, and Andreas Papasalouros. Towards a framework for trusting the automated learning of social ontologies. In Yaxin Bi and Mary-Anne Williams, editors, *Knowledge Science, Engineering and Management*, volume 6291 of *Lecture Notes in Computer Science*, pages 388–399. Springer Berlin / Heidelberg, 2010.

L. Laera, I. Blacoe, V. Tamma, T. Payne, J. Euzenat, and T. Bench-Capon. Argumentation Over Ontology Correspondences in MAS. *In the Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007), Honolulu, Hawaii, USA*, 2007.

L. Laera, V. Tamma, J. Euzenat, T. Bench-Capon, and T. Payne. Reaching Agreement over Ontology Alignments. *In the Proceedings of the Fifth International Semantic Web Conference (ISWC 2006), Athens, Georgia, USA*, 2006.

D. Lee, J. Choi, H. Choe, S. Noh, S. Min, and Y. Cho. Implementation and Performance Evaluation of the LRFU Replacement Policy. *In the Proceedings of the Twenty-Third Euromicro Conference (EUROMICRO 1997), 'New Frontiers of Information Technology'. Short Contributions, Budapest, Hungary*, pages 106–111, 1997.

G. Lefcoe. The Regulation of Superstores: The Legality of Zoning Ordinances Emerging from the Skirmishes between Wal-Mart and the United Food and Commercial Workers Union. *Arkansas Law Review*, 58:833, 2005.

D.B. Lenat, R.V. Guha, K. Pittman, D. Pratt, and M. Shepherd. Cyc: Toward Programs with Common Sense. *Communications of the ACM*, 33(8):30–49, 1990. ISSN 0001-0782.

H.T. Lin and E. Sirin. Pellint–A Performance Lint Tool for Pellet. *In the Proceedings of the Seventh International Semantic Web Conference, in the Workshop on OWL: Experiences and Directions, (OWLED 2008) Karlsruhe, Germany*, 2008.

L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a Complete OWL Ontology Benchmark. *Sure, Y., Domingue, J., eds.: In the Proceedings of the Third European Semantic Web Conference (ESWC 2006), Volume 4011 of LNCS., Budva, Montenegro*, pages 125–139, 2006.

R. MacGregor. The Evolving Technology of Classification-Based Knowledge Representation Systems. *Principles of Semantic Networks*, pages 385–400, 1991.

A. Maedche and S. Staab. Measuring Similarity Between Ontologies. *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 15–21, 2002.

S. Markovitch and P.D. Scott. The Role of Forgetting in Learning. *In the Proceedings of the Fifth International Conference on Machine Learning (ICML 1988), Ann Arbor, Michigan, USA*, pages 459–465, 1988.

D. Maynard, W. Peters, and Y. Li. Metrics for Evaluation of Ontology-Based Information Extraction. *In the Proceedings of the World Wide Web Conference (WWW 2006), Workshop on the Evaluation of Ontologies for the Web (EON 2006), Edinburgh, Scotland*, 2006.

D.L. McGuinness. Conceptual Modeling for Distributed Ontology Environments. *In the Proceedings of the Eighth International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues (ICCS 2000), Darmstadt, Germany*, 2000.

D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. *In the Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2000), Breckenridge, Colorado, USA*, pages 483–493, 2000.

D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. *W3C Recommendation: http://www.w3.org/TR/owl-features/*, 2004.

G.A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39, 1995.

J. Mun, M. Shin, and M. Jung. A Goal-Oriented Trust Model for Virtual Organization Creation. *Journal of Intelligent Manufacturing*, pages 1–10, 2009. ISSN 0956-5515.

National Chemical Emergency Centre. *Dangerous Goods Emergency Action Code List 2009*. TSO, 2009. ISBN 9780113413263.

N.F. Noy and M. Klein. Ontology Evolution: Not the Same as Schema Evolution. *Knowledge and Information Systems*, 6(4):428–440, 2004.

N.F. Noy and M.A. Musen. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, and F. van Harmelen. Marvin: Distributed Reasoning over Large-Scale Semantic Web Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):305–316, 2009. ISSN 1570-8268.

H. S. Packer, N. Gibbins, and N. R. Jennings. Ontology evolution through agent collaboration. *Artificial Intelligence and the Simulation of Behaviour 2009 Convention (AISB 2009), Edinburgh, UK*, April 2009.

H. S. Packer, N. Gibbins, and N. R. Jennings. Collaborative learning of ontology fragments by cooperating agents. *In the Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2010), Toronto, Canada*, 2: 89–96, 2010a.

H. S. Packer, N. Gibbins, and N. R. Jennings. Forgetting fragments from evolving ontologies. *In the Proceedings of the Ninth International Semantic Web Conference (ISWC 2010), Shanghai, China*, 6496(11), November 2010b.

H. S. Packer, T. Payne, N. Gibbins, and N. R. Jennings. Evolving Ontological Knowledge Bases Through Agent Collaboration. *In the Proceedings of the Sixth European Workshop on Multi-Agent Systems (EUMAS 2008), Bath, UK*, 2008.

I. Palmisano, V. Tamma, T. Payne, and P. Doran. Task oriented evaluation of module extraction techniques. *In the Proceedings of the Eighth International Semantic Web Conference (ISWC 2009), Washington, DC*, 5823:130–145, 2009.

P. Plessers and O. De Troyer. Resolving Inconsistencies in Evolving Ontologies. *The Semantic Web: Research and Applications*, pages 200–214, 2006.

R. Porzel and R. Malaka. A Task-Based Approach for Ontology Evaluation. In *ECAI Workshop on Ontology Learning and Population, Valencia, Spain*, 2004.

G. Qi, Y. Wang, P. Haase, and P. Hitzler. A Forgetting-based Approach for Reasoning with Inconsistent Distributed Ontologies. *In the Proceedings of the Workshop on Ontologies: Reasoning and Modularity (WORM 2008), Tenerife, Spain*, 348, 2008.

A. Rector and R. Stevens. Barriers to the Use of OWL in Knowledge Driven Applications. *In the Proceedings of the Seventh International Semantic Web Conference, in the Workshop OWL: Experiences and Directions, (OWLED 2008) Karlsruhe, Germany*, 2008.

S.L. Reed and D.B. Lenat. Mapping Ontologies into CYC. *In the Proceedings of American Association of Artificial Intelligence Conference, Workshop on Ontologies for the Semantic Web, Edmonton, Alberta, Canada*, 2002.

M. Romero, J. Vázquez-Naya, C. Munteanu, J. Pereira, and A. Pazos. An approach for the automatic recommendation of ontologies using collaborative knowledge. *Knowledge-Based and Intelligent Information and Engineering Systems*, 6277:74–81, 2010.

S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, 1995. ISBN 0136042597.

Alan Ruttenberg, Jonathan A. Rees, Matthias Samwald, and M. Scott Marshall. Life sciences on the semantic web: the neurocommons and beyond. *Briefings in Bioinformatics*, 10(2):193–204, 2009.

R. Sarika, H. Siddhartha, and K. Karlapalem. Database Driven RoboCup Rescue Server. *RoboCup 2008: Robot Soccer World Cup XII, Suzhou, China*, pages 602–613, 2009.

S. Schenk. On the Semantics of Trust and Caching in the Semantic Web. *In the Proceedings of the Seventh International Conference on the Semantic Web (ISWC 2008), Karlsruhe, Germany*, pages 533–549, 2008.

Julian Seidenberg and Alan Rector. Web Ontology Segmentation: Analysis, Classification and Use. *In the Proceedings of the Fifteenth International Conference on World Wide Web (WWW 2006), 23rd-26th May, Edinburgh, Scotland*, pages 13–22, 2006.

M. Sensoy and P. Yolum. A cooperation-Based Approach For Evolution Of Service Ontologies. *In the Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), Estoril, Portugal*, pages 837–844, 2008.

D. Shasha and K. Zhang. Approximate Tree Pattern Matching. *In Pattern Matching Algorithms*, 1997.

J. Shen, R. Becker, and V. Lesser. Agent Interaction in Distributed POMDPs and its Implications on Complexity. *In the Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2006), Hakodate, Japan*, page 536, 2006.

A. Sheth and C. Ramakrishnan. Semantic (Web) Technology In Action: Ontology Driven Information Systems for Search, Integration and Analysis. *IEEE Data Engineering Bulletin*, 26(4):40–48, 2003.

R. Siebes, D. Dupplaw, S. Kotoulas, A.P. De Pinninck, F. Van Harmelen, and D. Robertson. The openknowledge system: an interaction-centered approach to knowledge sharing. *In the Proceedings of the OTM Confederated International Conference on On the Move to Meaningful Internet Systems (OTM 2007), Vilamoura, Portugal*, pages 381–390, 2007.

E. Sirin and B. Parsia. Pellet: An OWL DL reasoner. *In the Proceedings of the 2004 International Workshop on Description Logics (DL 2004), Whistler, British Columbia, Canada*, page 212, 2004.

E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.

M.J. Smith and M. Desjardins. Learning to Trust in the Competence and Commitment of Agents. *In the Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), May 10-15, Budapest, Hungary*, 18 (1):36–82, 2009. ISSN 1387-2532.

L.K. Soh. Multiagent, Distributed Ontology Learning. *Working Notes of the Second International Joint Conference of Autonomous Agents and Multi-Agent Systems (AAMAS) Workshop on Ontologies in Agent Systems (OAS 2002) Workshop, Bologna, Italy*, 2002.

L.K. Soh and C. Chen. Balancing Ontological and Operational Factors in Refining Multiagent Neighborhoods. *In the Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005), Amsterdam, The Netherlands*, pages 745–752, 2005.

J.F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. MIT Press, 2000. ISBN 0534949657.

L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-Driven Ontology Evolution Management. *In the Proceedings of the Thirteenth European Conference on Knowledge Engineering and Knowledge Management (EKAW 2002), Siguenza, Spain*, 2002.

L. Stojanovic, A. Maedche, N. Stojanovic, and R. Studer. Ontology Evolution as Reconfiguration-Design Problem Solving. *In the Proceedings of the Second International Conference on Knowledge Capture (K-CAP 2003), Sanibel Island, Florida, USA*, pages 162–171, 2003.

R. Studer, V.R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

J. Tappolet, C. Kiefer, and A. Bernstein. Semantic web enabled software analysis. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2010. ISSN 1570-8268.

B. Thuraisingham. Building Trustworthy Semantic Webs. *In the Proceedings of the IEEE International Conference on Information Reuse and Integration, (IRI 2009), 10-12 August 2009, Las Vegas, Nevada, USA*, 2009.

C. Trojahn, P. Quaresma, and R. Vieira. Conjunctive Queries for Ontology based Agent Communication in MAS. *In the Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), Estoril, Portugal*, pages 829–836, 2008.

J. van Diggelen, RJ Beun, F. Dignum, RM van Eijk, and JJ Ch. Meyer, Optimal Communication Vocabularies and Heterogeneous Ontologies. *International Workshop on Agent Communication (AC 2004), New York, NY, USA*, 3396:76–90, 2004.

R.M. Van Eijk, F.S. De Boer, W. Van Der Hoek, and J.J.C. Meyer. On Dynamically Generated Ontology Translators in Agent Communication. *International Journal of Intelligent Systems*, 16(5):587–607, 2001.

K. Wang, A. Sattar, and K. Su. A Theory of Forgetting in Logic Programming. *In the Proceedings of the National Conference on Artificial Intelligence (AAAI 2005), Pittsburg , USA*, 20(2):682, 2005.

K. Wang, Z. Wang, R. Topor, J. Pan, and G. Antoniou. Role Forgetting in ALC Ontologies. *In the Proceedings of the Eighth International Semantic Web Conference (ISWC 2009), Washington, DC*, pages 666–681, 2009.

T. Wang, B. Parsia, and J. Hendler. A Survey of the Web Ontology Landscape. *In the Proceedings of the Fifth International Semantic Web Conference (ISWC 2006), Athens, Georgia, USA*, pages 682–694, 2006.

Z. Wang, K. Wang, R. Topor, and J.Z. Pan. Forgetting Concepts in DL-Lite. *In the Proceedings of the Semantic Web Research and Applications: Fifth European Semantic Web Conference (ESWC 2008), Tenerife, Canary Islands, Spain*, page 245, 2008.

T. Weithöner, T. Liebig, M. Luther, and S. Bohm. What's Wrong with OWL Benchmarks? *In the Proceedings of the Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006), Athens, Georgia, USA*, pages 101–114, 2006.

T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. Henke, and O. Noppens. Real-World Reasoning with OWL. *In the Proceedings of the Fourth European Conference on the Semantic Web (ESWC 2007) workshop Research and Applications, Innsbruck, Austria*, pages 296–310, 2007.

F. Wiesman and N. Roos. Domain Independent Learning of Ontology Mappings. *In the Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), New York City, New York, USA*, 2:846–853, 2004.

P. Winoto, G. McCalla, and J. Vassileva. An Extended Alternating-Offers Bargaining Protocol for Automated Negotiation in Multi-Agent Systems. *In the Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002), Edmonton, Canada*, pages 969–970, 2002.

M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

Z. Yang, D. Zhang, and C. Ye. Evaluation Metrics for Ontology Complexity and Evolution Analysis. *In the Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE 2006)*, pages 162–169, 2006.

H. Yao, A.M. Orme, and L. Etzkorn. Cohesion Metrics for Ontology Design and Application. *Journal of Computer Science*, 1(1):107–113, 2005.

B. Yu and M.P. Singh. An Agent-Based Approach to Knowledge Management. *In the Proceedings of the Eleventh International Conference on Information and Knowledge Management (CIKM 2002), ACM New York, NY, USA*, pages 642–644, 2002.