Auton Agent Multi-Agent Syst (2014) 28:337–380
DOI 10.1007/s10458-013-9225-1

Agent-based decentralised coordination for sensor networks using the max-sum algorithm

A. Farinelli · A. Rogers · N. R. Jennings

Published online: 23 May 2013

Abstract In this paper, we consider the generic problem of how a network of physically distributed, computationally constrained devices can make coordinated decisions to maximise the effectiveness of the whole sensor network. In particular, we propose a new agent-based representation of the problem, based on the factor graph, and use state-of-the-art DCOP heuristics (i.e., DSA and the max-sum algorithm) to generate sub-optimal solutions. In more detail, we formally model a specific real-world problem where energy-harvesting sensors are deployed within an urban environment to detect vehicle movements. The sensors coordinate their sense/sleep schedules, maintaining energy neutral operation while maximising vehicle detection probability. We theoretically analyse the performance of the sensor network for various coordination strategies and show that by appropriately coordinating their schedules the sensors can achieve significantly improved system-wide performance, detecting up to 50 % of the events that a randomly coordinated network fails to detect. Finally, we deploy our coordination approach in a realistic simulation of our wide area surveillance problem, comparing its performance to a number of benchmarking coordination strategies. In this setting, our approach achieves up to a 57 % reduction in the number of missed vehicles (compared to an uncoordinated network). This performance is close to that achieved by a benchmark centralised algorithm (simulated annealing) and to a continuously powered network (which is an unreachable upper bound for any coordination approach).

Electronic supplementary material The online version of this article (doi:10.1007/s10458-013-9225-1) contains supplementary material, which is available to authorized users.

A. Farinelli (✉)
Computer Science Department, University of Verona, Verona, Italy
e-mail: alessandro.farinelli@univr.it

A. Rogers · N. R. Jennings
Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK
e-mail: acr@ecs.soton.ac.uk

N. R. Jennings
e-mail: nrj@ecs.soton.ac.uk

Keywords Decentralised coordination · Max-sum · Wide area surveillance · Sensor networks

1 Introduction

Increasing attention is being devoted to applications involving networks of low-power wireless sensing devices that are deployed within an environment in order to acquire and integrate information. Such networks have found application in wide-area surveillance [30], animal tracking [55], and monitoring environmental phenomena in remote locations [19]. A fundamental challenge within all such applications arises due to the fact that the sensors within these networks are often deployed in an ad hoc manner (e.g. dropped from an aircraft or ground vehicle within a military surveillance application), and thus, the local environment of each sensor, and hence the exact configuration of the network, can not be determined prior to deployment. Rather, the sensors themselves must be equipped with the capability to autonomously adapt, sometime after deployment, once the local environment in which they (and their neighbours) find themselves has been determined. Examples of such adaptation include determining the most energy-efficient communication paths within the network once the actual reliability of communication links between individual sensors has been measured in situ [39], dynamically determining the optimal orientation of range and bearing sensors to track multiple moving targets as they move through the sensor network [12], and in the application that we consider in detail in this paper, coordinating the sense/sleep schedules (or duty cycles) of power constrained sensors deployed in a wide-area surveillance task, once the degree of overlap of the sensing fields of nearby sensors has been determined.

A common feature of these autonomous adapting problems is that the sensors must typically choose between a small number of possible actions (e.g. which neighbouring sensor to transmit data to, which target to focus on, or which sense/sleep schedule to adopt), and that the effectiveness of the sensor network as a whole depends not only on the individual choices of action made by each sensor, but on the joint choices of interacting sensors. Thus, to maximise the overall effectiveness of the sensor network, the constituent sensors must typically make coordinated, rather than independent, decisions. For example, in the context of energy-efficient routing, sensors should coordinate to avoid routing all messages through the same agent, thus consuming all its battery power; in the context of target tracking, agents should coordinate to decide on which target to focus, so to have more accurate estimates of the target locations, and, finally, in the context of the energy constrained sensors involved in wide area surveillance, sensors should coordinate their sense/sleep schedules trying to minimise the time during which parts of the environment are left without active sensors. Furthermore, these coordinated decisions must be performed despite the specific constraints of each individual device (such as limited power, communication and computational resources), and the fact that each device can, typically, only communicate with the few other devices in its local neighbourhood (due to the use of low-power wireless transceivers, the small form factor of the device and antenna, and the hostile environments in which they are deployed). Additional challenges arise through the need to perform such coordination in a decentralised manner such that there is no central point of failure and no communication bottleneck, and to ensure that the deployed solution scales well as the number of devices within the network increases.

In more detail, here we focus on a specific problem concerning the autonomous adaptation of sensors within a wireless network deployed in an urban environment to detect vehicle movements on a road network. Within this setting, decentralised coordination for energy

management is a key challenge, and a common conflicting requirement is to maximise the lifetime of the sensor network, while also collecting the maximum amount of information possible. In particular, increasing attention has been devoted to sensor nodes which are able to harvest energy from the environment using multiple sources (such as solar cells or micro generators that can exploit vibrational energy or small temperature differences) in combination [53]. When equipped with sufficient energy harvesting resources, and the ability to model and predict future energy usage and harvesting, such sensors may then control their duty cycle (effectively switching between active sensing modes and low-power sleep modes) in order to operate in an *energy neutral* mode, and hence, exhibit an indefinite lifetime [18]. However, since the sensing ranges of these sensors will typically overlap with one another, the overall effectiveness of the sensor network depends not only on the sensors' individual choice of duty cycles, but also on the combined choice of neighbouring sensors whose sensing ranges overlap. With an ad hoc sensor deployment, these interactions are not known prior to deployment, and thus, we describe how the sensors may auto-adapt by first learning the interactions between their neighbours (i.e. how much their neighbours' sensing fields overlap with their own), and then coordinating their sense/sleep schedules in order to address the system-wide performance goal of maximising the probability that a vehicle is detected.

Problems of this nature can be addressed by representing each sensor as an *autonomous agent* that collaborates with its peers to learn a joint action *policy* [22,16]. This decentralized learning problem can be formalized using a number of different methods. A possible one is the Markov decision problem framework, and more specifically decentralized MDPs (Dec-MDPs) [4] where each agent receives local observations and performs local actions, but the environment evolution and system performance depend on the joint actions of all the agents. However, while these models can be used to find optimal solutions to learning and coordination, their inherent complexity [4] often prevents such techniques from being used for practical applications.[1] To combat this, recent advances in approximate solution techniques for POMDPs [50,35] have been developed and these show that by exploiting problem structure it is possible to scale to a significantly higher number of agents (i.e., hundreds or even thousands of agents). For example, in [50] the authors propose a distributed method to approximately solve POMDPs which is based on the use of Coordination Locales (i.e., system configurations where important interactions among agents take place). By focusing on such locales the authors are able to solve problems involving hundreds of agents. Similarly, in [35] the author shows that it is possible to address problems involving up to a thousand agents by exploiting the locality of interactions, where the key elements are the use of the Factored POMDP model and the use of approximate value functions. While these approaches are very promising, a key element that influences the scaling factor is the locality of interactions and these approaches only scale to large systems when agents' interactions are very sparse. However, in our specific application domain, we consider configurations where agents must coordinate with a large subset of team mates (i.e., up to 64 neighbours in some configurations) and so such approaches do not seem to be suitable.

Hence, here we prefer to focus on the coordination problem and address learning in a separate phase. In more detail, we model our coordination problem with a constraint network [6]. Such networks are often represented by graphs in which the nodes represent the agents (in this case the sensors) and the edges represent constraints that arise between the agents depending on their combined choice of action. Constraints can either be *h*ard (i.e., relations

[1] Problems used to benchmark Reinforcement Learning techniques based on MDPs typically involve a few agents with a few actions, see for example the distributed sensor network problem used in [22] where eight sensors must collaborate to track three targets.

that describe accepted joint assignments of the variables) or *s*oft (i.e., a real valued function that describes cost or reward for each joint variable assignment). When the constraint network includes only *h*ard constraints the associated problem of finding a variable assignment that satisfies all constraints is usually referred to as a distributed constraint satisfaction problem (DCSP). Indeed DCSP approaches have been successfully used to represent the coordination problem of agents involved in target tracking tasks (e.g., [3,11,32]). However, in this paper, we focus on the more general setting of distributed constraint optimization problems (DCOPs), where agents must agree on a variable assignment that maximises (or minimises) the sum of the constraints' values.

To date, a number of algorithms have been proposed for solving DCOPs, and they can be broadly divided into two main categories: exact algorithms that are guaranteed to provide the optimal solution (such as OptAPO [29], ADOPT [33] and DPOP [36]) and approximate algorithms that are typically based upon entirely local computation such as distributed stochastic algorithm (DSA) [12] or maximum gain message (MGM) [28]. Now, while exact algorithms find useful application within large computational systems, they do not address many of the additional challenges that are present when considering low power wireless devices deployed within sensor networks. In particular, all the above exact algorithms calculate the globally optimal solution, and such optimality demands that some aspects of the algorithm grows exponentially in size (because finding an optimal solution for a DCOP is NP-hard problem [33]). Such exponential relationships are simply unacceptable for embedded devices that exhibit constrained computation, bandwidth and memory resources. For example, within the wide area surveillance scenario that motivates our work, the requirement to operate for an indefinite lifetime imposes the use of extremely low-power devices, and thus we are using the Texas Instruments CC2431 System-on-Chip devices as a demonstration platform. This is a low-power device incorporating an IEEE 802.15.4 compliant RF transceiver, 8 kByte RAM, and a 32 MHz 8 bit 8051 micro-controller in a 7×7 mm package (see [49] for further details on the deployment of the max-sum algorithm on this platform). Moreover, most optimal approaches (e.g., DPOP or ADOPT) require some form of preprocessing of the constraint graph (e.g., pseudo-tree arrangement) before executing the algorithm, hence they are not able to quickly react to changes in the constraint network that might be due to the addition/removal of sensors or malfunctioning of the devices. In this respect, we note that there are approaches to efficiently maintain pseudo-trees in the face of changes such as agent addition/removal [47]. However, to obtain a new variable assignment the agents must still re-run the optimization procedure even if only a minimal part of the problem changes.[2] In contrast, approximate algorithms often converge to poor quality solutions, and more importantly, since they can not provide any guarantees on the quality of the solution, empirical good behaviours in specific domains are hard to generalise across different applications.

Against this background, there is a clear need for decentralised coordination algorithms that make efficient use of the constrained computational and communication resources found within wireless networks sensor systems, and yet are able to effectively represent complex interactions among sensors (i.e., interactions that are domain-specific and that may depend on the joint actions of groups of sensors). Moreover, sensors should be able to negotiate over the

[2] A notable exception is the superstabilizing version of DPOP (S-DPOP) proposed in [37], where the authors aim to minimize changes in the optimization protocol when there are low impact failures in the system. Nevertheless, similar to the original DPOP approach, S-DPOP requires agents to exchange large messages (where the size of the messages is exponential in the tree-width of the pseudo-tree arrangement). Hence such an approach would not be feasible in the context of low-power devices that constitute our reference application platform.

best possible actions continuously to quickly adapt to possible changes in the environment (e.g., due, for example, to hardware failures or sensor addition/removal).

To this end, we propose an agent-based decentralised coordination approach based on a DCOP formulation of our wide area surveillance application scenario. Notice that, while several previous approaches in the DCOP community addressed problems related to sensor networks (e.g., [56,25]) most of this work focuses on target tracking/detection. In contrast, here we address the specific problem where sensors have energy constraints, can harvest energy from the environment and aim to schedule their sense/sleep cycles so to achieve energy neutral operation. Hence the DCOP formulation we propose here is significantly different from previous work, both in terms of types of actions that agents can perform and in terms of the constraints that hold among the agents. Moreover, our solution is based on a *factor graph* representation of the sensors' interactions [23] and we investigate the use of the max-sum algorithm [5], an approximate solution technique for decentralised coordination [21,10,46,41] that has been successfully deployed on low-power devices [49], as well as on unmanned aerial vehicles [8].

The max-sum algorithm belongs to the generalised distributive law (GDL) framework [1], which is a unifying framework for inference in graphical models frequently used in information theory, artificial intelligence and statistical physics. In particular, here, we exploit the extensive evidence that demonstrates that GDL techniques generate good approximate solutions when applied to cyclic graphs (in the context of approximate inference through 'loopy' belief propagation on Bayesian networks [34], iterative decoding of practical error correcting codes [26], clustering of large datasets [13], and solving large scale K-SAT problems involving thousands of variables [31]). These algorithms effectively propagate information around the network such that the solution converges to a *neighborhood maximum*, rather than a simple local maximum [54]. Specifically, we apply the max-sum algorithm on a bipartite factor graph, which represents interactions among agents. The use of factor graphs for GDL techniques was introduced by Kschischang et al. [23] and proved to be a very powerful and expressive framework for these techniques.

To summarize, in this paper we make the following contributions to the state of the art:

- We propose a DCOP formulation of our coordination problem, and more precisely, we propose the use of a factor graph representation of the agents' interactions. Specifically, we discuss two possible factor graph mappings, one based on a decomposition of the global optimisation function into the individual agents' utilities, and the other based on the interactions among neighbouring agents. We discuss the limitations and benefits of the two in terms of computational efficiency, as well as responsiveness to changes in the environment. Given the factor graph formulation of our problem, we then use the max-sum algorithm to generate approximate solutions to the general social welfare maximising problem through decentralised local message passing between interacting sensors.
- We formally model our wide area surveillance problem and theoretically analyse the performance of the sensor network in the case of (i) continuously powered, (ii) synchronised, (iii) randomly coordinated, and (iv) optimally coordinated sensors. Our analysis indicates that by appropriately coordinating their sense/sleep schedules, the sensors can achieve a significantly improved system-wide performance, detecting up to 50 % of the events that the randomly coordinated network fails to detect.
- Finally, we exercise our coordination mechanism in a realistic simulation of our wide-area surveillance problem. We empirically evaluate our mechanism within a software

simulation (based on the RoboCup rescue simulation environment[3]) of the scenario. We demonstrate that the sensors are capable of acquiring (through an initial period observing events within the environment) the appropriate information necessary to coordinate their sense/sleep schedules, and that they may then use the max-sum algorithm that we have derived here to do so. Our approach makes no assumptions regarding the sensing fields of the sensors, nor does it require the sensors to know their own location, nor that of the neighbouring sensors with whom they can communicate. By using our approach, we achieve up to a 57 % reduction in the number of missed vehicles (compared to an uncoordinated network), and this performance is shown to be close (on average about 25 %) to that achieved by a benchmark centralised optimisation algorithm (simulated annealing), and to a continuously powered network (within 10 % in the worst case), which represents an unreachable upper bound for any coordination approach.

The remainder of this paper is structured as follows: in Sect. 2 we present our factor graph representation and max-sum decentralised coordination algorithm. We introduce and theoretically analyse the wide area surveillance problem in Sects. 3 and 4 we then apply our approach by computing the degree of overlap with neighbouring sensors (in the absence of any a priori knowledge), and then coordinate with their neighbours to maximise the effectiveness of the overall network to this problem. Finally, we conclude and discuss future work in Sect. 5.

2 The max-sum approach to coordination

The max-sum algorithm is a specific instance of a general message passing algorithm that exploits the GDL in order to decompose a complex calculation by factorising it (i.e. representing it as the sum or product of a number of simpler factors) [1]. In our case, it represents a combination of the best features of the optimal and the approximate stochastic algorithms. It can make efficient use of constrained computational and communication resources, and yet be able to attain close to optimal solutions.

The idea of factoring agent interactions that we exploit in our work has previously been used for action selection in multi-agent systems. Specifically, Guestrin et al. [15] introduce the *coordination graph* framework to provide a tractable planning algorithm in a dynamic multi-agent system, highlighting the underlying relationships between graphical models and influence diagrams. Furthermore, the max-sum algorithm was previously proposed as a decentralised technique for agent coordination to exploit the factorisation model provided by the coordination graph framework. In particular Kok and Vlassis [21,22] propose the use of the max-sum algorithm to compute coordinated action for a group of interacting agents, and to provide a decentralised solution to reinforcement learning.

With respect to this previous work, we propose the use of the factor graph to model the agent interactions and to provide an operational framework that defines the messages exchanged in the max-sum algorithm. The use of the factor graph has several benefits, the most interesting of which is the possibility to *explicitly* represent complex interactions among the agents that can not be captured by binary relationships. In more detail, coordination graphs, as defined in [15], can model k-ary relationships among the agents (with $k > 2$). However, the graphical model they use to represent agent interactions does not represent k-ary relationships *explicitly*, as the mapping between the constraint graph and the constraint network is not one-to-one (i.e., several constraint networks might have the same constraint graph). In particular, a constraint network can be graphically represented, with a one-to-one

[3] see www.robocuprescue.org

mapping between the graphical representation and the constraint network, only by using a hyper-graph (see [6] for further details). In such cases, the factor graph essentially represents a hyper-graph, where factors represent k-ary constraints among variables. This aspect is critical when modelling the interactions among the sensors in the wide area surveillance scenario that we consider here (where the sensing fields of more than two sensors may overlap). In fact, since the factor graph provides not only a representation of the agent interactions, but also a computational framework, the max-sum algorithm specified on the factor graph directly provides an approach to solve problems that comprise such k-ary functions.[4]

Moreover, the use of a factor graph allows us to propose different ways of modelling the coordination problem we face, each favouring different aspects of the solution approach, such as, for example, responsiveness to unexpected changes versus computation effort faced by the agents (see Sect. 2.2 for further details).

The use of a factor graph together with the max-sum algorithm has been already successfully used to coordinate the operation of low-power devices [10] and mobile sensors [46]. Specifically, in [10] max-sum has been evaluated in a benchmarking graph colouring problem and validated on low-power devices, while in [46] it has been employed to coordinate the movement of mobile sensors that must collaborate to acquire accurate information on environmental parameters (e.g., temperature or gas concentration). However, here we focus on a significantly different application domain, hence, with respect to this previous work, the formulation of the problem, as well as the evaluation methodology, are very different.

In the following, we first provide a generic DCOP formalization for multi-agent coordination (Sect. 2.1), we then discuss the factor graph representation that we use to apply the max-sum algorithm, highlighting differences with the standard constraint graph representation frequently used in DCOPs (Sect. 2.2). Next, we provide a pseudo-code description of the operations associated with the max-sum algorithm (Sect. 2.3) as well as an example of its execution (Sect. 2.4). We then discuss the message update schedule for the max-sum algorithm (Sect. 2.5) and the guarantees on convergence and solution quality that it provides (Sect. 2.6). Finally, we present an analysis of the coordination overhead in terms of communication cost and computational complexity (Sect. 2.7).

2.1 Distributed constraint optimization

A standard DCOP formalization of a multi-agent coordination problem is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, where $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a set of agents and $\mathcal{X} = \{x_1, \ldots, x_s\}$ is a set of variables, each variable x_i is owned by exactly one agent a_i, but an agent can potentially own more than one variable. The agent a_i is responsible for assigning values to the variables it owns. $\mathcal{D} = \{D_1, \cdots, D_s\}$ is a set of discrete and finite variable domains, and each variable x_i can take values in the domain D_i. Then, $\mathcal{F} = \{F_1, \ldots, F_n\}$ is a set of functions that describe the constraints among variables. Each function $F_i : D_{i_1} \times \cdots \times D_{i_{k_i}} \rightarrow \Re \cup \{-\infty\}$ depends on a set of variables $\mathbf{x_i} \subseteq \mathcal{X}$, where $k_i = |\mathbf{x_i}|$ is the arity of the function and $-\infty$ is used to represent hard constraints. Each function assigns a real value to each possible assignment of the variables it depends on.

The goal is then to find a variable assignment that maximises the sum of constraints:

$$\arg\max_{\mathbf{x}} \sum_{i} F_i(\mathbf{x_i}) \tag{1}$$

[4] Notice that the analysis and empirical evaluation performed in [21,22] only include pairwise interactions.

DCOPs are usually graphically represented using an *interaction graph*, where variables are represented as circles and an edge between two variables indicates that the two variables participate in a constraint. For ease of presentation, and following a common assumption in the DCOP literature, we assume that each agent controls exactly one variable.

To further clarify this concept, we show in Fig. 1a an example in which three sensors, $S1$, $S2$, $S3$, interact through a common overlapping area of their sensing range. While we will detail the associated coordination problem in Sect. 3, for now let us consider that agents must coordinate their actions to maximise event detection in overlapping areas. Therefore, in a standard DCOP formalization, constraints connect sensors that share an overlapping area; Fig. 1b shows the corresponding interaction graph.

In the following we detail a factor graph representation for the same sensor configuration to clarify the differences between the two.

2.2 Factor graph representation

A factor graph is a bipartite graph comprising two types of nodes: variable nodes (usually depicted as circles) and function nodes (usually depicted as squares) [5,23]. Undirected links connect each function to the variables it depends on. The factor graph is a widely used graphical representation of factored functions, e.g. functions that can be expressed as a sum of components such as the function reported in Eq. 1. A factor graph explicitly represents the relationships among variables though the functions nodes. This is in contrast to the constraint graph discussed above where two variables are connected via an edge if they participate in some constraints.

To further clarify this concept consider Fig. 1c, where we show a factor graph representation of the agent interaction configuration in 1a. Notice that in this last graphical representation both the binary and the ternary constraints are explicitly represented, while the edges in Fig. 1b only indicate that the three variables share some constraints among them.

Now, there are many ways of modelling the coordination problem represented by Eq. 1 with a factor graph, as all we need to ensure is that the sum of the functions that we choose to represent the agents' interactions is equivalent to the objective function expressed by Eq. 1. In other words, we can use various decompositions for a given problem setting, and thus we can have several distinct factor graph representations of the same problem. Moreover, the use of different factor graphs impacts on the computation performed by the max-sum algorithm. In particular, here we focus on two important classes of possible factor graph modelling that have been previously used for the deployment of the max-sum algorithm in the context of decentralized coordination: the first one is based on the functions that describe the direct interactions among the sensors, (i.e., the interaction-based factor graph), and has been used for example in [38,7]; the other is based on the utility that each sensor receives depending on the variables' assignment of its neighbours (i.e., the utility-based factor graph), and has been used for example in [10,46].

2.2.1 Interaction-based factor graphs

In the interaction-based factor graph, functions represent interactions of neighbouring agents. This can be considered as a direct translation of the constraint graph to a factor graph where we simply introduce one function node for each constraint and, as above, we have one variable for each sensor.

For example, consider the situation depicted in Fig. 2, where Fig. 2a shows three sensors interacting with their immediate neighbours through pairwise overlaps of their sensing areas,

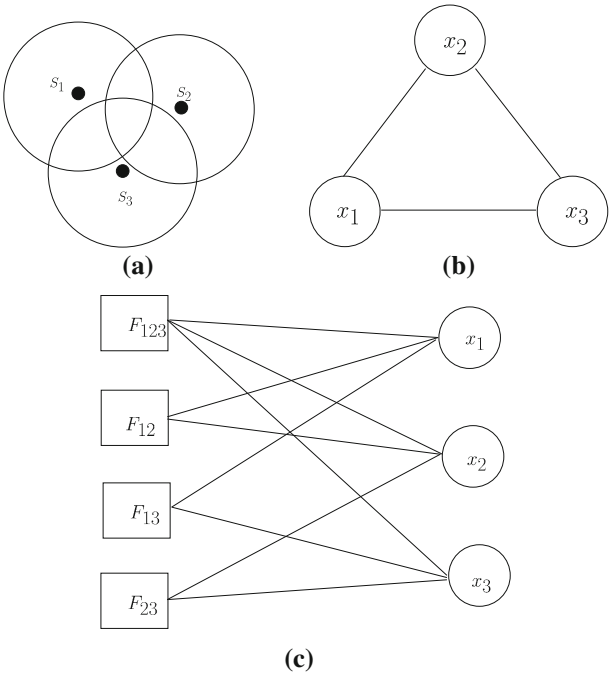

Fig. 1 A diagram showing **a** three sensors and their overlapping areas, **b** the corresponding constraint graph, and **c** a factor graph representation

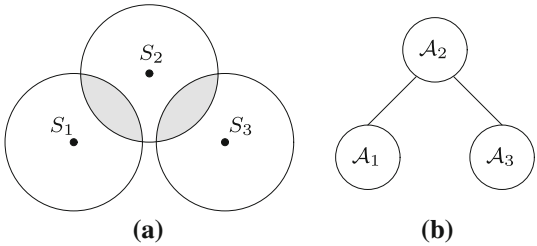

Fig. 2 A diagram showing **a** the position of three sensors in the environment whose sensing ranges overlap, and **b** the agent constraint graph

while Fig. 2b shows its corresponding constraint graph. The corresponding interaction based factor graph is reported in Fig. 3a, where function $F_{12}(x_1, x_2)$ and function $F_{23}(x_2, x_3)$ directly represent the constraints that hold between the agents.

Notice that, the max-sum algorithm requires both variable and function nodes to perform computation for updating messages (see Sect. 2.3), hence each variable and function node must be allocated to one agent that is responsible to perform such computation. The allocation of variables to agents is straightforward because each sensor has as corresponding variable and hence the agent that is responsible for that sensor will control the corresponding variable. In contrast, in the interaction-based factor graph the allocation of function nodes to agents is not clear because there are function nodes the are shared between different variables. Therefore this representation requires a previous negotiation phase between the agents to decide which agent is in charge of the shared functions. This negotiation phase could be theoretically implemented in a very simple and straightforward way, because the content of max-sum messages does not depend on which agent performs the computation, therefore

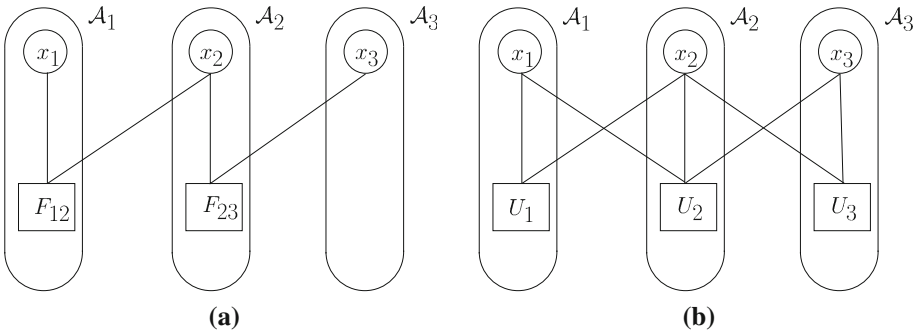

Fig. 3 Two different factor graph representations for the problem instance reported in Fig. 2 : Interaction-based (**a**) and Utility-based (**b**)

any allocation policy could be applied. For example, in Fig. 3a the agent with the lowest id among all agents sharing the function is the one that is deemed to be in charge for performing the computation of that function.[5]

2.2.2 Utility-based factor graphs

In the utility-based factor graph, the objective function must be appropriately decomposed so that each individual function represents the utility of one agent and the sum of the agent's utilities corresponds to the objective function. Such a decomposition is domain specific and a suitable decomposition for our wide area surveillance application will be detailed in Sect. 3.

To further clarify the basic ideas behind the utility-based factor graph, consider again the situation depicted in Fig. 2a. We can build a utility-based factor graph that represents this situation by adding one variable per sensor representing its possible sleep/sense schedule, and one function per sensor representing its individual utility. Next, for each sensor, we connect its utility function with its own variable and with all the variables of its neighbours (i.e., the sensors that exhibit an overlapping area with it). For example, focusing on sensor S_2, we connect the function node representing U_2 with variables x_2 (the sensor's variable) and with variables x_1 and x_3 (neighbours' variables). The resulting factor graph is shown in Fig. 3b. The overall function represented by this factor graph is given by $U = U_1(x_1, x_2) + U_2(x_1, x_2, x_3) + U_3(x_2, x_3)$ which is the social welfare function for the system.

Notice that by using this formalization there is a clear allocation of variables and function nodes to agents. In other words, each agent is responsible for deciding the allocation of its own variable, for receiving messages for its function and variable nodes and for updating the messages that flow out of its function and variable nodes. In this way, agents can negotiate over the best possible actions continuously, thus being able to quickly react to possible changes in the environment.

On the other hand, this formalization is not efficient in terms of the computation that each agent must perform. This is because: (i) it results in functions which have an increased number of arguments with respect to the original constraint graph and (ii) it can create loops

[5] Notice that, while the content of the max-sum messages will not change the load balance among the agents will be different depending on the strategy used to allocate the functions. However this issue is outside the scope of the current paper and we refer the interested reader to [44] where computational tasks related to GDL algorithms are allocated to agents explicitly considering their computation and communication capabilities.

in the factor graph which are not present in the corresponding constraint graph. The latter is an important issue as max-sum is known to be optimal on acyclic factor graphs but provides no general guarantees on optimality when cycles exist (see Sect. 2.3 for further details). For example, confronting Fig. 2b, which shows the constraint graph corresponding to our exemplar situation, with Fig. 3b it is clear that in the constraint graph there are only binary constraints between the agents but function U_2 in the factor graph is a ternary function. Moreover, the constraint graph is acyclic, while the factor graph is not. Despite this, the utility based factor graph is indeed a good choice for representing our problem because it allows agents to start running the coordination procedure as soon as they discover which are their neighbours. This is in contrast to the interaction based factor graph, which requires in general some form of negotiation, before running the max-sum algorithm, to decide which agent is responsible for the computation associated to shared functions. Hence, the utility based factor graph is well suited for dynamic environments where neighbours can change over time (e.g., due to hardware failures or sensor addition/removal).

To summarise, the choice of the factor graph representation clearly depends on application specific requirements. Since here we wish to allow sensors to quickly react to possible changes in the environment, in the rest of the paper we will use the *utility-based* factor graph representation.

2.3 The max-sum algorithm

The max-sum algorithm operates directly on the factor graph representation described above. When this graph is cycle free, the algorithm is guaranteed to converge to the global optimal solution such that it finds the joint assignment that maximises the sum of the agents' utilities. In general, there can be multiple assignments that provide the optimal solution, in this case agents have to perform an extra value propagation phase (as in DPOP) to be sure they achieve the optimal assignment. Another approach, often used to avoid this extra coordination phase, is to break the symmetry of the problem by artificially inserting a small random preference for each agent on the values of their domain [10], while this works well in practice there are no guarantees that agents will coordinate on the optimal assignment without the value propagation phase.

When applied to cyclic graphs (as is often the case in real settings when using the utility-based factor graph representation), there is no guarantee of convergence, but extensive empirical evidence demonstrates that this family of algorithms generate good approximate solutions [23,27].

Specifically, the max-sum algorithm proceeds by iteratively passing messages from variables to functions, and from functions to variables. These messages are defined as follows:

– **From variable to function:**

$$q_{i\to j}(x_i) = \alpha_{ij} + \sum_{k\in\mathcal{M}_i\setminus j} r_{k\to i}(x_i) \tag{2}$$

where $\mathcal{M}_i$ is a vector of function indexes, indicating which function nodes are connected to variable node i, and α_{ij} is a scaler chosen such that $\sum_{x_i} q_{i\to j}(x_i) = 0$,[6] in order to normalise the message and hence prevent them increasing endlessly in the cyclic graphs that we face here.

[6] As stated in [41] this normalisation will fail in the case of a negative infinity utility that represents a hard constraint on the solution. However, we can replace the negative infinity reward with one whose absolute value is greater than the sum of the maximum values of each function.

– **From function to variable:**

$$r_{j\to i}(x_i) = \max_{\mathbf{x}_j \setminus i} \left[U_j(\mathbf{x}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k\to j}(x_k) \right] \tag{3}$$

where $\mathcal{N}_j$ is a vector of variable indexes, indicating which variable nodes are connected to function node j and $\mathbf{x}_j \setminus i \equiv \{x_k : k \in \mathcal{N}_j \setminus i\}$.

The messages flowing into and out of the variable nodes within the factor graph are functions that represent the total utility of the network for each of the possible value assignments of the variable that is sending/receiving the message. At any time during the propagation of these messages, agent i is able to determine which value it should adopt such that the sum over all the agents' utilities is maximised. This is done by locally calculating the function, $z_i(x_i)$, from the messages flowing into agent i's variable node:

$$z_i(x_i) = \sum_{j \in \mathcal{M}_i} r_{j\to i}(x_i) \tag{4}$$

and hence finding $\arg\max_{x_i} z_i(x_i)$.

Thus, although the max-sum algorithm is approximating the solution to a global optimisation problem, it involves only local communication and computation. Moreover, notice that, in most previous applications, the max-sum algorithm was used as a centralised optimisation technique (e.g., for efficient iteratively decoding of error correcting codes [26]). In our setting the factor graph is actually physically divided among the sensors within the network, and thus the computation of the system-wide global utility function is carried out through a distributed computation involving message passing between agents.

Algorithm 1 max-sum

```
1: Q ← ∅ {Initialize the set of received variable to function message}
2: R ← ∅ {Initialize the set of received function to variable message}
3: while termination condition is not met do
4:   for j ∈ N_i do
5:     r_{i→j}(x_j) = computeMessageToVariable(x_j, U_i, Q)
6:     SendMsg(r_{i→j}(x_j), a_j)
7:   end for
8:   for j ∈ M_i do
9:     q_{i→j}(x_i) = computeMessageToFunction(x_i, U_j, R)
10:    SendMsg(q_{i→j}(x_i), a_j)
11:  end for
12:  Q ← getMessagesFromFunctions()
13:  R ← getMessagesFromVariables()
14:  x_i^* = updateCurrentValue(x_i, R)
15: end while
```

In more detail, Algorithm 1 reports a pseudo-code description of the operations that each agent performs to implement the max-sum algorithm. At each execution step, each agent computes and sends the variable to function and function to variable messages. Such messages depend on the receiver variable (or function) and are computed according to Eqs. 2 and 3 respectively. Algorithms 2 and 3 report a pseudo-code description of the operations

Algorithm 2 computeMessageToVariable(x_j, U_i, **Q**)

Input: x_j: the receiver's variable, U_i: the sender's function, **Q**: the current set of variable to function messages received by the sender.
Output: $r_{i\to j}(x_j)$ the function to variable message from function U_i to variable x_j.
1: $r_{i\to j}(x_j) = -\infty$
2: **for** $\mathbf{d}_i \in \mathbf{D}_i$ {all joint assignments of $\mathbf{x}_i$} **do**
3: $\sigma = U_i(\mathbf{d_i})$
4: **for** $d_k \in \mathbf{d_j}$, $(k \neq j)$ **do**
5: $\sigma = \sigma + q_{k\to i}(d_k)$ $\{q_{k\to i} \in \mathbf{Q}\}$
6: **end for**
7: $r_{i\to j}(d_j) = \max r_{i\to j}(d_j), \sigma$ $\{d_j \in \mathbf{d_i}\}$
8: **end for**
9: **return** $r_{i\to j}(x_j)$

Algorithm 3 computeMessageToFunction(x_i, U_j, **R**)

Input: x_i: the sender's variable, U_j: the receiver's function, **R**: the current set of function to variable messages received by the sender.
Output: $q_{i\to j}(x_i)$ the variable to function message from variable x_i to function U_j.
1: $q_{i\to j}(x_i) = \mathbf{0}$
2: **for** $r_{k\to i} \in \mathbf{R}$ $k \neq j$ **do**
3: $q_{i\to j}(x_i) = q_{i\to j}(x_i) + r_{k\to i}(x_i)$
4: **end for**
5: $\alpha_{ij} = -\frac{\sum_{d_i \in D_i} q_{i\to j}(d_i)}{|D_i|}$
6: $q_{i\to j}(x_i) = q_{i\to j}(x_i) + \alpha_{ij}$
7: **return** $q_{i\to j}(x_i)$

required to compute such messages.[7] The agent updates the incoming **Q** and **R** messages and then update its current value by computing the variable assignment that maximises function $z_i(x_i)$. Notice that, the value of the agent variable does not have any influence on message computation. Therefore, if the termination condition does not depend on this value, line 14 could be taken out of the main while loop without affecting the final assignment computation. Here, we stop the max-sum algorithm after a fixed amount of executions of the while loop, hence we could compute the assignment outside the while loop, but we prefer to provide this more general version of the pseudo-code.

2.4 Worked example

Figure 4 shows a subset of the messages that are exchanged when the max-sum algorithm is executed on the factor graph shown in Fig. 3b. Note that, with both the utility-based and the interaction-based factor graph representation, the nodes of the factor graph are distributed across the various agents within the system, and, as such, some messages are internal to a single agent (e.g., the message $q_{2\to 2}(x_2)$) while other messages are sent between agents (e.g., $q_{2\to 3}(x_2)$). To better illustrate the functioning of the max-sum algorithm we can consider the computation of a sample variable to function message, $q_{2\to 3}(x_2)$, and a sample function to variable message, $r_{2\to 1}(x_1)$.

[7] This pseudo-code description is based on the procedures for max-sum message computation presented in [7]

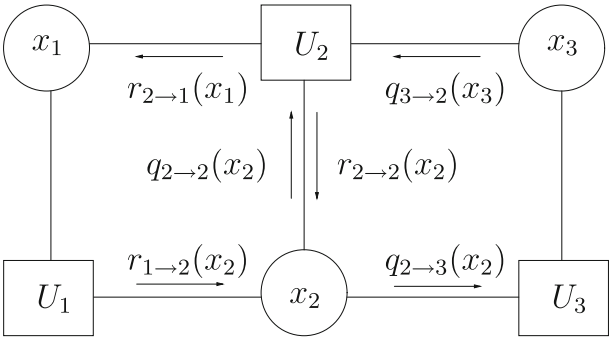

Fig. 4 Subset of the messages exchanged over the factor graph using the max-sum algorithm

We first consider the variable to function message $q_{2\to3}(x_2)$, and for ease of exposition we do not consider the scaler α_{23}. Thus, following Eq. 2, message $q_{2\to3}(x_2)$ is given by:

$$q_{2\to3}(x_2) = \sum_{k\in\mathcal{M}_2\setminus 3} r_{k\to2}(x_2)$$

Considering that in our case $\mathcal{M}_2 \setminus 3 = \{1, 2\}$, we can expand the summation to obtain:

$$q_{2\to3}(x_2) = r_{1\to2}(x_2) + r_{2\to2}(x_2)$$

Therefore, for variable to function messages each agent only needs to aggregate the information received from all the neighbouring functions, without considering the receiver of the message (i.e., function node of agent 3 in our case). The aggregation is performed simply by summing the messages. Notice that messages in the max-sum algorithms do not contain a single assignment value, but contain one real value for each possible variable assignment. In our case, since the domain of the variables are discrete, we represent each message as a vector with $|D_i|$ components. Then the above summation can be directly implemented as a component-wise sum of the vectors as Algorithm 2 illustrates. This is possible because all the messages refer to the same variable. The scaler α_{23} is computed so to have $\sum_{d_i\in D_2} q_{2\to3}(d_2) = 0$, as Line 5 of Algorithm 2 shows.

We now turn to the computation involved with a message sent from a function to a variable and in particular we focus on message $r_{2\to1}(x_1)$ from Eq. 3 we have:

$$r_{2\to1}(x_1) = \max_{\mathbf{x}_2\setminus 1}\left[U_2(\mathbf{x}_2) + \sum_{k\in\boldsymbol{\mathcal{N}_2}\setminus 1} q_{k\to2}(x_k)\right]$$

Considering that in our case $\mathcal{N}_2 \setminus 1 = \{2, 3\}$ and that $\mathbf{x}_2 \setminus 1 = \{x_2, x_3\}$ we then have:

$$r_{2\to1}(x_1) = \max_{x_2,x_3}\left[U_2(x_1, x_2, x_3) + q_{2\to2}(x_2) + q_{3\to2}(x_3)\right]$$

Therefore, for function to variable messages each agent needs to maximise a function which results from the summation of its utility and the messages received from neighbouring variables. Again, since in our case, the variables are discrete, functions can be represented as tables. Algorithm 3 reports the pseudo-code description of the operations required for the message computation. Notice that the for loop in line 2 iterates over all the joint assignments of variables in $\mathbf{x}_i$ (i.e., $\mathbf{x}_2 = \langle x_1, x_2, x_3\rangle$ in our example). Note that, this results in a number

Fig. 5 Computation for message $r_{2\to 1}(x_1)$

x_1	x_2	x_3	$U(x_1\ x_2\ x_3)$	$q_{2\to 2}(x_2)$	$q_{3\to 2}(x_3)$	Sum	$r_{2\to 1}(x_1)$
0	0	0	−3	−1	−1	−5	−1
0	1	0	−2	0	−1	−3	−1
1	0	0	−2	−1	−1	−4	0
1	1	0	−1	0	−1	−2	0
0	0	1	−2	−1	0	−3	−1
0	1	1	−1	0	0	−1	−1
1	0	1	−1	−1	0	−2	0
1	1	1	0	0	0	0	0

of iterations that is exponential in the number of neighbours that each agent has. However, this is typically much less than the total number of agents within the system.

An alternative way to describe the above computation is to join the tables representing $U_j(\mathbf{x}_j)$ with all the incoming messages and then sum the corresponding rows. A join operation here is required because the function we are summing over are defined on different variables. Similarly, the maximisation can be implemented by projecting the columns corresponding to the variables we are maximising over (e.g., variables x_2 and x_3 in our example) and removing duplicate rows that have lower values. Figure 5 shows an exemplar computation of message $r_{2\to 1}(x_1)$ using the method described above, the projection operation is illustrated by deleting the columns that refer to x_2 and x_3. The last column reports the maximum value in the *Sum* column for each of the possible values of x_1.

Finally notice that, while here we focus on applications where each agent action can be represented as a discrete variable, the max-sum algorithm can be used also in the case of continuous variable. In that case however, the operations performed by the agents must be carefully adapted; we refer the interest reader to [45] where the max-sum algorithm is applied to a multi-agent system with continuously valued variables and piecewise linear functions.

2.5 Message update schedule

The messages described above may be randomly initialised, and then updated whenever an agent receives an updated message from a neighbouring agent; there is no need for a strict ordering or synchronisation of the messages. In addition, the calculation of the marginal function shown in Eq. 4 can be performed at any time (using the most recent messages received), and thus, agents have a continuously updated estimate of their optimum assignment.

The final state of the algorithm depends on the structure of the agents' utility functions, and, in general, three behaviours can be observed:

1. The preferred joint assignment of all agents converges to fixed values that represent either the optimal solution, or a solution close to the optimal, and the messages also converge (i.e. the updated message is equal to the previous message sent on that edge), and thus, the propagation of messages ceases.
2. The preferred joint assignment converges as above, but the messages continue to change slightly at each update, and thus continue to be propagated around the network.
3. Neither the preferred joint assignment, nor the messages converge and both display cyclic behaviour.

Thus, depending on problem being addressed, and the convergence properties observed, the algorithm may be used with different termination rules:

1. Continue to propagate messages until they converge, either changing the value assignment of the agent continuously to match the optimum indicated, or only after convergence has occurred.
2. Propagate messages for a fixed number of iterations per agent (again either changing the value assignment of the agent continuously or only at termination).

Notice that, both the above rules require only local information for the agents, i.e., each agent only needs to check whether the assignment for the variable it controls changed, whether the new messages it should send differ from the previous messages, or simply count the number of executions of the message update steps. The first termination rule favours the quality of the solution. When the algorithm converges, it does not converge to a simple local maximum, but to a neighbourhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space [54]. Depending on the structure of the factor graph, this neighbourhood can be exponentially large. For practical applications the second termination rule is often preferred. In fact, empirical evidence shows that the max-sum algorithm reaches good approximate solutions in few iterations. Finally, notice that for dynamic scenarios in which the utilities of the agents or the interactions of the agents change over time (perhaps due to sensor failures or additions), the max-sum algorithm can run indefinitely without any termination rule; the agents can decide at every cycle which value to choose based on Eq. 4, and operate on a continuously changing coordination problem. In this way changes that affect the problem configuration will be directly reflected by a change in messages exchanged and thus will be considered by the agents in their assignment decision. Notice that, such continuous behaviour can be obtained also by using other coordination approaches such as DSA. Precisely assessing the merits of max-sum with respect to DSA (or other similar coordination approaches) in such dynamic settings requires however further investigations that fall outside the scope of the present contribution.

2.6 Guarantees on convergence and solution quality

As previously mentioned, empirical evidence shows that GDL-like algorithms are able to provide very good approximate solutions for large scale problems with complex structure [31,13]. Nonetheless, providing theoretical guarantees on convergence and quality of provided solutions for GDL-like algorithms is still an open area of investigation. In particular, theoretical guarantees can only be provided regarding the quality of solutions when the algorithm has converged [54], and guarantees of convergence can only be provided for specific graph topologies, which typically only contain a single loop [54,51]. Notice that, this is clearly not the case in our settings as the factor graph representation reported in Fig. 3b shows.

Since convergence and solution quality are dependent on the structure of the factor graph, we can obtain structures which are more likely to converge and increase the quality of the approximate solutions obtained, by modifying the factor graph. In particular, we can perform transformations that are similar to those used in graphical models to build junction trees; these involve stretching variables, and clustering both variables and functions [23]. These transformations do not change the problem being solved (i.e., the new factor graph represents the same global function as before), however, by applying a sequence of such transformations, all the loops may be removed from the factor graph, with the result that

the max-sum algorithm is then guaranteed to converge to the optimal solution. However, in general, this process incurs an exponential increase in computation cost and message size.

In this respect, the GDL framework on which we build, subsumes the DPOP algorithm; it provides the same guarantees on solution quality and convergence, with the same complexity in terms of message size and computation [52]. However, we can exploit the generality of the GDL framework by applying the above mentioned transformations only to critical portions of the factor graph that exhibit many loops. In this way, we can perform a principled trade-off between solution quality and computation/communication overhead involved in running the algorithm. A full discussion of this extension to the algorithm is beyond the scope of this paper. However, a concrete example can be found in [10], where this idea was exploited in the context of graph colouring by modifying the utility function of each agent to explicitly consider constraints among its neighbours. This effectively corresponds to cluster function nodes that form cliques and result in better performance in graphs with many small loops. Moreover, we can obtain bounded approximations of the optimal solution by using the bounded max-sum (BMS) approach [41]. The main idea behind BMS is to optimally solve an acyclic, relaxed version of the original factor graph where some of the dependencies between variables and functions have been ignored. By carefully choosing which dependencies should be removed BMS can efficiently provide accurate bounds of the optimal solution.

2.7 Coordination overhead

As mentioned before, for our target application we favour the use of the utility-based factor graph representation. When using such a representation, each agent controls one variable (x_i) and one function ($U_i(\mathbf{x}_i)$). Thus, as detailed in Algorithm 1, each agent sends two messages to each neighbour at each execution step: one message from the agent variable x_i to the neighbour function $U_j(\mathbf{x}_j)(q_{i\to j}(x_i))$ that depends on the sender variable x_i, and one message from the agent function $U_i(\mathbf{x}_i)$ to the neighbour variable x_j $(r_{i\to j}(x_j))$ that depends on the receiver variables. Since in our application variables are discrete, each of these messages is a vector that contains a number of values which equals the possible values of the variable's domain $d_i = |D_i|$. Summarizing the number of values communicated by the each agent at each execution step can be expressed as $O(kd)$ where k is the number of neighbours for the agent and d is the maximum cardinality of the variables' domains.

The computational complexity of the local optimization procedure is dominated by the computation of the messages from functions to variables which require a maximization step that, following Algorithm 3, must go through all the joint assignments of the neighbours. Again, given our utility-based factor graph representation each function directly depends on all the neighbours and on the agent, hence the computational complexity of the maximization step can be expressed as $O(d^{k+1})$. Notice that, the computational effort associated with the message update computation could be significantly reduced by an advanced maximization procedure that consider the natural decomposition of the utility function into a sum of smaller factors. For example, consider the factor graph in Fig. 4 and the computation of message $r_{2\to 1}(x_1)$, the function of three variables $U_2(x_1, x_2, x_3)$ can be decomposed into a sum of two functions that depends on two variables each: $U_2(x_1, x_2, x_3) = F_{12}(x_1, x_2) + F_{23}(x_2, x_3)$. This decomposition can be exploited by optimization techniques such as for example, Cluster Tree Elimination or Bucket Elimination which are known to be powerful techniques for decomposable functions [6]. Such decomposition of the utility function is directly exploited by the interaction-based factor graph representation. For example, the interaction-based factor graph shown in Fig. 3a does not contain any ternary constraints, while the corresponding utility based factor graph shown in Fig. 3b includes function $U_2(x_1, x_2, x_3)$. Following this

approach, the computational complexity associated with the message update is dominated by $O(d^z)$ where z is the maximum arity of the functions that agent i is controlling. In most applications, and in our wide area surveillance scenario, z can be significantly smaller than the number of neighbours of the agents. While, the representation based on the interaction-based factor graph would result in a significant reduction of computation as mentioned before, here we focus on the utility-based representation because in this formulation each agent has a clear responsibility over functions to control and therefore each agent can update messages and perform the coordination phase without the need of any pre-processing phase (e.g., pseudo-tree building or any means of allocating functions to agents). This benefit can not be easily quantified with an empirical evaluation as it is not directly related to a measurable performance metric. Nonetheless, it is an important aspect when developing our system.

3 The wide area surveillance problem

Having presented our coordination approach, we now focus on its application to an illustrative wide area surveillance problem for a sensor network. Thus, in this section, we describe and analyse a formal model of the problem that we address, in order to calculate an upper bound on the increase in system-wide performance that can be accrued through coordination. In performing this analysis, we make a number of simplifying assumptions (common in previous work in this field [17]) which we subsequently relax in Sect. 4 where we show how agents can compute the information required for coordination through an initial period observing events within the environment, and can then use the max-sum algorithm to perform the decentralised coordination.

3.1 Problem description

Following the utility-based factor graph formulation reported in Sect. 2.2.2, our problem formulation includes M sensors, where each sensor is controlled by a specific agent. Each agent has control over a discrete variable x_i that represents the possible schedules of the sensor. Each agent interacts locally with a number of other agents such that we can define a set of local functions, $F_i(\mathbf{x}_i)$, that depend on the schedules of a subset of the agents (defined by the set $\mathbf{x}_i$). In particular, in our wide area surveillance problem, the subsets of interacting agents are those whose sensor sensing areas overlap, and the utility describes the probability of detecting an event within the sensor's sensing range.

Within this setting, we wish to find the joint schedule for all the sensors, $\mathbf{x}^*$, such that the sum of the individual agents' utilities (i.e., the social welfare) is maximised:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{i=1}^{M} U_i(\mathbf{x}_i) \tag{5}$$

Furthermore, in order to enforce a truly decentralised solution, we assume that each agent only has knowledge of, and can directly communicate with, the few neighbouring agents on whose assignment its own utility directly depends.

3.2 Theoretical model

We assume that multiple sensors are deployed according to a Poisson process with rate per unit area λ_s (i.e. within a unit area we expect to find λ_s sensors). The use of Poisson processes

to describe these events is common within the literature [17], and represents a generic, non-domain specific model of a random sensor deployment. Each sensor has a circular sensing field, with radius r, and is tasked with detecting transient events within its sensing field. We make no assumptions regarding the process by which events occur, and we consider a general case in which events may have a limited duration in which they remain detectable after their initial appearance. Note that our model is not limited to uniformly distributed events in space or time, as long as we have no prior belief as to when and where events may occur. Event duration is described by another Poisson process with rate per unit time λ_d. Thus, the probability of an event lasting time t is given by $\lambda_d e^{-\lambda_d t}$.

We assume that the sensors are able to harvest energy from their local environment, but at a rate that is insufficient to allow them to be powered continually. Thus at any time a sensor can be in one of two states: either sensing or sleeping. In the sensing state the sensor consumes energy at a constant rate, and is able to interact with the surrounding environment (e.g. it can detect events within its sensing field and communicate with other sensors). In the sleep state the sensor can not interact with the environment but it consumes negligible energy. To maintain energy neutral operation, and thus exhibit an indefinite lifetime, sensors adopt a duty cycle whereby within discrete time slots they switch between these two states according to a fixed schedule of length L. We denote the schedule of sensor i by a vector $\mathbf{s}_i = \{s_0^i, \ldots, s_{L-1}^i\}$ where $s_k^i \in \{0, 1\}$, and $s_k^i = 1$ indicates that sensor i is in its active sensing state during time slot k (and conversely, it is sleeping when $s_k^i = 0$). We assume that this schedule is repeated indefinitely, and in this paper, we specifically consider schedules in which the sensor is in its sense state for one time slot, and in a sleep state for all $L-1$ other time slots (i.e. $\sum_{k=0}^{L-1} s_k^i = 1$). For example, considering $L = 2$ there would be only two possible schedules for sensor i: $\{1, 0\}$ and $\{0, 1\}$. This represents the simplest description of a power constrained sensing schedule, however, given our model of event duration and our assumption of having no prior beliefs on event occurrence, considering only this type of sensing schedule seems a reasonable assumption. In fact, it essentially tries to minimise the off time between two activations of one sensor thus reducing the probability of missing an event in the area that is covered only by that sensor. Nonetheless, we note that the theoretical analysis that we perform, and the max-sum coordination algorithm that we have presented in the last section, can be applied for any discrete schedule. Notice that, if we do not have any constraints on the number of time steps in which the sensor can be in a sensing state the variable domains will be exponential in L, to represent all possible combinations of sense/sleep states. Therefore, while we can still use max-sum to address this problem, the associated computational effort might become prohibitive. This however would be a problem for most DCOP techniques, for example, even a very simple and computationally cheap approach such as DSA or MGM would incur an exponential element in the local optimization step that depends on the size of the variables' domains (see Sect. 4.3 for further details on the DSA approach we use here).

3.3 The coordination problem

Figure 6 illustrates the coordination problem that results from this scenario. In this specific example, three sensors, $\{S_1, S_2, S_3\}$, are randomly deployed and exhibit overlapping sensing fields. In order to maintain energy neutral operation, each sensor can only actively sense for half of the time (i.e. $L = 2$), and thus, each sensor has a choice from two sensing schedules: either $\{1, 0\}$ or $\{0, 1\}$.

The system-wide goal is to maximise the probability that events are detected by the sensor network as a whole. This is achieved by ensuring that the area covered by the three sensors is actively sensed by at least one sensor at any time. However, with the sensing schedules

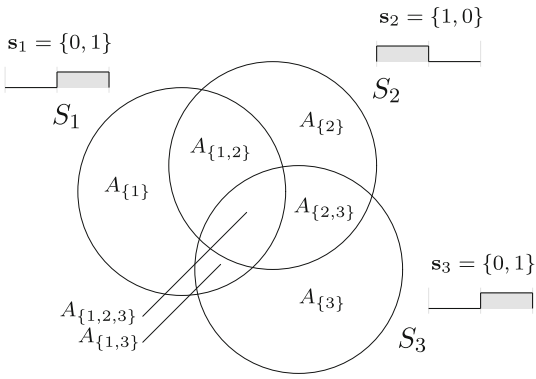

Fig. 6 Example coordination problem in which three sensors, $\{S_1, S_2, S_3\}$, have sensing fields that overlap

available, it is clearly not possible to ensure that area $S_1 \cap S_2$, area $S_2 \cap S_3$ and area $S_1 \cap S_3$ are all sensed continually. Thus, the sensors must coordinate to ensure that the minimal area possible exhibits the minimal periods during which no sensor is actively sensing it. In this case, the optimal solution is the one shown where $\mathbf{s}_1 = \{0, 1\}$, $\mathbf{s}_2 = \{1, 0\}$ and $\mathbf{s}_3 = \{0, 1\}$. Note that this leads to areas $A_{\{1,2\}}$, $A_{\{2,3\}}$ and $A_{\{1,2,3\}}$ being sensed continually, and the smallest area, $A_{\{1,3\}}$, and of course the three non-overlapping areas, exhibiting intermittent sensing.

In a larger sensor deployment, each of these three sensors is also likely to overlap with other sensors. Thus, finding the appropriate sensing schedule of each sensor, such that the probability of detecting an event is maximised, is a combinatorial optimisation problem. As such, this problem is similar to the graph colouring problem commonly used to benchmark DCOP algorithms (see [33] for example). However, an important difference is that in our sensor scheduling problem we can have interactions between multiple sensors (as is the case in the example shown in Fig. 6), rather than interaction between just pairs of sensors (as is the case in the standard graph colouring problem).

3.4 Theoretical analysis

Given the model described above, we now quantify, through a theoretical analysis, the gain in performance that coordination can yield. To this end, we consider four specific cases:

- **Continuously Powered Sensors**:
  We initially ignore the energy constraints of the sensors and assume that they remain in their sensing state continuously. This represents an absolute upper bound on the performance of the network.
- **Synchronised Sensors**:
  We assume that the sensors are limited to sensing for just one in every L time slots, and that the choice of which time slot to use is identical for all sensors; thus sensors in this case exhibit no adaptation.
- **Randomly Coordinated Sensors**:
  As above, we assume sensors are limited to sensing for just one in every L time slots, but the choice of which time slot to use is made randomly by each individual sensor with no coordination with nearby sensors.
- **Optimally Coordinated Sensors**:
  We again consider sensors limited to sensing for just one in every L time slots, but we

consider that they are able to optimally coordinate the choice of sensing time slot with neighbouring sensors whose sensing fields overlap with their own.

In each case, we calculate the *probability of event detection*, PED, (i.e., the probability that any event that occurs within the environment is indeed detected by the network).

3.4.1 Continuously powered sensors

If we assume that the sensors remain continuously in their sense state then an event will be detected if it occurs within the sensing field of at least one sensor. Given the Poisson process that describes the deployment of sensors, the probability that an event falls within the sensing field of m sensors is given by $(\lambda_s \pi r^2)^m \mathrm{e}^{-\lambda_s \pi r^2}/m!$. Thus, an event will be detected in all cases that $m > 0$, and thus, the overall probability of event detection is given by:

$$PED_{\text{continuous}} = 1 - \mathrm{e}^{-\lambda_s \pi r^2} \tag{6}$$

Clearly, increasing either the density of the sensor, λ_s, or the sensing field of the sensors, r, increases the probability with which events are detected.

3.4.2 Synchronised sensors

If the sensors are energy constrained and use a synchronised sensing schedule in which all sensors select the same single time slot for sensing, then an event will be detected if it occurs within the sensing field of at least one sensor whilst the sensors are actively sensing, or if the event occurs whilst the sensors are sleeping, but is still detectable when they next start actively sensing again. Given the Poisson process describing the time during which an event remains detectable after its initial occurrence, the probability of an event being detectable after time t is given by $\int_t^\infty \lambda_d \mathrm{e}^{-\lambda_d \tau} d\tau = \mathrm{e}^{-\lambda_d t}$. Thus, if we consider that an event occurs within any specific time slot, and define n as the number of time slots until the sensors are again in their sensing state (where $n = 0$ indicates that one of the sensors is currently in its sense state), then the probability of detecting the event is 1 when $n = 0$, and is given by $\int_0^{1/L} \mathrm{e}^{-\lambda_d (n/L - t)} dt = \frac{\mathrm{e}^{-\lambda_d n/L}}{\lambda_d} \left(\mathrm{e}^{\lambda_d/L} - 1\right)$ when $n \geq 1$.

Using this result, the fact that events are equally likely to occur within all L time slots, and the result for the probability that the event occurs within the sensing range of at least one sensor derived in the previous section, allows us to express the overall probability of event detection as:

$$PED_{\text{synchronised}} = \left(1 - \mathrm{e}^{-\lambda_s \pi r^2}\right) \sum_{n=0}^{L-1} \begin{cases} 1/L & n = 0 \\ \frac{\mathrm{e}^{-\lambda_d n/L}}{\lambda_d} \left(\mathrm{e}^{\lambda_d/L} - 1\right) & n \geq 1 \end{cases} \tag{7}$$

Figure 7a shows an illustration of this case when $L = 4$.

3.4.3 Randomly coordinated sensors

In order to calculate the probability of event detection when sensors are energy constrained but each uses a sensing schedule in which one time slot is independently randomly selected for sensing, we note that the effective sensing schedule of an area that falls within the sensing ranges of a number of sensors is described by the logical 'OR' of the schedules of each individual sensor. For example, Fig. 7b shows the case where an area is overlapped by three sensors, $\{S_1, S_2, S_3\}$, with individual sensing schedules, $\mathbf{s}_1 = \{1, 0, 0, 0\}$, $\mathbf{s}_2 = \{0, 0, 0, 1\}$,

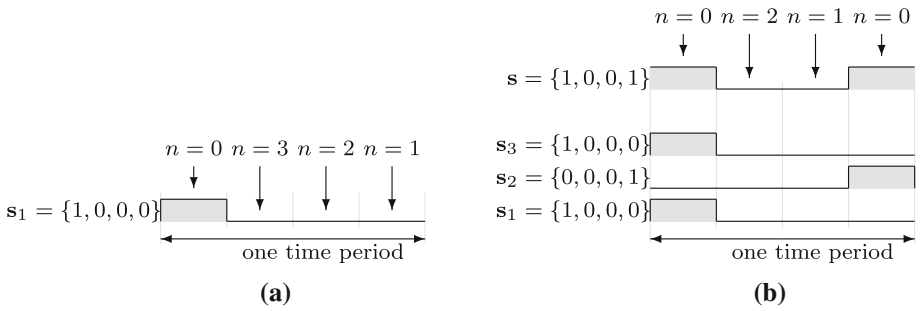

Fig. 7 Example showing the effective schedule for an area that falls within the sensing radius of **a** a single sensor with sensing schedule $\mathbf{s}_1 = \{1, 0, 0, 0\}$, and **b** three sensors with sensing schedules $\mathbf{s}_1 = \{1, 0, 0, 0\}$, $\mathbf{s}_2 = \{0, 0, 0, 1\}$ and $\mathbf{s}_3 = \{1, 0, 0, 0\}$

and $\mathbf{s}_3 = \{1, 0, 0, 0\}$, giving rise to the effective schedule of $\mathbf{s} = \{1, 0, 0, 1\}$. As above, given any such schedule we can calculate the probability of detecting an event within this area, by simply summing over each time slot and considering the number of time slots until the sensors are again in their sensing state (see Fig. 7b again).

Algorithm 4 presents a general method to calculate the probability that an event is detected if it occurs within an area whose sensing schedule is described by the vector $\mathbf{s} = \{s_0, \ldots, s_{L-1}\}$. Note that as λ_d increases (such that the events become increasingly transient), then the probability of detection decreases toward only detecting the event during the cycle in which the sensor is in its sense state (i.e. $1/L$). Conversely, as λ_d decreases toward zero (such that the events become increasingly long lived), then the probability of detecting the event approaches one.

Algorithm 4 $P(\text{detection}|\lambda_d, \mathbf{s})$

```
value ← 0
for i = 0 to L − 1 do
    n ← 0; j ← i
    while s_j = 0 do
        j ← mod(j + 1, L); n ← n + 1
    end while
    if n = 0 then
        value ← value + 1/L
    else
        value ← value + e^{−λ_d n/L} (e^{λ_d/L} − 1) /λ_d
    end if
end for
return value
```

We can then use this result to calculate the probability of detecting an event assuming that each sensor individually selects one of the L time slots in which to sense. We do so by summing over the probabilities that any point in the environment is within the sensing fields of m sensors, and that the sensing schedules of these m sensors combine to give any of the 2^L possible sensing schedules (denoted by $\mathcal{S}$). In the latter case, the probability of any sensing schedules, $\mathbf{s}$, arising from the combination of m individual schedules, each of length L with a single active sensing time slot, is given by $\sum_{k=0}^{n}(-1)^k \binom{n}{k}(n-k)^m/L^m$, where n is the number of sensing time slots in the combined schedule. Note that the numerator in this

expression is a standard result in probability theory regarding the number of ways in which m balls may be placed into L cups such that exactly n of them are not empty (see for example [42]), and the denominator is the total number of ways in which m balls may be placed in L cups. Algorithm 5 shows this calculation in pseudo-code.

Algorithm 5 $PED_{\text{random}}(\lambda_s, \lambda_d, r, L)$

$value \leftarrow 0$
for $m = 1$ to ∞ **do**
  $P(m) = (\lambda_s \pi r^2)^m \mathrm{e}^{-\lambda_s \pi r^2}/m!$
  for $\mathbf{s} \in \mathcal{S}$ **do**
    $n \leftarrow \sum_{k=0}^{L-1} s_k$
    if $n \leq m$ **then**
      $P(\mathbf{s}) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} (n-k)^m / L^m$
      $value \leftarrow value + P(m) \times P(\mathbf{s}) \times P(\text{detection}|\lambda_d, \mathbf{s})$
    end if
  end for
end for
return $value$

3.4.4 Optimally coordinated sensors

Finally, we can calculate an upper bound for the effectiveness of coordination between sensors. To do so, we assume that if any point in the environment is within the sensing fields of m sensors, then these sensors are able to perfectly coordinate their sensing schedules in order to maximise the probability that an event is detected at this point. This represents a strict upper bound on the probability that the network detects an event, since we ignore the real constraints on achieving this coordination for any given sensor network configuration.[8] Thus, if $m \geq L$ we assume that the area is continually sensed, and when $1 < m < L$ we assume that the sensor coordination gives rise to an optimal sensing schedule, $\mathbf{s}^*_{m,L}$. This optimal schedule can be automatically derived through exhaustive search using Algorithm 1, or more simply, by noting that the detection probability is maximised when the schedule contains m sensed time slots that are maximally separated. For example, if $L = 4$, then $\mathbf{s}^*_{1,4} = \{1, 0, 0, 0\}$, $\mathbf{s}^*_{2,4} = \{1, 0, 1, 0\}$ and $\mathbf{s}^*_{3,4} = \{1, 1, 1, 0\}$. Algorithm 6 shows this calculation.

Algorithm 6 $PED_{\text{optimal}}(\lambda_s, \lambda_d, r, L)$

$value \leftarrow 0$
for $m = 1$ to ∞ **do**
  $P(m) = (\lambda_s \pi r^2)^m \mathrm{e}^{-\lambda_s \pi r^2}/m!$
  if $m < L$ **then**
    $value \leftarrow value + P(m) \times P(\text{detection}|\lambda_d, \mathbf{s}^*_{m,L})$
  else
    $value \leftarrow value + P(m)$
  end if
end for
return $value$

[8] This is equivalent to the statement that zero clashes is a strict lower bound for solutions to a graph colouring problem, even though a specific problem instance may not be colourable.

3.5 Network performance comparison

Using the theoretical results presented in this section, we can calculate the maximum gain in system-wide performance that coordination may yield. These results are shown in Fig. 8 for cases where $L = 2$, 3 and 4, and in all cases, the departure rate of events is much greater than $1/L$ (i.e. events are very short lived). Short lived events represent the lower limit of performance of the network, because an event can only be detected if a sensor is in its active sensing state when the event occurs. As such, it represents the case where coordination can have the most significant impact.

In addition, we show results from a simulation of the sensor network described by our model in which a centralised optimisation routine (specifically simulated annealing) is used to calculate a sensor schedule against which to compare the theoretically calculated optimal. In contrast with the ideal upper bound, this algorithm provides feasible schedules, but cannot be used in practice to coordinate the sense/sleep schedules of the real sensors since it is centralised and assumes full knowledge of the topology of the network.

Notice that, while the theoretical analysis assumes that optimal coordination can be achieved for every point in the network, the theoretical optimal coordination case calculated in Sect. 3.4.4 is an upper bound for any operating conditions (as long as the distribution of sensors and the visibility time of the targets follow a Poisson distribution). Moreover, the centralised simulated annealing solution closely approximates this upper bound, indicating that this is a relatively tight bound, since it closely reflects what is possible in practice by running a simulated annealing algorithm on the network. Finally, this result suggests that the centralised simulated annealing solution is a useful benchmark for evaluating our decentralised max-sum algorithm against (as we do in Sect. 4). Clearly, as the density of the sensor deployment (λ_s) increases, then the probability of event detection increases, and in the limit, all events that occur within the environment are detected. Note that the optimally coordinated network always out performs the randomly coordinated network (as it must), and that as the density of the deployment increases, the gain increases. Indeed, in this example, for the case where $L = 4$, when $\lambda_s > 35$ the optimally coordinated network detects 50 % of the events that the randomly coordinated network fails to detect, or conversely, the optimally coordinated network is able to achieve the same level of performance as the randomly coordinated network with just 60 % of the sensors deployed. Summarising, these results indicate that coordination can yield a significant and worthwhile improvement in system-wide performance in this application.

4 Decentralised coordination for the wide area surveillance problem

The above analysis indicates that significant gains can be realised by coordinating the sense/sleep schedules of power-constrained sensors, and based on this, we now focus on decentralised coordination algorithms that can be deployed on the sensor nodes. Previous work in the area of wireless sensor networks has begun to address this challenge, for example, Hsin and Liu [17] consider coordinating the duty cycles of non-energy harvesting sensors in order to maintain a minimum probability of event detection while maximising the lifetime of the individual sensors. Giusti et al. [14] consider the problem of coordinating the wakeup time of energy neutral sensors, but do not explicitly consider the degree to which the sensing areas of neighbouring sensors overlap. Conversely, Kumar et al. [24] do explicitly deal with the expected overlap of neighbouring sensors in a setting where each point in the region must be covered by at least k sensors in order to correctly identify significant events

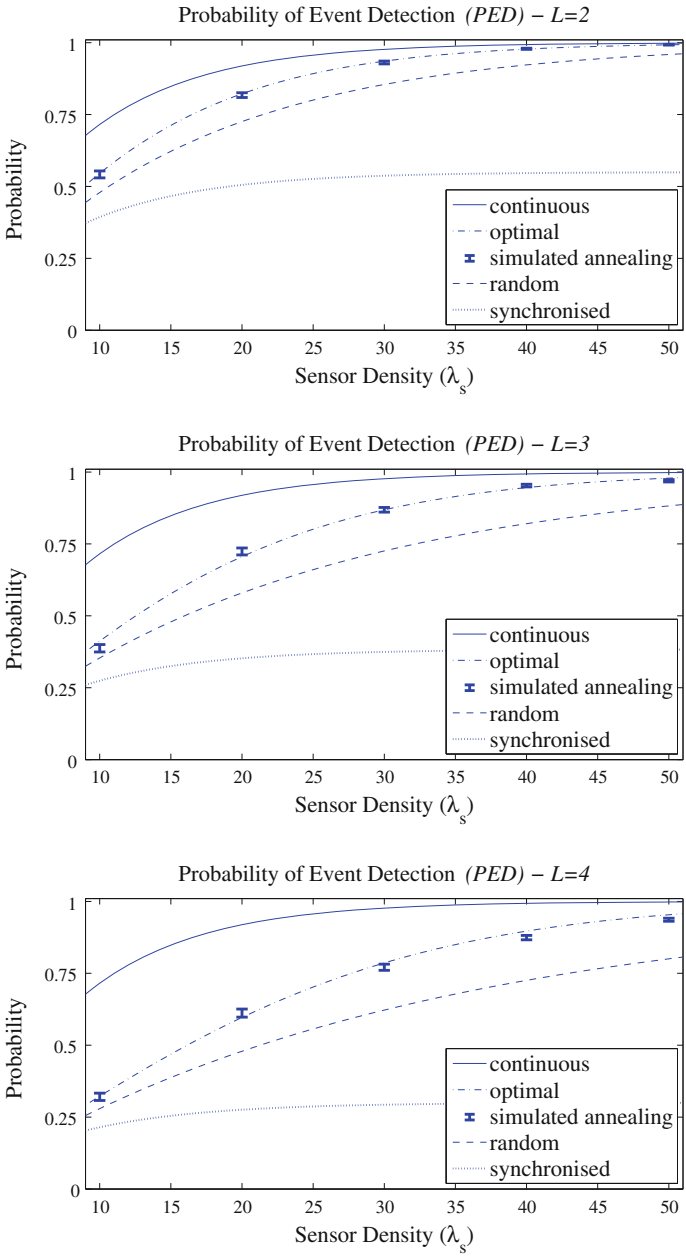

Fig. 8 Comparison of theoretical and simulation results for the probability of event detection for continuously powered, randomly coordinated and optimally coordinated sensors ($r = 0.2$ and $\lambda_d = 20$)

(k-coverage). However, rather than providing a coordination mechanism, they analyse a model of the problem, and provide guidance as to the number of sensors that should be deployed to achieve k-coverage and longevity of the network, in the absence of any coordi-

nation. Similarly, Ammari and Das [2] address the issue of k-coverage explicitly focusing on investigating the *conditional* connectivity of k-covered sensor networks, i.e. the minimal number of sensors whose removal disconnects the network into components each maintaining a specified property. In particular, they provide bounds for the conditional connectivity of a k-covered network given an isotropic model for the sensors.

However, much of this work assumes that the sensors have perfect a priori knowledge of their location, the location of their neighbours, and the degree of overlap of perfect circular sensing areas (see Sect. 3 for more details). In this section, we show how we can apply the max-sum algorithm to the wide area surveillance problem presented in Sect. 3, removing these restrictive assumptions, and thus developing an adaptive distributed coordination mechanism.

4.1 Applying the max-sum algorithm

To apply the max-sum coordination algorithm to the wide area surveillance problem it is necessary to first decompose the system-wide goal that we face (that of maximising the probability that an event is detected) into individual sensor utility functions. As shown in Sect. 3.3, the utility of each sensor is determined by its own sense/sleep schedule, and by those of sensors whose sensing fields overlap with its own. In the case that the sensors know the relative positions of these other sensors and the geometry of their sensing fields, and events are equally likely to occur anywhere within the area covered by the sensor network (strong assumptions common in the literature [17], and ones that we relax shortly), this utility function can easily be determined.

To this end, we define $\mathbf{N}_i$ to be a set of indexes indicating which other sensors' sensing fields overlap with that of sensor i and $\mathbf{k}$ is any subset of $\mathbf{N}_i$ (including the empty set). $A_{\{i\}\cup\mathbf{k}}$ is the area that is overlapped only by sensor i and those sensors in $\mathbf{k}$. For example, with respect to Fig. 9, the area $A_{\{1,2\}}$ is the area that is sensed only by sensors 1 and 2. In a slight abuse of notation, we represent the entire sensing area of sensor S_1 as S_1, and thus, note that the area $A_{\{1,2\}}$ is different from $S_1 \cap S_2$ because the area $S_1 \cap S_2$ would include also the sub area $S_1 \cap S_2 \cap S_3$. In general, we have:

$$A_{\{i\}\cup\mathbf{k}} = \bigcap_{j\in(\{i\}\cup\mathbf{k})} S_j \setminus \bigcup_{l\notin(\{i\}\cup\mathbf{k})} S_l$$

We define a function $G : 2^{\mathbf{X}} \rightarrow \mathcal{S}$ and $G(\mathbf{x}_{\{i\}\cup\mathbf{k}})$ is the combined sensing schedule of sensor i and those sensors in $\mathbf{k}$ (calculated through the logical 'OR' of each individual schedule, as

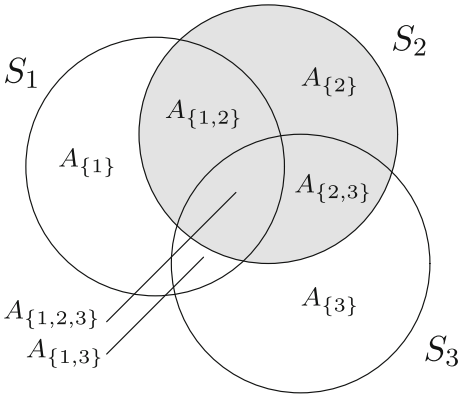

Fig. 9 Example showing the complete set of overlapping areas for three sensors S_1, S_2 and S_3

shown in Fig. 7). The utility of sensor i is then given by:

$$U_i(\mathbf{x}_i) = \sum_{\mathbf{k}\subseteq\mathbf{N}_i} \frac{A_{\{i\}\cup\mathbf{k}}}{|\{i\}\cup\mathbf{k}|} \times P(\text{detection}|\lambda_d, G(\mathbf{x}_{\{i\}\cup\mathbf{k}})) \tag{8}$$

where $P(\text{detection}|\lambda_d, G(\mathbf{x}_{\{i\}\cup\mathbf{k}}))$ is given by Algorithm 4. Note, that we scale the area by the number of sensors that can sense it to avoid double-counting areas which are represented by multiple sensors. This is important since we require that $\sum_i U_i(\mathbf{x}_i)$ is equal to the global probability of detecting an event. Also, note that when the set $\mathbf{k}$ is empty we consider the area covered only by the single sensor. For example, let us consider sensor S_2 shown in Fig. 9. To compute $U_2(\mathbf{x}_2)$ we need to consider all possible subsets of $N_2 = \{1, 3\}$. These subsets are: $\{\emptyset\}$, $\{1\}$, $\{3\}$, $\{1, 3\}$, therefore, $U_2(\mathbf{x}_2)$ is calculated considering the shaded areas $A_{\{2\}}$, $A_{\{1,2\}}$, $A_{\{2,3\}}$ and $A_{\{1,2,3\}}$.

4.2 Learning the mutual interaction of sensing fields

The utility function presented in Eq. 8 makes some strong assumptions regarding how each individual sensor calculates its utility. Specifically, it assumes that the sensors are able to determine the area of overlap of their own and their neighbouring sensors' sensing fields, and that they have no prior knowledge as to the distribution of events over these areas. In reality, sensors may have highly irregular and obscured sensing areas, they may not be able to determine the exact position of themselves, let alone neighbouring sensors, and events may be known to be more likely to occur in some areas than others. Thus in this section we relax these constraints, and describe how an additional *calibration phase* may be used to allow the individual sensors to learn these relationships through observing events in the environment prior to making a coordinated decision regarding their sense/sleep schedules.

To this end, rather than the theoretically tractable model of a wide area surveillance problem that we introduced in Sect. 3, we now consider a more realistic scenario based upon a simulation of an urban setting (based upon the RoboCup rescue simulation environment [20]). We again assume that energy harvesting sensors (with the same energy constraints and sense/sleep schedules as those previously considered) are randomly deployed within the environment, and these sensors are tasked with detecting vehicles that travel along the roads. We assume that the sensors have no a priori knowledge of the road network, and do not know their own location within it. We make no assumptions regarding the sensing fields of these sensors, although for ease of coding the simulation, we model these as circular fields with randomly assigned radii (which are unknown to the sensors). Figure 10 shows this simulation environment in operation. The area sensed by active sensors is shown in red, and moving vehicles are shown as white markers on the roads. A video of its operation is available online at https://vimeo.com/48231842

We then implement an additional calibration phase after deployment in which the sensors synchronise their sensing schedules and exchange information regarding the events that they have observed. In more detail, we implement the following scheme:

1. **Calibration Phase**:
   We assume that all sensors select the same sensing schedule, and thus, the sensors are all active and sense simultaneously. At regular intervals during this phase sensors exchange information regarding the events that they have detected, and they keep track of (i) the number of events that they observe individually, O_i , and (ii) the number of events that are both detected by themselves and a subset of their neighbours, $O_{\{i\}\cup\mathbf{k}}$. The exact form that this exchange of information takes depends on the nature of the sensors used, and

Fig. 10 Simulation of a wide area surveillance scenario (based on the RoboCup rescue simulation environment)

the events that they are detecting. Within our simulated wide area surveillance scenario, we assume either acoustic, seismic or visual sensors that are able to time stamp the appearance and disappearance of vehicles within their sensing fields. Comparison of the time stamps of observations of other sensors allows each sensor to identify whether vehicles are detected by multiple sensors as they cross its own sensing field.

For example, consider Fig. 11 in which two vehicles cross three overlapping sensing fields, and assume that sensor S_1 time stamps the appearance and disappearance of a vehicle at times 09:02:12 and 09:02:21 respectively, sensor S_2 time stamps the appearance and disappearance of a vehicle at times 09:02:15 and 09:02:24 respectively, and finally,

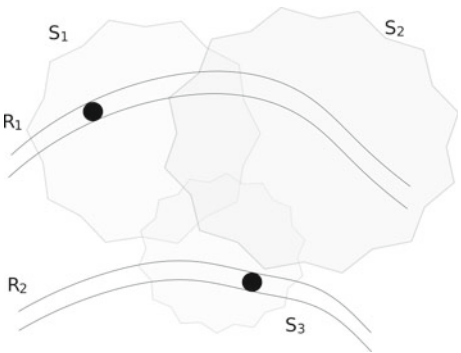

Fig. 11 Example showing the paths of two vehicles on roads, $\{R_1, R_2\}$, crossing the sensing fields of three overlapping sensors S_1, S_2 and S_3

sensor S_3 time stamps the appearance and disappearance of a vehicle at times 09:02:27 and 09:02:33 respectively. In this case, the intersection of the time stamps of sensors S_1 and S_2 lead these two sensors to conclude that $O_{\{1\}} = 1$, $O_{\{1,2\}} = 1$, $O_{\{2\}} = 1$, while the non-intersection of the time stamps of sensor S_3 leads it to conclude that $O_{\{3\}} = 1$. Note that in general, more complex techniques may be required to differentiate events when they occur concurrently. This will typically require some additional information such as the position of the event, or some recognisable characteristic of the event. Conversely, in other settings, such as tracking assets that are equipped with RFID tags, identification and detection automatically occur together. Within the data fusion and tracking literature, this problem is commonly known as data or track association [43]. Since data association is not the focus of this paper, in our simulated scenario we assume that events can be uniquely identified by their appearance and disappearance time.[9]
Moreover, notice that this calibration phase is a relatively long procedure where agents need to synchronize their sense/sleep schedules and exchange information for several communication cycles. Hence it should be run only when there is a high likelihood that the traffic load may change significantly, for example it could be run at fixed times of the day based on general information on traffic load dynamics (i.e., when there could be a transition from rush hour to a normal traffic situation and viceversa). In contrast, the calibration procedure should not be used in case of unexpected changes in the system configuration due for example to sensor malfunctioning or temporary communication breakdown, as the performance loss due to running this procedure would most likely be more significant than the possible gain due to having a more up to date system configuration.

2. **Coordination Phase**: The numbers of events observed in the calibration phase now acts as a proxy for the unknown areas of overlap between neighbouring sensors. Furthermore, it also captures the fact that events will not occur evenly over the entire area, but are restricted to certain areas (i.e. the roads in our case). Hence, the sensors now calculate their utility based on a modification of Eq. 8 given by:

$$U_i(\mathbf{x}_i) = \sum_{\mathbf{k} \subseteq \mathbf{N}_i} \frac{O_{\{i\} \cup \mathbf{k}}}{|\{i\} \cup \mathbf{k}|} \times P(\text{detection} | \lambda_d, G(\mathbf{x}_{\{i\} \cup \mathbf{k}})) \tag{9}$$

The sensors can now use the max-sum coordination algorithm presented earlier to coordinate their choice of sense/sleep schedule such that the utility of the overall sensor network is maximised, and hence, the probability of detection of a vehicle travelling within the area covered by the sensor network is maximised.

3. **Operational Phase**: Finally, the operational phase proceeds as before, sensors simply follow the sense/sleep schedule determined in the previous coordination phase. If during this phase a sensor fails the coordination algorithm above may simply be re-run to coordinate over the smaller sensor network. Likewise, should the position of sensors change, or new sensors be added, both the calibration phase and the coordination phase can be re-run to coordinate over the new environment in which the sensors find themselves. In Sect. 5 we shall describe our future work developing a more principled approach that allows for continuous self-adaption of the sensor network as the state of the environment, or the sensors themselves, changes over time.

[9] Hence, in the specific setting we consider here, sensors can compute the number of mutually observed events (i.e., $O_{\{i\} \cup \mathbf{k}}$) by sending at regular intervals a message to all neighbours that contains, for each detected vehicles, the time of appearance and the time of disappearance.

To validate this approach we now perform an empirical evaluation within our simulation environment comparing max-sum with various coordination mechanisms that we detail below.

4.3 Coordination mechanisms and coordination overhead

We compare results for four different coordination mechanisms:

- **Randomly Coordinated Sensors**: As in Sect. 3.4, the choice of each sensors' sense/sleep schedule is made randomly by each individual sensor with no coordination.
- **DSA Coordinated Sensors**: Using the results of the calibration phase, the sensors use the DSA algorithm (described below) to coordinate their sense/sleep schedules.
- **Max-sum Coordinated Sensors**: Using the results of the calibration phase, the sensors use the max-sum algorithm to coordinate their sense/sleep schedules.
- **Simulated Annealing Coordinated Sensors**: We use an offline centralised optimisation algorithm to benchmark the performance of DSA and max-sum. This is the same approach used in Sect. 3.5.[10]

As mentioned before, simulated annealing cannot be used in practice to coordinate the sense/sleep schedules of the real sensors because it is centralised and assumes full knowledge of the topology of the network. However, it is a useful benchmark for the performance of the decentralised coordination mechanisms we use here. In fact, a brute force approach for computing an optimal assignment would not scale to the instance size we are interested in, and while simulated annealing is not provably optimal, in our empirical evaluation it has very good performance being in the worst case less than 10 % away from the continuously powered network (which is an unreachable upper bound for any coordination approach). Moreover, in contrast to the optimal calculation presented in Sect. 3.4.4, simulated annealing does not make any assumption on sensors' distribution, event visibility time, and it does not assume that perfect coordination is always possible among sensors for every point in the environment. In this sense, simulated annealing represents what could actually be achieved by using a centralized optimization method in our experimental settings.

Algorithm 7 reports the pseudo-code description of the DSA algorithm we used here (which is similar to version DSA-C of [56]). In more detail, each agent executes the local optimization (line 5) only with probability p. When performing the optimization, the agent chooses a value for x_i that maximises the local utility U_i given the current values of neighbours $\mathbf{x_{-i}}$.[11] In our experiments, the termination condition is met when the number of executions of the while loop equals a predefined threshold, which is set to 300. Given our reference application where agents do not necessarily have a synchronized execution cycle for coordination, we decided to perform simulation following an asynchronous execution model where sensors execute the coordination algorithm independently of the others using the most up to date messages received by neighbours.

[10] Notice that, the empirical setting here is different from Sect. 3.5, but, as discussed below, Simulated Annealing still performs very close to the continuously powered network.

[11] As in version DSA-C of [56] we allow agents to change assignment whenever the utility does not degrade, hence agents are allowed to change the assignment also when the best alternative gives the same value of local utility. However, in [56] authors focus on a graph colouring problem and hence differentiate between situations where there is at least one conflict and the utility does not degrade (both DSA-B and DSA-C can change assignment) and situations where there is no conflict and the utility does not degrade (only DSA-C can change assignment). Since here we do not have hard constraints we do not consider conflicts but only the value of the utility, in this sense our version of DSA is similar to DSA-C.

In more details, following [12], we assume that each agent executes Algorithm 7 and communicates possible changes to its neighbours with a uniform time period (i.e., every τ milliseconds) and that the execution times of the agents are randomly distributed through one DSA optimization cycle. Within this setting, Fitzpatrick and Meertens [12] empirically show that the probability of execution p, which usually is a key parameter for DSA, does not have a significant impact on performance as long as the following condition holds: $1 - (1 - p)^L \leq P_T$, where L is communication latency and P_T is a constant for a given graph. Since here we assume instantaneous communication, in our setting the activation probability of DSA has only a minor impact (in the experiments we use $p = 0.6$). To evaluate the sensitivity of the execution model and activation probability on DSA performance in our specific setting, we compared results for DSA in the asynchronous execution model specified above and in the more standard synchronous execution model, where agents execute all at the same time and possible value changes are propagate in the next time step. In more detail, for each empirical configuration (number of sensors and number of time lots) considered in Fig. 16 we tested three activation probabilities $\{0.3, 0.6, 0.9\}$ for both the execution models averaging results over 100 runs. We then optimized DSA performance for each configuration and computed the difference in the percentage of missed vehicles between the synchronous and asynchronous model (where the percentage of missed vehicles is computed with respect to the continuous network model as specified in Sect. 4.4). Results show that the maximum difference between DSA performance in the two execution models was 0.8 % for DSA considering all neighbours and 0.5 % for DSA reducing the number of neighbours to 4 (see below for a discussion on the neighbour reduction process). Hence, we can conclude that while in general the level of exploration performed by DSA is dependent on the execution model in our specific setting this element appears to have a minor impact on the results.

As for communication overhead when executing the DSA algorithm, agents send in the worst case one message for each neighbour at each time step, and each message is one value indicating in which time step the agent will activate the sensor. Therefore the number of values communicated by each agent can be expressed as $O(k)$ where k is the number of neighbours. When executing the local optimization procedure, in the worst case, each agent must iterate through a number of values which is linear in the number of possible assignments for the variable x_i, i.e., $O(L)$ in our setting.

As for max-sum, each agent executes Algorithm 1. Also in this case we use an asynchronous execution model, and each agent terminates after 300 executions of the procedure. Following the analysis of max-sum coordination overhead in Sect. 2.7, the number of values communicated by each agent when executing the max-sum algorithm can be expressed as $O(Lk)$, while the computational complexity can be expressed as $O(L^{k+1})$. When the number of neighbours is high, as is the case for some of the network configurations that we consider in our empirical analysis, the computational effort associated with the message update procedure can become prohibitive, especially considering the memory and computation constraints of the low-power devices that are our target platform for deployment. To address this issue, we perform the max-sum coordination procedure with a reduced number of neighbours. In more detail, after we build the factor graph representation of our problem, each agent sorts its neighbours in decreasing order of the sum of events that they can both observe. For example, considering again the example in Fig. 9, agent a_1 will compute $V_2 = O_{1,2} + O_{1,2,3}$ for neighbour a_2 and $V_3 = O_{1,3} + O_{1,2,3}$ for neighbour a_3 and sort its neighbours accordingly. Next, each agent considers in the max-sum coordination procedure only its first r neighbours, where r is a predefined number. With this procedure we can control the computational complexity of the message update phase, that is now dominated by

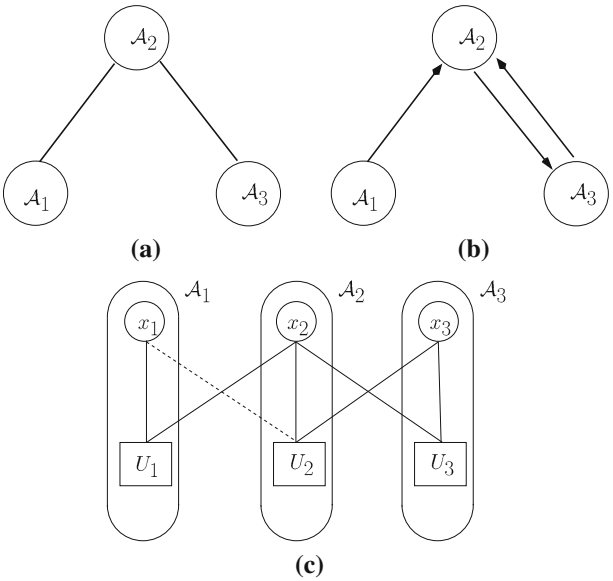

Fig. 12 A diagram showing **a** an interaction graph for three agents, **b** the neighbour relations after the neighbour reduction procedure (maximum number of neighbours set to 1), **c** the corresponding factor graph (the dashed edge represents the link removed from the neighbour reduction procedure)

$O(L^{r+1})$, as well as the communication overhead ($O(Lr)$). Notice that when we perform this neighbour reduction procedure, it might be the case that an agent a_1 considers another agent a_2 as its neighbour but a_2 does not consider a_1. To better understand this, consider the example in Fig. 12, where 12a shows the interaction graph and Fig. 12b shows neighbour relations after running the neighbour reduction procedure: a directed arrow from agent a_i to agent a_j indicates that a_i considers a_j as its neighbour in the coordination procedure. Figure 12c shows the resulting factor graph which is essentially a relaxation of the original factor graph with some links removed by the neighbour reduction procedure (dashed links in the picture). In the empirical evaluation, we apply the neighbour reduction procedure to DSA as well. In this case, each agent a_i shares information with all the agents that considers a_i to be their neighbour. However, when performing the local optimization step, a_i considers only the agents that it considers to be neighbours and ignores the others. For example, in the situation of Fig. 12 agent a_2 sends messages for value update to a_1 and a_3, but does not consider a_1 in its local optimization procedure. The benefit for DSA in using a reduced number of neighbours is to decrease communication. For example in Fig. 12, agent a_1 does not need to send any message because no one considers it to be a neighbour. While the communication overhead of DSA is usually moderate, the interaction graphs that we deal with in our empirical evaluation can be very dense (i.e., for 120 sensors we have an average density of 12.7). Moreover, in some configurations we can have agents that have a very high number of neighbours (i.e., up to 64 in our experiments). Since radio communication is usually a very energy consuming task for low-power devices, reducing the communication load is in our setting highly desirable.

Having described the coordination mechanisms that we consider, we now provide results for our empirical evaluation.

Algorithm 7 $DSA(p)$

$x_i \leftarrow$ Random Assignment
while termination condition is not met **do**
  $r \leftarrow$ Random number
  if $r < p$ **then**
    $V_i = U_i(x_i; \mathbf{x}_{-i})$
    $V_i^* = \max_{x_i} U_i(x_i; \mathbf{x}_{-i})$
    $x_i^* = \arg\max_{x_i} U_i(x_i; \mathbf{x}_{-i})$
    if $V_i \leq V_i^*$ **then**
      $x_i \leftarrow x_i^*$
      SendValue(x_i)
    end if
  end if
  ReceiveValues()
end while

4.4 Empirical evaluation

We empirically evaluate our coordination mechanisms by simulating the above three phases with various random deployments of sensors whose sensing ranges are assumed to be circular discs with radius drawn uniformly between 0.05 and 0.15 dim (where dim is the maximum dimension of the area in which the sensors are deployed). During the calibration phase we simulated the movement of various vehicles between random start and end points, and the sensors exchanged observations with one another regarding their observations during this time. During the coordination phase, the sensors use the max-sum algorithm over a fixed number of cycles, in order to coordinate their sensing schedules. Finally, during the operational phase the sensors use the sensing schedules determined in the negotiation phase, and we simulate the movement of 1,000 vehicles between random start and end points. We measure the operational effectiveness of the sensor network by calculating the percentage of vehicles that are missed by the sensor network (i.e. vehicles that move between their start and end point without ever being within the sensing field of an actively sensing sensor) and for those vehicles that are detected, we measure the time taken for the first detection (i.e. the time at which the network first becomes aware of the presence of the vehicle after it leaves its start point). In computing these measures we consider only vehicles that can be detected by a continuously powered network, i.e., we do not consider as missed vehicles those that never crossed the sensing field of the network.

In more detail, we first evaluate the impact of the number of vehicle paths used in the calibration phase on the network performance. Specifically, we considered various numbers of sensors ($\{30, 60, 90\}$) and we fixed the sensing schedule length to three ($L = 3$). We then varied the number of vehicle paths used for calibrating and we plot the percentage of missed vehicles (measured in the coordination phase) against this number.

Figure 13 reports the average and standard error of the means of the percentage of missed vehicles over 100 repetitions. Results indicate that for all the coordination mechanisms, performance increases by increasing the number of training paths. Moreover the increase in performance seems to be similar for all the coordination mechanisms. However, the difference in performance is less significant for higher values of training paths and this difference becomes negligible after 500 paths. Based on this analysis, we used in all the subsequent experiments 1000 vehicle paths in the calibration phase.

Next, we evaluate the impact of the neighbour reduction procedure described in the previous section on the network performance. In particular, Fig. 14 plots the percentage of vehicles

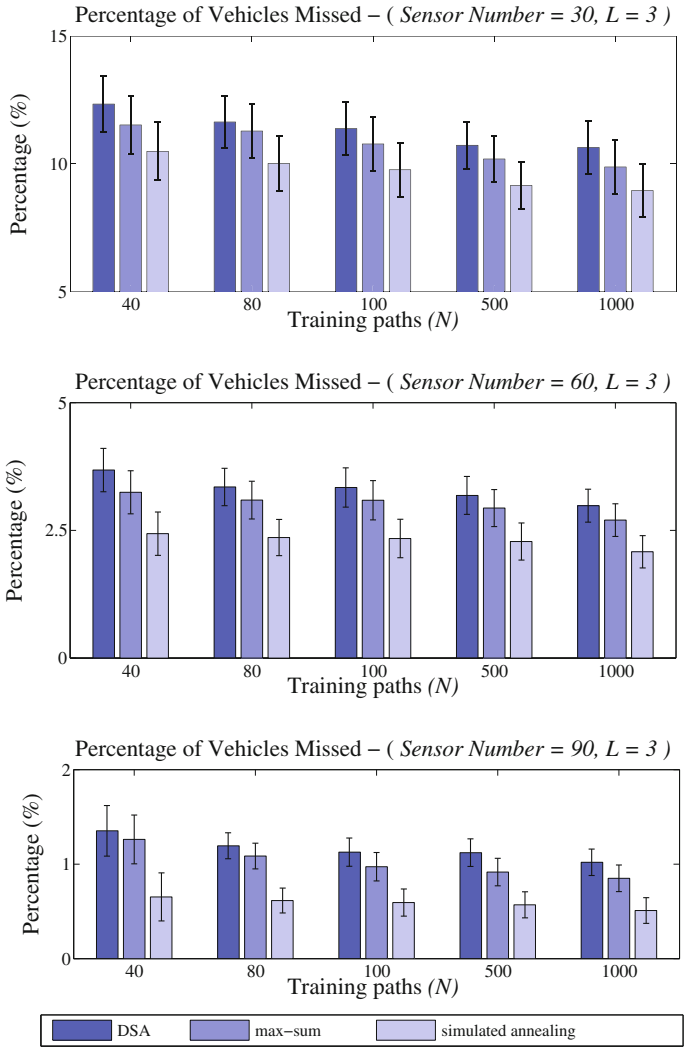

Fig. 13 Comparison of simulation results reporting the percentage of missed vehicles for a sensor network using DSA, max-sum, and centralised simulated annealing coordination algorithms plotted against the number of training paths used to calibrate the network

missed against the number of neighbours used in the coordination mechanisms. We compared the performance of DSA and simulated annealing with all neighbours, DSA and max-sum with a reduced number of neighbours (named DSA(r) and max-sum(r) respectively). Results show that, as expected, for both DSA(r) and max-sum(r) performance increases by increasing the maximum number of neighbours used in the coordination phase. However, the difference in performance between a maximum number of neighbours of 4 and 6 is rather small and DSA(4) and max-sum(4) reach comparable performance with the version of DSA and simulated annealing that consider all possible neighbours. This is an interesting result. Especially because, as mentioned before, the interaction graphs that correspond to our problem instance are very dense, (in this setting we have an average density of 4.34 and an average maximum

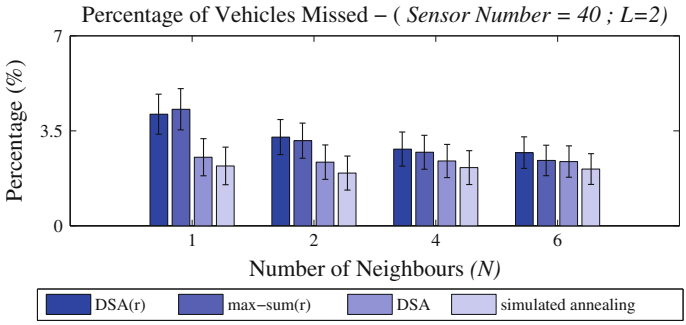

Fig. 14 Comparison of simulation results reporting the percentage of missed vehicles for a sensor network using DSA with r neighbours (DSA(r)), DSA using all neighbours (DSA), max-sum with r neighbours (max-sum(r)), a centralised simulated annealing coordination algorithms plotted against the number of neighbours used

degree of 27). Therefore, the neighbour reduction procedure is an effective way to substantially reduce the coordination overhead for max-sum and DSA, while maintaining good performance. Based on the above discussion, we use the neighbour reduction procedure in the following experiments with a maximum number of neighbours of 4.

As mentioned in Sect. 2.5, when running the max-sum algorithm one can observe different behaviours for convergence: (i) the messages converge to a fixed point (and consequently the joint assignment is stable), (ii) messages do not converge but the joint assignment converges, and (iii) messages do not converge and the joint assignment oscillates. Table 1 reports results for convergence of max-sum(4), DSA(4) and DSA, for a network with 40 sensors and $L = 2$. In more detail, we show the percentage of runs for which max-sum(4) converged (both for assignment and message) and the number of execution steps to reach convergence (these data are averaged over 100 runs). Results show that, as expected, DSA converges in all the runs and it converges in very few iterations. An interesting point is that the use of a reduced number of neighbours does not have a significant impact on convergence of DSA. The max-sum algorithm has a high rate of convergence, it takes longer to converge than DSA (particularly for message convergence), but the joint assignment typically stabilizes in about 20 iterations. In our experiments we consider two messages to be equal if their euclidean distance is smaller than a given threshold.[12] To give an indication of how the utility associated with the joint assignment evolves during the coordination phase, Fig. 15 reports the percentage of global utility achieved by the algorithms against the execution steps. We report the percentage of global utility (i.e., the sum of all $U_i(\mathbf{x}_i)$) achieved by the coordination mechanisms with respect to the global utility obtained with a continuously powered network. Notice that we use the global utility instead of the percentage of missed vehicles because here we are evaluating the joint assignment during the coordination phase, and thus before the sensors employ the negotiated joint assignment to actually detect vehicles. In this vein, Fig. 15a shows an exemplar trace of a run where max-sum(4) reached convergence (message and assignment) while Fig. 15b shows an exemplar trace of a run where max-sum(4) did not reach a stable assignment. In this latter case, max-sum(4) typically oscillates among assignments with values that lie in a relatively small interval.

Next, we evaluate the performance of the sensor network for three different length sensing schedules ($L = 2$, 3 and 4) and we investigate three different ranges of sensor number such

[12] We set the threshold to 10^{-3}.

Table 1 Comparison of percentage of converged runs and number of execution steps to reach convergence for a network with 40 sensors and $L = 2$ with different coordination algorithms: max-sum with four neighbours (max-sum(4)), DSA with four neighbours (DSA(4)), and DSA with all neighbours (DSA)

Coordination algorithm	Percentage of convergence	Average execution steps for convergence [standard error of the mean]
Max-sum(4)	Assignment = 97	Assignment = 19.9±[1.28]
	Messsage = 94	Message = 43.04±[3.90]
DSA(4)	100	4.9±[0.20]
DSA	100	4.95±[0.22]

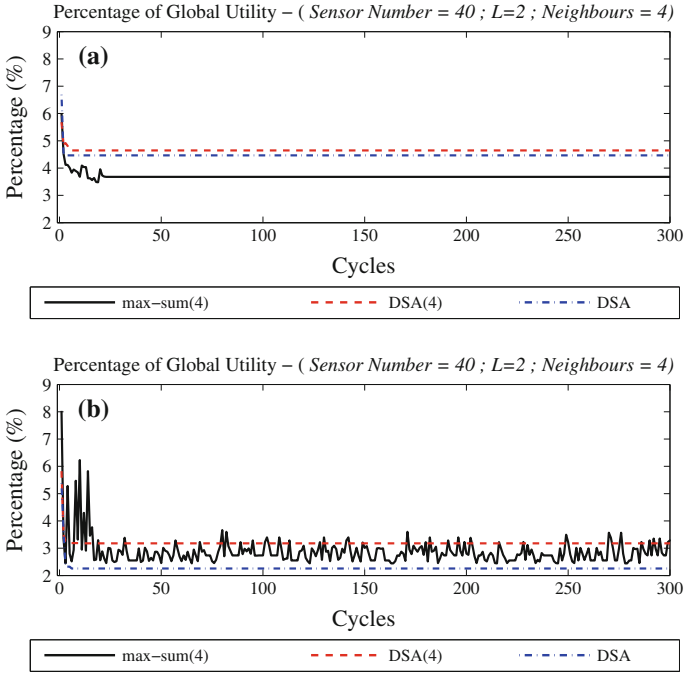

Fig. 15 Comparison of simulation results reporting the percentage of global utility for max-sum with four neighbours (max-sum(4)), DSA with four neighbours (DSA(4)), DSA using all neighbours (DSA), plotted against the simulation cycles. The two plot refer to one run where max-sum converged (joint assignment and messages) (**a**), and one run for which the max-sum joint assignment did not converge (**b**)

that the effective number of sensors (given by N/L) remained constant. In this way, in each deployment the total amount of energy that the network can use for sensing is the same. Notice however, that we keep the same size of the environment, therefore when more sensors are used there will be more overlap and, as results show, the network is able to detect more vehicles. Moreover, when there is more overlap among sensors, coordination has a higher impact on system performance.

The results of these experiments (averaged over 100 runs) are shown in Figs. 16 and 17, where the error bars represent the standard error of the mean in the repeated experiments. In more detail, Fig. 16 shows the percentage of vehicles that could be detected by a continuously powered network but fail to be detected by the coordinated sensor network; this

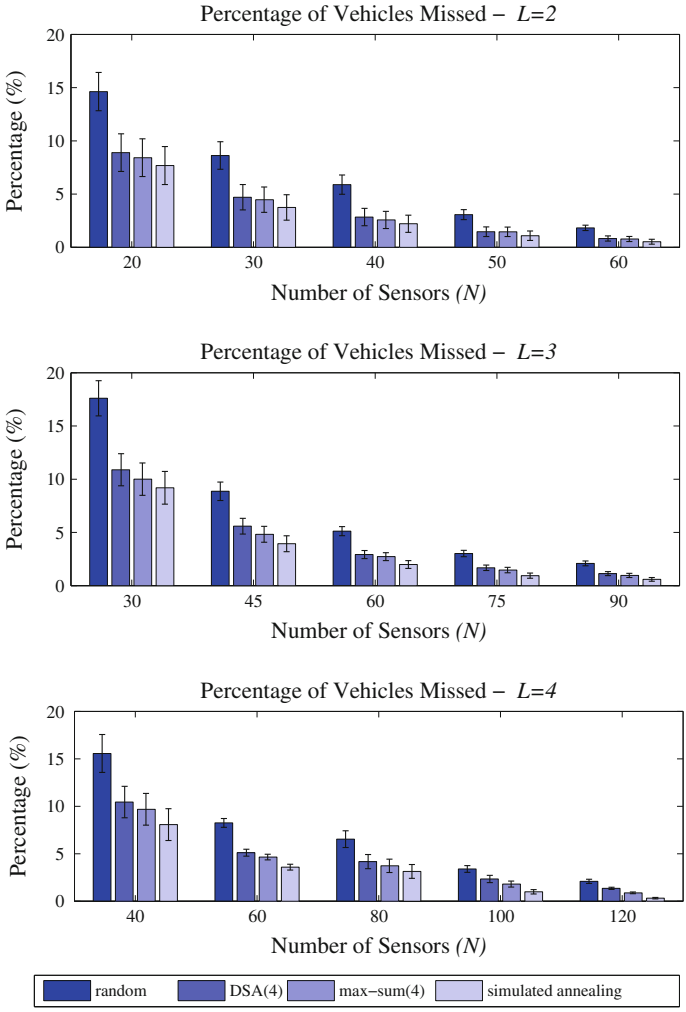

Fig. 16 Comparison of simulation results reporting the percentage of missed vehicles, for a sensor network using random, DSA(4), max-sum(4), and centralised simulated annealing coordination algorithms plotted against the number of deployed sensors

is our main metric for the performance of the network. Figure 17 shows the time that it took the coordinated sensor network to first detect each vehicle; a metric that we do not actively seek to minimise. Note that in all cases, the randomly coordinated sensor network performs the worst (failing to detect more vehicles and taking a longer time to detect them), and that the centralised simulated annealing approach provides the best solution. In more detail, averaging across all configurations, max-sum(4) achieves a 48 % improvement over the randomly coordinated network and simulated annealing shows a 25 % improvement over max-sum(4).[13]

[13] The performance improvement of a method X over a method Y is computed as (performance of X - performance of Y)/performance of X.

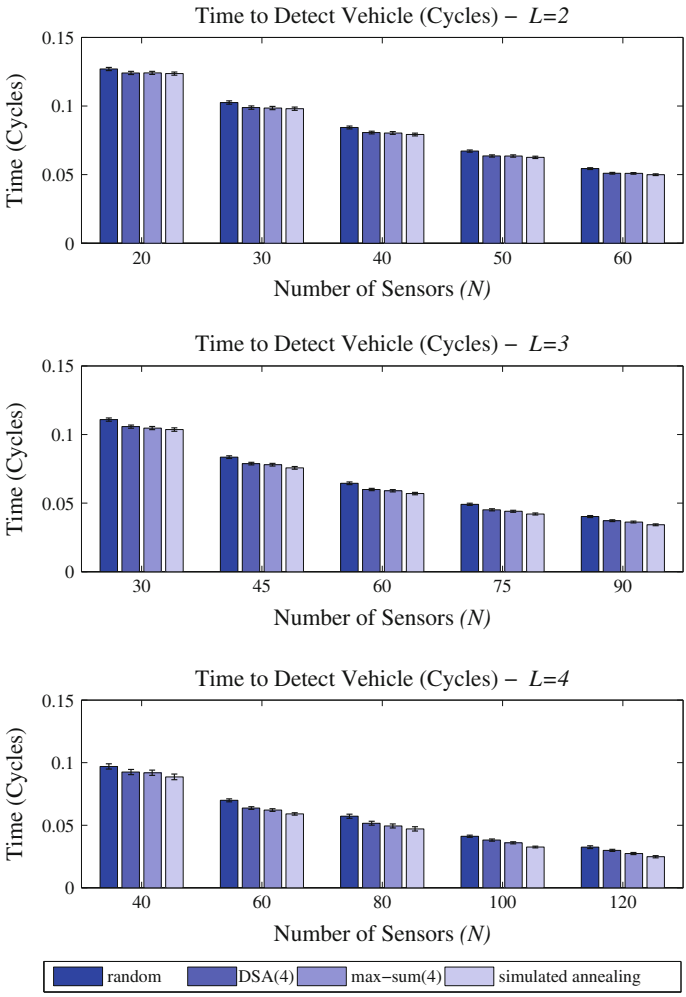

Fig. 17 Comparison of simulation results reporting the mean time to first detect a vehicle, for a sensor network using random, DSA(4), max-sum(4), and centralised simulated annealing coordination algorithms plotted against the number of deployed sensors

In most configurations, max-sum(4) and DSA(4) have comparable performance, with max-sum(4) usually being slightly superior (averaging across all configurations max-sum(4) shows a 10 % improvement over DSA(4)). The difference between the algorithms increases as both the number of sensors within the network and the length of sensing schedules increase. This trend is expected as the combinatorial coordination problem becomes harder as both these factors increase.

In more detail, Table 2 shows the results for both of these metrics for the specific case when $L = 4$ and $N = 120$. In this case, by using max-sum(4), we achieve a 57 % reduction in the number of missed vehicles (compared to the randomly coordinated network), and this performance is significantly better than DSA(4) (with a 35 % improvement of performance).

Table 2 Comparison of percentage of vehicles missed and time to detect vehicles for each coordination algorithm when $L = 4$ and $N = 120$

Coordination algorithm	Percentage of vehicles missed (%)	Time to vehicle (cycles) detect
Random	$2.0 \pm [0.4]$	$0.033 \pm [0.002]$
DSA(4)	$1.4 \pm [0.2]$	$0.030 \pm [0.002]$
Max-sum(4)	$0.9 \pm [0.2]$	$0.027 \pm [0.002]$
Simulated annealing	$0.3 \pm [0.2]$	$0.025 \pm [0.002]$

Notice that, this comparison is not considering the same level of communication overhead (in terms of number of bits exchanged over the network) but the same level of neighbours that the two algorithms consider in the optimization procedure. Hence, if we want to compare the two algorithms on the same level of communication one could increase the number of neighbours for DSA. However, a fair comparison in terms of communication load should also consider the device specific communication protocol (e.g., sending one message over the network with only one value, as it is the case in DSA, might actually be a waste of energy when packets have a fixed length) and since we prefer to avoid such low level details we report here the comparison based on the number of neighbours.

As for execution time, in the experiments we noticed that the execution time of the different algorithms is comparable. Therefore, we do not report this as a separate performance metric, as small differences in execution time are, in general, not good indicators of the computational requirements of the approach.[14] However, notice that, as stated in Sect. 2.7 message computation for max-sum using our utility-based factor graph representation shows an exponential elements in the number of neighbours, hence it does require more computation than DSA. The reader can have an estimation of the running time required to execute the experiments from the video at https://vimeo.com/48231842, which shows a live execution of the coordination algorithms.

Finally, we compare the performance of the max-sum and DSA algorithm in the presence of a lossy communication channel between the agents, a situation that is very likely to occur with low-power wireless devices. In more detail, Fig. 18 compares the percentage of missed vehicles for max-sum(4) and DSA(4) for a network with 40 sensors and $L = 2$ decreasing the probability of successful transmission of agent-to-agent messages. Results show that max-sum(4) performance remains almost constant, while DSA(4) performance significantly degrades when the probability of successful transmission decreases. The reason for this is that (as discussed in Sect. 2.7) when using max-sum each agent communicates utility information over each possible variable assignment every time it receives an updated message itself. In contrast, when using DSA, agents only communicate their preferred variable assignment and they only communicate this information when the assignment changes. Therefore, in this setting the minimal communication strategy of DSA can become disadvantageous.[15]

[14] Execution time is heavily dependent on many implementation specific details, which are not relevant to the core ideas of the technique. Simulated annealing is, in this respect, a notable exception, as it requires considerably more time than the other techniques. However, simulated annealing is used here only as a centralised upper bound on system performance.

[15] These results confirm the behaviour observed in [10] where max-sum and DSA were compared in the presence of a lossy communication channel on graph colouring benchmarks.

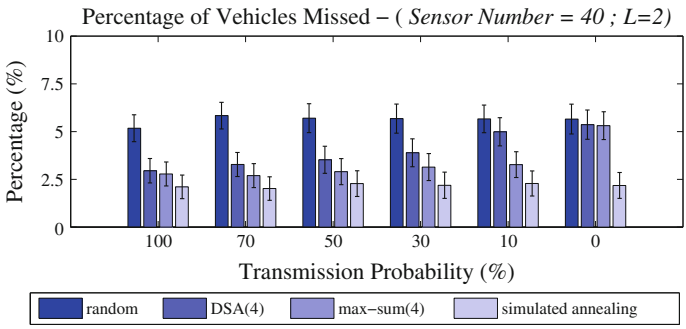

Fig. 18 Comparison of simulation results reporting the percentage of missed vehicles for a sensor network using DSA(4) and max-sum(4) in the presence of a lossy communication channel (random and simulated annealing are not affected by the message loss and are reported only for reference)

5 Conclusions

In this paper, we have presented a theoretical analysis of a wide area surveillance scenario and shown that coordination can yield significant gains in system-wide performance in this problem. We have discussed how agent-based decentralised coordination approaches, namely max-sum and DSA, can be applied in this setting and we have demonstrated how coordination can be achieved when sensors are deployed with no a priori information regarding their local environment. In such cases, agents must learn how their actions interact with those of neighbouring sensors, prior to using the coordination algorithm to coordinate their sense/sleep schedules in order to maximise the effectiveness of the sensor network as a whole. In a software simulation, we showed that this approach yields significant improvements in performance over the case of random coordination, and closely approaches that of a centralised optimisation algorithm (which has complete information regarding the network). The max-sum algorithm has comparable performance to DSA in most of the configurations we tested. However, it is significantly superior when the overlap among sensors is higher and the sensing schedule is longer. Moreover, max-sum performance is significantly less sensitive to the possibility of message losses than DSA. Nevertheless, DSA has a lower coordination overhead than max-sum, both in terms of communication and computation. Hence, when performance is comparable it is a valid coordination approach for some of our wide area surveillance settings.

In terms of future work, a first direction in this space is a full quantitative evaluation of the proposed approach with real sensors deployed for a specific application (e.g., for surveillance or monitoring). In particular, the validation of the max-sum algorithm on hardware presented in [49,8] shows that it can operate on limited hardware devices, and that it is able to adapt to unexpected changes of operating conditions (i.e. sensors that are added or removed from the environment or tasks that are inserted while the UAVs are executing their mission). However, a full quantitative evaluation of the approach in the wide area surveillance scenario considered here is important to properly assess its potential benefits for realistic applications. Specifically, such a deployment would be an ideal test-bed to properly compare the interaction-based and utility-based factor graph representations. In fact, it would allow us to consider important aspects that are hard to evaluate in simulation, such as the possibility for agents to update messages and perform the coordination phase without the need of any pre-processing phase

(e.g., any means of allocating functions to agents) and thus immediately react to any changes that could happen in the environment.

Second, we plan to address the trade-off between the performance of the max-sum algorithm and the coordination overhead. To this end, an interesting direction is to iteratively remove cycles from the factor graph, by applying known transformations, such as variable or function clustering [23], and estimate the increase in communication and computation overhead due to such transformations. In particular, diminishing the number of cycles in the factor graph has, in general, a positive effect on both convergence and solution quality. Thus a decentralised iterative approach that performs such transformations while estimating the introduced coordination overhead, would result in a flexible technique to address such a trade-off. This approach could be merged with Bounded Max-Sum, in order to obtain an approach that can quickly provide bounds on the optimal solution and refine both bounds and solution quality by iteratively performing the above mentioned transformations. An initial investigation towards this direction is presented in [44], where junction trees are built to perform optimal coordination trying to minimise communication and computation overhead of the agents, but a more comprehensive analysis is still lacking.

Third, we plan to extend the work by relaxing the requirement for a separate calibration phase prior to the negotiation phase. In this context, the synchronised schedules of the sensors during the calibration phase correspond to a period of poor system-wide performance that is offset by improved system-wide performance during the operational phase. However, it is also possible to learn about the occurrence of events, and hence the overlap of sensors' sensing fields, during this operational phase. Thus, we would like to investigate online algorithms that can explicitly trade-off between exploratory behaviour (synchronising with neighbouring sensors to learn about the occurrence of events), and exploitative behaviour (using relationships already learnt to coordinate the sensors). Recent advances on collaborative approaches to reinforcement learning that exploit problem structure seem a promising direction to realise such an online approach [22]. Moreover, we have already taken an initial step in this direction by proposing a Bayesian Reinforcement Learning approach in a cooperative multi-agent system that exploits problem structure by decomposing the overall coordination problem into regional sub-problems [48]. Applying this approach within this setting would remove the requirement for the three distinct phases. Rather, the sensors would continuously self-organise and self-adapt, changing sense/sleep schedules continuously to trade-off between exploration and exploitation. Such an approach would also naturally apply within dynamic settings where sensors' utilities may change at any time, sensors may fail, or additional sensors may be deployed. Moreover, the max-sum coordination algorithm that we derived in this paper already supports this continual behaviour since utility messages can be communicated, and sensors can estimate their optimal sensing schedule at anytime. Thus, it would appear to be a solid base on which to develop this more advanced behaviour.

Acknowledgment This work was funded by the ORCHID project (http://www.orchid.ac.uk/). Preliminary versions of some of the material presented in this article have previously appeared in the paper [40] and in the workshop paper [9]. In particular, in [9] we proposed the use of the max-sum approach to coordinate the sense/sleep cycles of energy constrained sensor networks for wide area surveillance, while in [40] we extend that contribution by removing the assumption that agents have *a priori* knowledge about their deployment. Here we significantly extend both the contributions by providing a more detailed discussion about the max-sum algorithm and new experiments. In particular we give a detailed description of the methodology to model coordination problem using different types of factor graphs and we include an example to clarify max-sum message computation. Moreover, we provide new experiments to evaluate the impact of the calibration phase on network performance, the trade-off between coordination overhead and performance, and finally the performance of coordination mechanisms to lossy communication.

References

1. Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *Information Theory IEEE Transactions*, *46*(2), 325–343.
2. Ammari, H. M., & Das, S. R. (2009). Fault tolerance measures for large-scale wireless sensor networks. *ACM Transactions on Autonomous and Adaptive System*, *4*(1), 1–28.
3. Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., et al. (2005). Sensor networks and distributed csp: Communication, computation and complexity. *Artificial Intelligence*, *161*(1–2), 117–147.
4. Bernstein, D. S., Zilberstein, S., Immerman, N. (2000) . The complexity of decentralized control of markov decision processes. In *Proceedings of UAI-2000*, pp. 32–37.
5. Bishop, C. M. (2006). *Pattern recognition and machine learning*. Berlin: Springer.
6. Dechter, R. (2003). *Constraint processing*. San Francisco: Morgan Kaufmann.
7. Delle Fave, F. M., Farinelli, A., Rogers, A., Jennings, N. R. (2012). A methodology for deploying the max-sum algorithm and a case study on unmanned aerial vehicles. In *IAAI 2012: The Twenty-Fourth Innovative Applications of Artificial Intelligence Conference*, pp. 2275–2280.
8. Delle Fave, F. M., Rogers, A., Xu, Z., Sukkarieh, S., Jennings, N. R. (2012). Deploying the max-sum algorithm for coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 469–476.
9. Farinelli, A., Rogers, A., Jennings, N. R. (2008). Maximising sensor network efficiency through agent-based coordination of sense/sleep schedules. In *Proceedings of the Workshop on Energy in Wireless Sensor Networks in conjuction with DCOSS 2008*.
10. Farinelli, A., Rogers, A., Petcu, A., Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pp. 639–646.
11. Fernández, R., Béjar, R., Krishnamachari, B., Gomes, C., & Selman, B. (2003). *Distributed sensor networks a multiagent perspective, chapter communication and computation in distributed CSP algorithms* (pp. 299–319). Dordrecht: Kluwer Academic.
12. Fitzpatrick, S., & Meertens, L. (2003). *Distributed sensor networks a multiagent perspective, chapter distributed coordination through anarchic optimization* (pp. 257–293). Dordrecht: Kluwer Academic.
13. Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(5814), 972.
14. Giusti, A., Murphy, A. L., & Picco, G. P. (2007). Decentralised scattering of wake-up times in wireless sensor networks. In *Proceedings of the Fourth European Conference on Wireless Sensor Networks*, pp. 245–260.
15. Guestrin, C., Koller, D., Parr, R. (2001). Multiagent planning with factored mdps. In *Advances in neural information processing systems (NIPS)*, pp. 1523–1530, Vancouver.
16. Guestrin, C., Lagoudakis, M., Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of ICML-02*, pp. 227–234.
17. Hsin, C., Liu, M. (2004). Network coverage using low duty-cycled sensors: Random & coordinated sleep algorithm. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, pp. 433–442.
18. Kansal, A., Hsu, J., Zahedi, S., & Srivastava, M. B. (2007). Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems*, *6*(4), 54–61.
19. Kho, J., Rogers, A., & Jennings, N. R. (2009). Decentralised control of adaptive sampling in wireless sensor networks. *ACM Transactions on Sensor Networks*, *5*(3), 19–35.
20. Kitano, H. (2000). Robocup rescue: A grand challenge for multi-agent systems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS)*, pp. 5–12.
21. Kok, J. R., Vlassis, N. (2005). Using the max-plus algorithm for multiagent decision making in coordination graphs. In *RoboCup-2005: Robot Soccer World Cup IX*, Osaka.
22. Kok, J. R., & Vlassis, N. (December 2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, *7*, 1789–1828.
23. Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *42*(2), 498–519.
24. Kumar, S., Lai, H. T., Balogh, J. (2004). On k-coverage in a mostly sleeping sensor network. In *Proceedings of the Tenth Annual International Conference on Mobile Computing and Networking (MobiCom 2004)*, pp. 144–158.
25. Lesser, V., Ortiz, C. L., & Tambe, M. (2003). *Distributed sensor networks a multiagent perspective*. Dordrecht: Kluwer Academic.

26. MacKay, D. J. C. (1999). Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, *45*(2), 399–431.
27. MacKay, D. J. C. (2003). *Information theory, inference, and learning algorithms*. New York: Cambridge University Press.
28. Maheswaran, R. T., Pearce, J. P., Tambe, M. (2004). Distributed algorithms for dcop: A graphical-game-based approach. *In the 17th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pp. 432–439.
29. Mailler, R., Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, pp. 438–445.
30. Makarenko, A., Durrant-Whyte, H.F. (2004). Decentralized data fusion and control algorithms in active sensor networks. In *Proceedings of Seventh International Conference on Information Fusion (Fusion 2004)*, pp. 479–486.
31. Mezard, M., Parisi, G., & Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. *Science*, *297*(5582), 812–815.
32. Modi, P. J., Scerri, P., Shen, W. M., & Tambe, M. (2003). *Distributed sensor networks a multiagent perspective, chapter distributed resource allocation* (pp. 219–256). Dordrecht: Kluwer Academic.
33. Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, *161*, 149–180.
34. Murphy, K. P., Weiss, Y., Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in, Artificial Intelligence (UAI'99)*, pp. 467–475.
35. Oliehoek, Frans A. (2010). Value-based planning for teams of agents in stochastic partially observable environments. PhD thesis, Informatics Institute, University of Amsterdam.
36. Petcu, A., Faltings, B. (2005). DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, (IJCAI 2005)*, pp. 266–271.
37. Petcu, A., Faltings, B. (2005). S-dpop: Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, pp. 449–454, Pittsburgh, AAAI.
38. Ramchurn, S., Farinelli, A., Macarthur, K., Polukarov, M., & Jennings, N. R. (2010). Decentralised coordination in robocup rescue. *The Computer Journal*, *53*(9), 1–15.
39. Rogers, A., David, E., & Jennings, N. R. (2005). Self-organized routing for wireless microsensor networks. *Systems Man and Cybernetics Part A IEEE Transactions*, *35*(3), 349–359.
40. Rogers, A., Farinelli, A. and Jennings, N. R. (2010). Self-organising sensors for wide area surveillance using the max-sum algorithm.
41. Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (February 2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, *175*(2), 730–759.
42. Rozanov, Y. A. (1977). *Probability theory: A concise course*. Dover Publications: Dover Books on Mathematics Series.
43. Bar Shalom, Y., & Fortmann, T. E. (1988). *Tracking and data association*. Boston: Academic-Press.
44. Stefanovitch, N., Farinelli, A., Rogers, A., Jennings, N. R. (2011). Resource-aware junction trees for efficient multi-agent coordination. In *Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pp. 363–370, Taipei.
45. Stranders, R., Farinelli, A., Rogers, A., Jennings, N. R. (2009). Decentralised control of continuously valued control parameters using the max-sum algorithm. In *8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 601–608.
46. Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, pp. 299–304.
47. Sultanik, E. A., Lass, R. N., Regli, W. C. (2009). Dynamic configuration of agent organizations. In *Proceedings of the 21st international jont conference on Artifical, intelligence, IJCAI'09*, pp. 305–311.
48. Teacy,W. T. L., Chalkiadakis, G., Farinelli, A., Rogers, A., Jennings, N. R., McClean, S., Parr, G. (2012). Decentralized bayesian reinforcement learning for online agent collaboration. In *11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 417–424.
49. Teacy W. T. L., Farinelli, A., Grabham, N. J., Padhy, P., Rogers, A., Jennings, N. R. (2008). Max-sum decentralised coordination for sensor systems. In *7th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1697–1698.

50. Velagapudi, P., Varakantham, P., Sycara, K., Scerri, P. (2011). Distributed model shaping for scaling to decentralized pomdps with hundreds of agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, Vol. 3, AAMAS '11, pp. 955–962.
51. Vinyals, M., Cerquides, J., Farinelli, A., Rodrguez-Aguilar, J. A. (2010). Worst-case bounds on the quality of max-product fixed-points. In *In Neural Information Processing Systems (NIPS)*, pp. 2325–2333. Vancouver: MIT Press.
52. Vinyals, M., Rodriguez-Aguilar, J., & Cerquides, J. (2011). Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, *22*, 439–464.
53. Weddell, A. S., Harris, N. R., White, N. M. (2008). Alternative Energy Sources for Sensor Nodes: Rationalized Design for Long-Term Deployment. In *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference* (I^2MTC 2008). (in press).
54. Weiss, Y., & Freeman, W. T. (2001). On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, *47*(2), 723–735.
55. Zhang, P., Sadler, C., Lyon, S., Martonosi, M. (2004). Hardware design experiences in zebranet. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*.
56. Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (January 2005). Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1–2), 55–87.