UNIVERSITY OF SOUTHAMPTON

Incremental Search Algorithms for On-Line Planning.

by

Jason D. R. Farquhar

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

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

August 2004

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Jason D. R. Farquhar

An *on-line* planning problem is one where an agent must optimise some objective criterion by making a sequence of action selection decisions, where the time and resources used in making decisions count in assessing overall solution quality. Typically in these problems it is not possible to find an optimal complete solution before an initial action must be executed, instead to maximise its performance the agent must interleave decision making and execution. This thesis investigates using decision theoretic techniques to solve these problems by equipping the agent with the ability to reason about the "complexity induced" uncertainties in its information and the costs of computation. The basic thinking/execution interleaving is provided by incremental search, where decisions are made incrementally based upon a guided partial search through the local space of possible solutions.

The major sub-problems such an agent must solve are; i) decision making—how to make decisions whilst in a state of "complexity induced uncertainty", ii) search control—which node to expand next, iii) stopping—when to stop searching. Decision making is treated as a value estimation problem. By representing the agent's uncertainty in probabilistic terms this can be solved using decision theoretic techniques. Existing decision making systems are analysed and new low computational cost approximate decision theoretic algorithms developed. These are shown to give significant improvements in decision quality. Search control is also treated as an estimation problem, in this case estimating the expected value of computation (EVC), which is the expected benefit of a computation in reducing the agent's uncertainty (hence improving action selection). New, low computational cost, approximations for the EVC are also developed and shown to give significant improvements in decision quality. Sophisticated stopping can be achieved by trading-off the EVC of further search against its computational cost. Experimental results show that integration of the sub-problem solutions is critical to agent performance. The results also show that (assuming no adverse interactions) improving decision making gives the greatest improvements, with search control and stopping offering more modest benefits.

Contents

List of Figures

Nomenclature

$\Pr(x)$	probability of event x
$\Pr_X(c)$	probability of random variable X satisfying condition c
$\mathrm{p}_X(x)$	probability density function (PDF) for real random variable X
$\mathrm{P}_X(x), \mathrm{P}_X^<(x)$	cumulative density function (CDF) for real random variable X, i.e. $\int_{-\infty}^{x} \mathrm{p}_X(z)dz$
$\mathrm{P}_X^>(x)$	Complementary CDF for real random variable X, i.e. $1 - \mathrm{P}_X(x) = \int_{x}^{\infty} \mathrm{p}_X(z)dz$
$\mathrm{E}_X\{f(\cdot)\}$	expectation over random variable X for function $f(\cdot)$
$\langle X \vert Y \rangle$	the average of random variable X, given condition Y
$x \in \mathcal{X}$	a state from the state space $\mathcal{X}$
$y \in \mathcal{Y}$	an observation from the observation space $\mathcal{Y}$
$u \in \mathcal{U}$	an action from the set of actions $\mathcal{U}$
$\mathbf{h} \in \mathcal{H}$	an abstract feature vector from the feature space $\mathcal{H}$
$x{:}y$	the range of integer values from x to y
$z_{t_0:t_1}, z(t_0{:}t_1)$	the sequence of values of variable z from time t_0 to t_1
f_x	the state transition function
f_y	the state observation function
O	an agent observation pair, (u, y)
M	an agent architecture
$b \in \mathcal{B}$	an agent belief state from the belief space $\mathcal{B}$
$c \in \mathcal{L}$	a computation drawn from the language $\mathcal{L}$
c_α	the special computation corresponding to selection and execution of the Bayes action α
α	the current Bayes action, $\alpha = \max_u V(x, u)$
β	the current second best action
π	a policy
π^*	an optimal policy
$\pi_g(\mathbf{V})$	a greedy policy w.r.t. the value function $\mathbf{V}$
$[\pi_1, \pi_2]$	the composite policy formed by acting according to policy π_1 initially and then according to policy π_2
V_π	the value function for the policy π
V^*	the value function of an optimal policy

$\hat{V}$	an estimated value
V^i	an *intrinsic* value
R	an objective function
r	an immediate reward function
C	a computational cost function
E	an environment
T	a trajectory function
γ	the labelled local search space graph
$\mathcal{N}_{\mu,\sigma}$	the normal distribution with mean μ and standard deviation σ
Φ	the CDF of the standard normal distribution
$\mathcal{H}$	the Heaviside unit step function
δ	the Dirac delta function (δ_y the delta function with origin at y)

Acknowledgements

During the period of my Ph.D. I have had cause to be grateful for the advice, support and understanding of many people. In particular I would like to thank the following:

- Anna for putting up with my occasional grumpiness and variable working hours and for her continuous encouragement through the seemingly endless slog of the write-up.
- My parents and siblings for their encouragement and support.
- The Southampton posse, both the Farnborough survivors – James and Ant – and the "old skool" – Phil and Mark.
- To Owen, for his insightful comments and meticulous proofreading of this thesis.
- To the many friends I have made in ISIS who have made my time in ISIS both interesting and enjoyable.
- And last but by no means least my two supervisors, Chris Harris and Adam Prügel-Bennett, for giving me the freedom to pursue my own ideas and the encouragement to keep going when things go wrong.

Chapter 1

Introduction and overview

1.1 The on-line planning problem

My goal is the construction of autonomous intelligent agents to solve real-world problems, such as control of autonomous vehicles or chemical plants.

The fundamental problem such agents face is to take a sequence of observations of a particular environment and generate a sequence of actions in real-time in such a way as to optimise some objective criterion which depends on the agent's "behaviour", i.e. the sequence of environmental states and agent actions which arise from the agent-environment interaction. Further, the agent must cope with a range of possible environments and objective criteria, which may be initially unknown or change dynamically. Figure 1.1 illustrates the on-line planning problem.

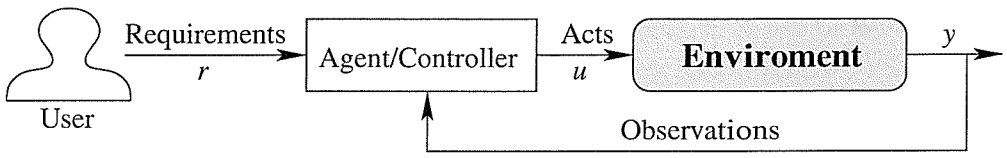


FIGURE 1.1: The On-Line Planning Problem is one of mapping observations to actions in such a way as to best fulfil the users requirements.

As an example of an on-line planning problem consider an agent which transports widgets in a factory. The set of possible environments are agent's possible locations within the factory. The possible objectives are the set of possible goal locations which the agent could be asked to reach in minimal time. Alternatively, consider an agent which controls a chemical factory. Here, the set of environments are the possible factory configurations. The range of desired chemical production rates give the set of possible objective criteria.

Given an on-line planning problem, the problem we face as agent designers is to; "design an agent which solves the on-line planning problem as well as possible given its limitations, both

physical (in terms of the actions it can execute) and computational (in terms of the decision procedures it can implement)."

This is an operational definition which imposes no requirements on the agent's internal operation, only on the externally observable effects of this operation matter, i.e. what counts is what the agent does, not what it thinks or even whether it thinks at all. This allows great flexibility in agent design—so long as they perform well we don't care if they perform elaborate reasoning to construct plans, or just react without thinking. Thus, we view internal operations of the agent, such as planning or learning, as occurring *in the service* of finding the right action.

This is important as typically in on-line planning problems the dynamics of the environment mean the cost of decision making depends on the current context. For example, in a chemical plant the cost of thinking time increases markedly if a vessel is about to explode! Thus, complex decision making procedures must "pay for themselves" by improving action selection enough to offset their cost. The design of the agent must be *bounded rational* (Russell and Subramaniam 1995; Parkes 1996) and take account of both the theoretical "optimality" of its decisions and the resources used in reaching them. Further, since it is not typically viable to find an optimal complete action sequence before an initial action must be executed, the agent's decision making must be;

- *reactive*, where it can revise its plans to respond to unforeseen events, and
- *incremental*, where it commits to the most urgent decision first.

Three main approaches have been used in the literature to solve on-line planning problems;

- the *programming* approach, where the system designer hand-codes a controller off-line,
- the *planning* approach, where the agent automatically derives the desired controller on-line based upon some suitable (partial) world model, and
- the *learning* approach, where the agent adaptively modifies its controller based upon its experiences.

Examples of the programming approach include; the reactive systems used in mobile robotics (Brooks 1982; Kaelbling and Rosenschein 1990; Agre and Chapman 1990), Schoppers (1987) universal planning system, and the controllers developed using traditional feedback control theory (Jacobs 1974; Barnett and Cameron 1985). AI Planning systems (Drummond and Tate 1989; Allen, Hendler, and Tate 1990) and Receding Horizon/Model Predictive Controllers (Mayne, Rawlings, Rao, and Scokaert 2000; Garcia, Prett, and Morari 1989) are examples of the planning approach. Examples of the third approach include systems based upon Reinforcement learning (Sutton and Barto 1998) or adaptive control (Astrom and Wittenmark 1994).

The three approaches have different strengths and weaknesses which make them variously beneficial in different application domains. For example it is hard to beat the simplicity

(1) identify the current problem to be solved

(2) **do**

(3) assess the *relevance* of the possible observations/computations

(4) perform the most relevant observation/computation

(5) **until** additional information is *useless*

(6) *decide* on the best action (sequence) to perform, u_α, using the available information

(7) change the external action to u_α

(8) update the agent's internal state to reflect the action choice

(9) **goto** 1

FIGURE 1.2: The generic incremental decision algorithm is an iterative process of problem identification (1), deliberation (2)-(5), and decision (6).

and efficiency of traditional feedback for fixed-point control of linear time-invariant systems, however for complex highly predictable problems, such as airline scheduling, planning approaches are superior. It is also possible to combine these approaches in a single system which contains planning, learning and programmed (reactive) components. For example the CIRCA system (Musliner, Durfee, and Shin 1993) combines reactive control modules with classical planning techniques.

1.2 The incremental decision model of on-line planning

In order to clarify the issues involved in designing on-line agents it is useful to view the agent's "life" as consisting of a sequence of decision problems where it must choose the next action to execute. To solve each decision problem the agent uses its sensors and computational resources to infer from its current information the additional information it needs to make its next decision. During this deliberation the external action remains fixed, either at some default "null" action or at the previously chosen action. Then at some point the agent decides that it must *stop deliberating and act* so it commits to its choice and changes its external action. This then changes the agent's decision problem so it begins solving the next incremental decision problem which is defined by the information it now has available (i.e. including the results of its previous deliberations) and the consequences of its previous action choice. This is the incremental decision model of on-line planning. The pseudo-code for the agent's internal operation is given in Figure 1.2.

This is a totally generic way of describing the operation of an agent function as any or all of the steps may be null. For example, in a reactive architecture only a single fixed duration

computation is performed based directly on the current observation to choose the action. However, in classical planning a long sequence of computations is performed to identify an optimal complete action sequence before the first action is executed, after which the stored plan is used open-loop to generate later actions. Finally, in learning systems the computations correspond to updates to the agent's stored information so later action choices are made with respect to this new information.

As noted previously by other researchers (Pemberton 1995; Russell and Wefald 1989) the incremental decision model of on-line planning makes clear that consideration of the agent's finite capabilities requires the agent design address four sub-problems;

1) **Decision Making** (step 6) – how does the agent make "good" action selections based upon the often incomplete (and and therefore uncertain) information it has available? These decisions must take account of the complexity induced uncertainty caused by the agents computational limitations.

2) **Exploration** (steps 3-4) – given the agent can only perform a limited amount of computation to gather information before a decision must be made, how does it structure its computations so it makes the "best" decisions possible?

3) **Stopping** (step 5) – given that further computation has both variable costs due to delaying decision making and benefits due to improvements in the information action choice is based upon, how does the agent decide when it is time to *stop thinking and commit to action*?

4) **Cycle Avoidance** (step 8) – given the limited *local* information the agent has available to make each decision, how does it modify its operation to avoid repeating past mistakes and ensure progress towards a solution?

These sub-tasks are highly inter-dependent. The best decision depends on how later decisions are made and hence on the exploration and stopping strategy used. The best exploration strategy depends on how the decisions are made and when exploration is stopped. The best stopping point depends on what further explorations will be made and how decisions are made.

The exploration and stopping problems are only important when computation is limited or costly, that is when the agent must be bounded rational. They are, *meta-reasoning* problems (Russell and Wefald 1989; Horvitz 1990; Parkes 1996) concerned with controlling the agent's internal problem solving process, and are required to ensure the agent makes best use of its limited computational resources – too little deliberation can lead to mistakes, while too much can lead to lost opportunities. The decision making sub-task is essentially the control problem which traditional planning and control techniques focus on.

Coping effectively with the uncertainty in decision outcomes is a key requirement for any solution to these problems. In many cases this uncertainty is not inherent in the problem but

due to the complexity of the problem and the computational limitations of the agent. For example, in theory the optimal first move in chess could be derived from the deterministic rules. However, this is impossible for any practical agent, so it must make its action choices in a state of "complexity-induced uncertainty" (Mayer 1994) over the outcome of its actions.

This thesis focuses on the use of Decision Theoretic techniques for coping with this uncertainty when solving the sub-problems faced by an incremental search agent. There are many possible ways of representing and making decisions under uncertainty, including; fuzzy sets (Zadeh 1965), Dempster-Shafer intervals (Shafer 1976), possibilistic representations, and Bayesian probabilities (Pearl 1988). However, *normative* or *decision-theoretic* (DT) techniques (Neumann and Morgenstern 1944; Russell and Norvig 1995) provide a demonstrably rational way to make decisions under uncertainty. Normative techniques are desirable because they have a sound theoretical framework and performance which can be analysed objectively. They also fit well into the on-line control model presented here if the objective function is treated as a utility function.

Decomposing the agent design problem in this way helps by allowing us to consider each of the sub-problems relatively independently (whilst, of course, acknowledging that their interactions may cause us problems later). This decomposition summarises the overall approach taken in this thesis to analysing designs for solving the on-line planning problem, where (after some initial discussion of the basic meta-control techniques available in Chapter's 2 and 3) we focus in turn on the decision making (Chapter 4), exploration (Chapter 5) and stopping and cycle avoidance problems (Chapter 6). Thus the reader should keep this model and sub-problem decomposition in mind as an overview of how the sections of this work fit together.

1.3 Test domains

Throughout this thesis the algorithms developed are tested experimentally on two test domains, namely the sliding puzzle problems and the grid problems. To test the scalability of the algorithms two levels of problem complexity are used, the 8 and 15 puzzle problems and the 10x10 and 100x100 grid problems. These two textbook domains were chosen because they are simple to understand and easy to manipulate—allowing techniques and hypotheses to be quickly evaluated and understood before moving on to more realistic problems. The puzzle domains have been the mainstay of heuristic search research for over 30 years, leading Gasching (1977) to call it the "fruit fly" of heuristic search. Thus, there is a wealth of existing work and results for comparison.

A generic sliding puzzle problem consists of a $n \times n$ grid containing $n^2 - 1$ numbered sliding tiles and a "blank", as shown in Figure 1.3. A specific problem gets its name from the number of tiles, thus the 8-puzzle uses a 3x3 grid and the 15-puzzle a 4x4 grid. The only legal action is to move one of the tiles horizontally or vertically adjacent to the blank into the blank's position. In this thesis all actions have fixed unit cost. A solution to the problem is a sequence of actions

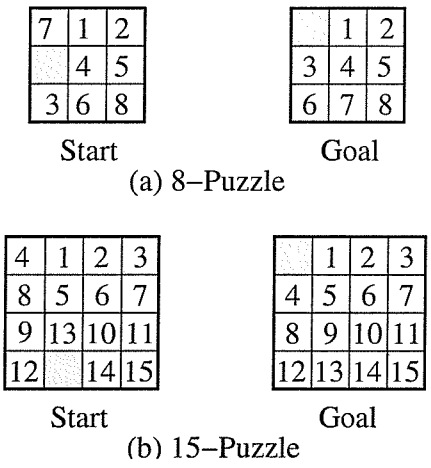


FIGURE 1.3: This figure shows example problems for the 8-Puzzle in part (a) and 15-Puzzle in part (b). Each puzzle consists of a set of numbered sliding tiles and a blank location (indicated by the shaded square) into which the horizontally or vertically adjacent tiles may move. The goal location shown is the one used throughout this thesis.

which transform a given starting tile configuration (state) into a particular goal configuration. In this thesis the goal state is fixed at those shown in Figure 1.3. The starting states are randomly generated by a 250 step random walk from the goal configuration. The standard heuristic function for sliding puzzle problems is the Manhattan Distance (Doran and Michie 1966), which is the sum over all tiles of the horizontal and vertical distance between the tile's current and goal locations. For example, the Manhattan distance for the tile 7 in the start state of Figure 1.3(a) is 3 (2 rows + 1 col) and the heuristic estimate for the start state is 5. The Manhattan heuristic is guaranteed to be an underestimate of the true number of actions required to reach the goal state. The state space size for a sliding tile puzzle on a $n \times n$ grid is $(n \times n)!/2$, thus the 8-puzzle state space size is (3x3)!/2 = 181440, and the 15-puzzle state space size is (4x4)!/2 = 10461394944000 = 1.04x10^{14}.

A generic grid problem also consists of a $n \times n$ grid of cells, as shown in Figure 1.4. There is a single agent on this grid that occupies one of the cells. The agent's only legal actions are to move from its current cell to one of the horizontally or vertically adjacent ones. Each cell has allocated to it a fixed randomly selected cost, which is the cost of any move *into* that cell. In this thesis the costs are chosen at uniform random from the range 1 to 10. Moves which would take the agent off the grid also have a randomly allocated cost but leave its current location unchanged. A solution to the problem is a sequence of actions which moves the agent from a given starting location (state) to a particular goal location. In this thesis the goal is always at the cell (n, n) and the starting state chosen at uniform random from the other locations. The standard heuristic for the grid problems is also the Manhattan distance, which is the sum of the horizontal and vertical distance between the agent's current and goal states. As the minimum cost of a move is 1 this is guaranteed to be an underestimate of the true cost of getting to the goal. The state space size for a generic $n \times n$ grid problem is $n \times n$, thus the 10x10 grid problem's state space size is 100 and the 100x100 grid problem's state space size is 10,000.

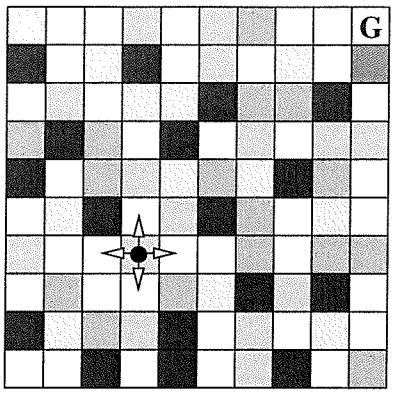


FIGURE 1.4: This figure shows an example 10x10 grid problem. The point marks the agent's current location and the arrows indicate the available moves. The shading of each cell indicates the cost of moving into that cell. The **G** in the top right marks the goal location. The 100x100 grid is similar with the goal in the top right, except the grid is 10 times bigger.

1.4 Contributions

The principle contribution of this thesis is a systematic investigation into the relative merit of decision theoretic approaches for solving the various sub-problems (i.e. decision making, search control, stopping and cycle avoidance) faced by an incremental search based on-line planning agent. The unique step-by-step approach used in this investigation allows the comparison of the relative importance of the sub-problems as well as the interactions between the sub-problems and the benefits offered by the approaches studied. The interplay between the different sub-problems turns out to be crucial, and often unexpected.

Another contribution is a unique formal analysis of the on-line planning problem and incremental search based on-line planning agents. This analysis clarifies the many assumptions and approximations inherent in existing incremental search algorithms. Further it suggests areas where it may be beneficial to relax existing assumptions or introduce new approximations. This analysis led to another contribution which is the development of novel efficient algorithms for decision making, search control and stopping based on using parameterised distributions for value estimation.

The meta-control based on-line planning agent development work in this thesis has been published as;

Farquhar, J.D.R. and C. Harris (2001, December). A proposal for an on-line real-time planner. In J. Levine (Ed.), *Proceedings of the Twentieth Workshop of the UK Special Interest Group on Planning (PlanSig01)*, University of Edinburgh, Edinburgh, pp. 276–282.

Farquhar, J.D.R. and C. Harris (2002, July). An Empirical Comparison of Incremental Search Algorithms for On-Line Planning In *Working Notes of the AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, Edmonton, Alberta, Canada.

It is intended that further publications will present the remaining work and thesis conclusions.

1.5 Thesis outline

The remainder of this thesis is organised as follows;

Chapter 2 presents a formal state-space model of on-line planning problems as reward maximisation. The value of a state is identified as a key concept for solving on-line planning problems. Decision theory is introduced as a principled way of making *rational* decisions under uncertainty and used to formalise the concept of *bounded optimality*. The chapter concludes with a discussion of the conditions under which explicit deliberation, such as search, may be useful.

Chapter 3 focuses on the architectural design of deliberative on-line agents. A formal model of the meta-control problem is introduced and shown to be equivalent to a super-set of the original on-line planning problem. It is then shown how the meta-control problem can be approximated as a Markov Decision Problem (MDP). The remainder of this chapter develops the value estimate MDP approximation which, based upon the subjective value equivalence assumption, forms the basis for all the meta-control algorithms developed in this thesis. A review of other meta-control techniques is provided at the relevant points throughout this chapter.

Chapter 4 examines the decision making component of an on-line planning agent. Efficient value estimation is identified as the key step in making effective decisions. Incremental search is proposed as a method of focusing on the most important local features for value estimation and spreading decision making effort over task execution. Learning or inference techniques can be used to develop functions which estimate action values given the information contained in the local search space. New and existing techniques for value estimation are discussed. This chapter concludes by empirically comparing the performance of the individual decisions made using these value estimation techniques.

Chapter 5 is concerned with the search control component of an on-line planning agent. Based upon the value estimate MDP it is shown that efficiently estimating the expected value of computation (EVC) is key for effective search control. By identifying how a computation could affect the agent's decision making the myopic EVC, root state sensitivity, and expected step size EVC approximations are described. After discussing how these estimates can be efficiently implemented this chapter concludes by comparing their performance experimentally when making single decisions.

The first part of Chapter 6 examines the stopping component of an on-line planning agent. It is shown that whilst stopping is a meta-control problem performance can be improved by using more accurate estimates for the expected costs and benefits of all further search. The second part of Chapter 6 examines how an incremental search agent can escape local optima and avoid cycling forever. Existing cycle avoidance techniques are discussed and value update identified

as the most applicable for on-line planning. The final part of Chapter 6 compares the most promising incremental search agent designs developed in this thesis for solving complete on-line planning problems.

Chapter 7 brings together the work of this thesis, draws conclusions and identifies possible directions for future development.

Chapter 2

The agent design problem

This chapter presents the agent design problem in a more formal way, introduces decision theory and bounded rationality, and discusses when deliberative approaches are most appropriate to solving it.

2.1 The general agent design problem

Informally, a general agent design problem concerns a pair of interacting dynamical systems; the agent which we must specify, and a pre-defined environment within which the agent will "live". The agent interacts with the environment by taking as input observations and outputting actions. The observations return partial information about the current state of the environment. The actions affect the environment in some way changing its current state. The designer's problem is to construct the agent dynamics such that its interacts with the environment in some pre-specified desirable fashion. Thus the agent design problem is in fact just a "not so special" instance of an optimal control synthesis problem (Jacobs 1974), with the environment the plant and the agent the controller.

2.2 A state-space model

Due to its flexibility and intuitive nature (following (Russell and Subramaniam 1995)) we adopt a discrete state space formalism for describing the on-line planning problem. For simplicity we will also use a discrete time representation.

We take a very agent-centric view where the environment contains *everything* external to the agent which can influence its operation. In particular, the user and their requirements are part of the environment. Hence, for a taskable agent an additional observable part of the environment must include some indication of the users current requirements, such as a key-pad to enter the goal destination.

2.2.1 The environment

We begin by defining the environment.

Definition 2.1 (Environment, E). An environment consists of;

- a set of states, $\mathcal{X}$, where each state, $x \in \mathcal{X}$, is a unique complete description of the current situation,
- a distinguished initial state, $x_0 \in \mathcal{X}$, and time, t_0,
- a set of actions, $\mathcal{U}$,
- a state transition function, $f_x(x, u, t) \rightarrow x'$ for $x, x' \in \mathcal{X}$ and $u \in \mathcal{U}$, which describes how the agent's actions modify the current state, x, at time t,
- a set of observations $\mathcal{Y}$,
- an observation function, $f_y(x, t) \rightarrow y$, for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, which describes how the agent's observations depend on the current state, x.

Notice, by construction and without loss of generality this model is deterministic and Markov, as we assume each state contains all the information relevant to the future dynamics of the system. Any non-Markov and/or non-deterministic problem can be modelled in this way by extending the state description to include additional unobservable noise or history terms which encode all the relevant information upon which future transitions or observations could depend.

In abstract terms an agent can be described as a function which maps from observation sequences into actions, $\pi(y_{0:t}, u_{0:t-1}, t) \rightarrow u_t$. Adopting the term from discrete optimal control we will call this function a *policy* to distinguish this abstract representation from the more detailed agent models developed later[1].

When an agent is placed in a particular environment E it will produce a particular trajectory, which consists of the sequence of actions $u_{0:t}$ the agent executes in response to the observations it receives, and the sequence of states $x_{0:t}$ the environment passes through in response to these actions. We will use the notation $T(\pi; E) \rightarrow (x_{0:t}, u_{0:t})$ to denote the trajectory generated by an agent implementing a policy π when started in environment E.

2.2.2 The objective function

We assume the user exists in the external world and knows nothing about the agent's internal state. Thus, the user's requirements can depend only on the agent's externally observable

[1]Russell and Subramaniam (1995) use the term "agent function" to describe this mapping. I prefer to use policy because it captures better its abstractness and makes clear the connection to the policies computed by dynamic programming algorithms.

behaviour, i.e. the trajectory it generates. Hence, we model the user's requirements in terms of an objective function over agent trajectories.

Definition 2.2 (Objective Function, R). The objective function is a real-valued function of the agent's trajectory, $R(x_{0:t}, u_{0:t}, 0{:}t) \to \mathbb{R}$.
Note, the explicit time-dependence captures the real-time aspects of the problem.

Further, as is usual in optimal control problems, we assume without loss of generality that the objective function is *recursive*, *Markov* and *additive* such that,

$$\begin{aligned} R(x_{0:t}, u_{0:t}, 0{:}t) &= R(x_{0:t-1}, u_{0:t-1}, 0{:}t{-}1) + r(x_0, u_t, t), && (2.1)\\ &= \sum_{\tau \in 0..t} r(x_\tau, u_\tau, \tau) && (2.2)\\ &= r(x_0, u_0, 0) + R(x_{0:t}, u_{0:t}, 1{:}t). && (2.3) \end{aligned}$$

where $r(x_0, u_0, t)$ is the *immediate* reward for taking the given action in this state at this time, and $R(x_{0:t}, u_{0:t}, t_1{:}t_2)$ is the total accumulated reward for the time's t_1 to t_2 of the trajectory segment $(x_{0:t}, u_{0:t})$. This is still a general reward function as any reward function can be represented by suitably extending the state definition to include the necessary additional information.

This objective function is representative of many real-world situations where the agent has to achieve multiple conflicting objectives. For example, consider our factory delivery agent which has to deliver packages to n locations as rapidly as possible. Generally, the correct delivery of each package has some value dependent on its importance but independent of which other packages have been delivered, hence the value of multiple package deliveries is just their summed value.

It should be emphasised that the actual objective function exists externally to the agent. It is part of the design problem and only the designer need have direct knowledge of it. Whilst, as noted above, a taskable agent should have some indication of its current objective, this does not mean the agent requires an explicit representation of the actual objective function. For example, a taskable navigation robot needs to know its current destination but not necessarily that every second of delay reduces the objective function by say 6. It is part of the designer's task to decide whether or not explicitly representing the objective function within the agent is useful.

The objective function defines the agent's current objective, which as noted above, forms part of the agent's environment. As the environment and objective function may vary independently of each other, we will use E, R when we wish to emphasise that in this environment the agent's objective function is R. We will use the term *task* to refer to the particular environment/objective function combination which the agent is to solve.

2.2.3 Designing Optimal Agents—Value functions and the Bellman equation

By definition, the objective function depends on the complete (possibly infinite) trajectory the agent takes, and so the quality of a policy depends on the complete trajectory it induces. Thus, we define the situation dependent quality, or *value*, of a policy as;

Definition 2.3 (Value, $V_\pi(x_0, t_0; E, R)$). The value of a complete policy π with respect to a task E, R with initial state x_0 and time t_0 is defined as the total objective value accumulated from the future trajectory, T, induced by the policy. That is,

$$V_\pi(x_0, t_0; E, R) = R(T(\pi; E, R, x_0, t_0)). \tag{2.4}$$

Now, when in the situation x_0, t_0 the policy will execute the action $u_0 = \pi(f_y(x_0), \cdot, t_0)$. Thus, using $[u, \pi]$ to denote the policy which initially executes the action (sequence) u and then operates according to policy π we can write,

$$V_\pi(x_0, t_0; E, R) = V_{u_0,\pi}(x_0, t_0; E, R) = R(T([u_0, \pi]; E, R, x_0, t_0)) \tag{2.5}$$

Combining (2.3) and (2.5) the policy value can be decomposed into an immediate reward from the initial action u_0 and a future value of the rest of the trajectory,

$$V_\pi(x_0, t_0; E, R) \quad = \quad r(x_0, u_0, t_0) + R(T(\pi; E, R, O_0, x_1), 1{:}\infty), \tag{2.6}$$

where $x_1{=}f_x(x_0, u_0, t_0)$, $O_{0:t}{=}(u_{0:t}, y_{0:t})$ is an observation sequence, $T(\pi; E, R, O_{0:t-1}, x_t)$ is the conditional trajectory the policy π induces from the state x_t *given* that it has already seen $O_{0:t}$. Note, this augmented state is necessary as by definition the policy, π, can depend on the trajectory taken so far.

Extending (2.6) to include an arbitrary initial observation prefix, we obtain,

$$V_\pi(x_t, O_{0:t-1}; E, R) = V_{u_t,\pi}(x_t, O_{0:t-1}; E, R) = r(x_t, u_t, t) + V_\pi(x_{t+1}, O_{0:t}; E, R) \tag{2.7}$$

where $u_t = \pi(f_y(x_t), O_{0:t-1}, t)$ and $x_{t+1} = f_x(x_t, u_t, t)$.

Equation (2.7) is the familiar Bellman equation (Bellman 1957) from dynamic programming (DP) for the enlarged space of all state and observation prefixes.

By maximising this value over all possible policies we could find the optimal policy for this task, (denoted $\pi^*_{E,R}$), i.e.

$$\pi^*_{E,R} = \operatorname*{argmax}_{\pi \in \Pi} V_\pi(E, R) = \operatorname*{argmax}_{\pi \in \Pi} V_\pi(x_0; E, R). \tag{2.8}$$

However, performing this maximisation over the extremely large space of all possible policies is likely to be computationally expensive. For a more efficient alternative, note that the optimal policy has a corresponding unique optimal value function, V^*. By definition V^* must satisfy

the Bellman equation with respect to the action chosen by the optimal policy. Now, since the optimal policy picks the best action in each situation, i.e. the one which maximises the future value, the optimal value function must satisfy Bellman's optimality equation,

$$V^*(x_t, O_{0:t-1}; E, R) = \max_{u_t}[V_{u_t,\pi^*}(x_t, O_{0:t-1}; E, R)], \quad (2.9)$$

$$= \max_{u_t}[r(x_t, u_t, t) + V^*(x_{t+1}, O_{0:t}; E, R)]. \quad (2.10)$$

Because this operation backs-up information from successor states to their predecessors it is also called the dynamic programming backup operation. This equation can be used to compute the optimal value function directly in a number of different ways, the most direct being dynamic programming (Bellman and Dreyfus 1962). The optimal policy can be found from the optimal value function using only *local* information by maximising the value gradient. That is,

$$\pi^*_{E,R}(x_t, O_{0:t-1}) = \operatorname*{argmax}_{u_t}[r(x_t, u_t, t) + V^*(x_{t+1}, O_{0:t}; E, R)]. \quad (2.11)$$

In words we can say an optimal policy is one which for any observation sequence realisable in the environment executes the action which maximises the final objective value *when all future actions are selected according to the optimal policy.*

The two equations (2.10) and (2.11) are the key computing optimal and near optimal policies. Thus, with respect to a particular environment and objective function, the designer (theoretically at least) can compute the optimal policy. Unfortunately, this is not generally possible for a number of reasons, one being the fact that the designer will usually be unsure of the agent's exact environment and objective function.

2.2.4 Design time task uncertainty

At design time the designer will be uncertain about the exact environment the agent will be operating in, either because his models are incomplete or because the environment is non-deterministic. Further, the designer does not know in advance with certainty what the user's requirements, and hence the agent's objective, will be. Thus the designer will be uncertain which particular task the agent needs to solve. It is a key part of the designer's job to cope with this uncertainty and design the agent so it performs well in all the the possible environment/objective function combinations. Optimising the agent design over this set of possible tasks increases the difficulty of the designers problem.

Further, in general the agent will suffer from some form of *perceptual aliasing* where, due to its limited sensing and computational abilities, the agent is unable to distinguish between some set of tasks. Thus the designer will need some way to trade-off improved performance in one subset of the indistinguishable tasks against reduced performance in another. To see why consider an agent which implements a policy, π. By definition this policy selects actions based upon observation sequences. Thus, if all observation sequences uniquely identify the current

environment, $E \in \mathbf{E}$, and objective, $R \in \mathbf{R}$, then the uniquely identify the optimal action. Thus, the designer can treat each task independently and use the preceding definition, (2.10) and (2.11), to compute an optimal policy for each task, $\pi^*_{E,R}$. The optimal policy over all possible tasks can then be found by combining the task specific policies, i.e. $\pi^* = \bigcup_{E \in \mathbf{E}, R \in \mathbf{R}} \pi^*_{E,R}$. However, if different environments and objectives require different optimal actions but produce the same observations then there is some ambiguity in the definition of an optimal policy. For example, given indistinguishable environments E_1 and E_2 with respective optimal actions $u^*_{E_1}$ and $u^*_{E_2}$, is it better to be sub-optimal in E_1 or E_2 or both? To resolve this ambiguity we need some way to extend the objective function and optimality criterion so we can trade-off the different values of the indistinguishable uncertain situations.

2.2.5 Decision Theory and rationality

Many ways of performing this trade-off are possible, such as minimising the worst case performance, or maximising the probability of optimality. However, *normative* or *decision-theoretic* (DT) techniques (Neumann and Morgenstern 1944; Savage 1954) provide an intuitive and demonstrably rational approach with an extensive theoretical and empirical heritage. At the heart of decision theory are the axioms of *utility* and *probability*, which respectively provide a comprehensive theory for representing the relative desirability of alternatives and the agent's uncertainty, i.e.

$$Decision\ Theory = Utility\ Theory + Probability\ Theory.$$

Utility theory requires that we represent the users requirements in terms of a real-valued *utility function* over possible outcomes. This function has the property that the ordering of outcomes with respect to increasing utility is identical to the ordering with respect to the increasing preference. Utility is thus an aggregate measure of all dimensions of an outcomes worth, including cost, benefit and risk, relative to the agent and its current situation. The *utility principle* (Russell and Norvig 1995, p 473) shows that such a utility function exists providing the preferences meet certain reasonable consistency relations (known as the axioms of utility). For example, preferences should be transitive to prevent our agent being used as a *money pump* (Binmore 1992, p 95). Probability theory requires that we represent any uncertain information in a way consistent with the standard axioms of probability.

Decision theory, in its modern form, originates with the work of Ramsey (1931), Neumann and Morgenstern (1944), Cox (1946) and Savage (1954). A complete description of decision theory for the uninitiated is well beyond the scope of this dissertation – the interested reader should see the introductory texts (Savage 1954; Raiffa and Schlaifer 1961; Watson and Buede 1987) or (von Winterfeldt and Edwards 1986). For applications of decision theory in AI see (Russell and Wefald 1989; Horvitz 1990; Dean and Wellman 1991; Doyle 1992).

Given utility and probability functions defined correctly, decision theory simply says that when we are uncertain the most rational thing we can do is try to ensure we do as well as possible *on average*, i.e. we should maximise expected utility. Formally, the fundamental theorem of decision theory defines a rational agent as;

Definition 2.4 (Rational agent). A rational agent selects actions *as if* it was; (1) assigning utilities to the possible outcomes of actions, (2) assigning probabilities to the outcomes which might occur given *its current* uncertainty, and (3) choosing the actions which *maximise expected utility*.

Notice, that the definition of a rational agent does not require the agent necessarily computes expected utilities, know what one is or is even aware of its own preferences; only that, if it is to be rational, it must *act as if* it was performing this maximisation. Also, this definition is *subjective* depending on the agent's personal assessments of outcome probabilities (for this reason DT is sometimes called subjective rational decision theory). Thus, when talking about decision theoretic rationality one must be careful to define the information available with respect to which rationality is being assessed.

In the on-line planning case it is clear that the objective function serves a similar role of specifying preference as the utility function in decision theory. Hence, in the remainder of this thesis, we will assume that, (1) the objective function *is* a correct utility function, and (2) that all uncertainties (in the design problem) are represented using correct probabilities. Given these assumptions the optimal action, with respect to complete and correct information about the uncertainties in the current environment and utility function, is clearly the one which maximises expected utility. Thus we obtain Russell's (1995) definition of Perfect Rationality (this is equivalent to Good's (1971) "Type I" rationality, and Simon's (1982) "substantive rationality"),

Definition 2.5 (Perfect rationality). A perfectly rational agent for a set of environments, $\mathbf{E}$, and utility functions, $\mathbf{R}$, selects actions as if it was maximising the expected utility of the of the outcomes of the actions with respect to the distribution of environments and utility functions.

This is the extended definition of optimality under uncertainty which we were seeking. We can now define an optimal policy, π^*, with respect to the environment and objective uncertainty as one which maximises expected utility with respect to the distribution of environments and utility functions,

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_{R \in \mathbf{R}, E \in \mathbf{E}} \Pr(E, R) R(T(\pi, E, R)). \tag{2.12}$$

We can also now define the value of a policy (which is maximal for the optimal policy) as its expected value with respect to the environment and objective uncertainty,

$$V_\pi = \sum_{R \in \mathbf{R}, E \in \mathbf{E}} \Pr(E, R) V_\pi(x_0, t_0; E, R), \tag{2.13}$$

where x_0, t_0 are the initial state and time for the environment E, R.

We have now all the tools in place to specify the agent design problem.

Definition 2.6 (General agent design problem). Given a set of possible environments, $\mathbf{E}$, and utility functions, $\mathbf{R}$, and a distribution over them $\Pr(E, R)$, the agent design problem is to design an agent, A, such that the policy it implements, π_A, has maximal expected utility. This is equivalent to solving the following constrained optimisation problem,

$$\text{maximise} \quad \sum_{R \in \mathbf{R}, E \in \mathbf{E}} \Pr(E, R) R(x_{E,R}(0{:}\infty), u_{E,R}(0{:}\infty)) \tag{2.14}$$

$$\text{such that} \quad \begin{aligned} x_{E,R}(t{+}1) &= f_x(x_{E,R}(t), u_{E,R}(t), t | E, R), \\ u_{E,R}(t) &= \pi_A(y_{E,R}(0{:}t), u_{E,R}(0{:}t), t), \\ y_{E,R}(t) &= f_y(x_{E,R}(t), t | E, R), \end{aligned} \quad \begin{aligned} & t \in 0..\infty, \\ \text{for } & E \in \mathbf{E}, \\ & R \in \mathbf{R} \end{aligned} \tag{2.15}$$

where $\pi_A(y_{0:t}, t)$ is the free parameter and for an arbitrary variable z, $z_{E,R}$ denotes the instance of variable z associated with environment E and objective function R.

Directly computing such an optimal policy, under different domain assumptions which limit the types of environment and utility functions considered, is the focus of most traditional work in optimal control, classical planning (Drummond and Tate 1989), (Russell and Norvig 1995, Ch. 11), and stochastic control (Bertsekas and Shreve 1978).

2.3 Designing bounded optimal agents

Decision theory via the definition of perfect rationality allows us to, theoretically at least, determine how a perfect agent should act. In theory then, all that remains to optimally solve the on-line planning problem is to construct an agent which *implements* an optimal policy by providing it with the data and methods required to compute it. This is essentially the traditional top-down approach to agent design, i.e. start with an abstract problem description, devise a method for solving it optimally and then implement this method in an agent. By construction, such an agent will solve our on-line planning problem optimally *given sufficient (computational) resources*. Russell and Subramaniam (1995) use the term *calculative rationality* to describe such agents. Unfortunately, as is well known, deterministic planning is intractable (Chapman 1987; Bylander 1994). Hence, as on-line planning problems are a super-set of planning problems, on-line planning is also intractable. Thus, for realistic problem sizes, any agent which implements an optimal policy will require infeasible computational resources.

One reason deterministic planning is so difficult is because the space of possible initial environments and objective functions are huge, at least proportional to the state space size which is exponential in the number of dimensions. Thus, any agent which is to solve *all* problems in this set will require either time (to explore all possible trajectories) and/or space (to store the

optimal solution for each possible state/objective pair) which is exponential with respect to the problem dimensionality.

Russell and Subramaniam (1995) provide a simple example. Consider an agent which is to play chess under tournament rules. This is clearly an on-line planning problem and is well known to be very difficult to solve with the space of possible environments equal to the state space size of $\approx 10^{40}$. Thus, an optimal agent which compute(s) an optimal policy for this problem requires, either enough memory to store the optimal policy directly, or a fast enough architecture to compute an optimal policy for each situation encountered so it can move before it runs out of time. Implementing such an agent is impossible with current technology (and probably always will be). Further, the success of current chess systems shows that true optimality is unnecessary for intelligent performance.

Thus, as the on-line planning problem is intractable in general, we cannot expect any realistic agent to solve it optimally. Therefore, if we wish to use decision theory, and specifically the principle of maximum expected utility, to guide agent design, we must modify the definition of rationality to require only feasible amounts of computation.

Simon's (1982) concept of *bounded rationality* is one such re-definition by requiring an agent to find some satisficing solution which meets some "aspiration criteria" rather than a DT optimal one. This simplifies the problem by allowing a much larger range of possible solutions. However, this redefinition seems somewhat arbitrary and even finding aspiration criteria for good satisficing solutions of low enough cost is difficult.

A more recent, principled and intuitive re-definition is *bounded optimality*, which simply requires the "agent does as well as possible given its computational resources" (Russell and Wefald 1991). This removes the requirement that the agent optimally solve intractable problems, and allows the designer to consider trade-offs between complexity and decision quality in order to make best use of the limited computational resources available.

To make this definition more precise we need some way of defining the agent's computational limitations. Any agent can only choose actions based upon its currently available information, i.e. the current observation, y_t, and some limited modifiable internal state, b_t. Hence, a more concrete way to specify the solution to an on-line planning problem is in terms of an agent function or behavioural mapping (Arkin 1998, pp 89), $A(y_t, b_t) \rightarrow (u_t, b_{t+1})$, which maps from the agents current observable state (i.e. its current observation and internal state) to its current action choice. Notice that this function operates continuously with one observation, internal state up-date and external state update (via the action generated which may include a null "I'm thinking" action) per time-step. Thus, even thinking operations cause the external state to change via their associated state side-effects. The agent function induces (implements) an equivalent policy, π_A, which can be found by applying the agent function iteratively to each observation sequence.

To represent the agent's computational limits we follow Russell and Subramaniam (1995) and decompose the agent function into an architecture, M, and a program, l. The architecture runs the program and defines the agent's computational resources by restricting the set of mappings the agent function can implement to those in the architecture's language, $\mathcal{L}$. The program defines which mapping from the architecture's language the agent performs in each case. Thus an alternative representation of the agent function is $M(y_t, b_t; l) \rightarrow (u_t, b_{t+1})$. Thus, we obtain Russell and Subramaniam's definition of bounded optimality,

Definition 2.7 (Bounded optimality). Given an agent architecture, M, and a distribution over environments and utility functions, $\Pr(E, R)$, a bounded optimal agent is one whose agent program, l^* has maximal expected utility. Thus, if $\pi_{M,l}$ represents the policy (i.e. observation sequence to action mapping) induced by program l on architecture M, then,

$$l^* = \arg\max_{l} \sum_{R \in \mathbf{R}, E \in \mathbf{E}} \Pr(R, E) R(T(\pi_{M,l}, E, R))$$

We can now define a refinement to the general agent design problem which takes account of the agent's computational limitations.

Definition 2.8 (Bounded agent design problem). Given a set of possible environments, $\mathbf{E}$, and utility functions, $\mathbf{R}$, with a distribution over them $\Pr(E, R)$ and an agent with architecture M, the agent design problem is to design an agent program l such that the policy the agent implements has maximal expected utility. This is equivalent to solving the following constrained optimisation problem,

$$\text{maximise} \quad \sum_{R \in \mathbf{R}, E \in \mathbf{E}} \Pr(E, R) R(x_{E,R}(0{:}\infty), u_{E,R}(0{:}\infty)) \tag{2.16}$$

$$\begin{aligned} & x_{E,R}(t{+}1) = f_x(x_{E,R}(t), u_{E,R}(t), t | E, R), && t \in 0..\infty, \\ \text{such that} \quad & (u_{E,R}(t), b_{E,R}(t{+}1)) = M(y_{E,R}(t), b_{E,R}(t); l), && \text{for } E \in \mathbf{E}, \\ & y_{E,R}(t) = f_y(x_{E,R}(t), t | E, R), && R \in \mathbf{R} \end{aligned} \tag{2.17}$$

This seems like progress. We have removed the requirement that the agent solve intractable problems in real-time, requiring only that it "do the right thing". We have also specified a complete procedure for finding such an agent – search through the finite space of all possible agent programs until you find the program whose induced policy has maximal expected value with respect to the distribution of environments and utility functions.

Unfortunately, one immediately apparent problem with this proposal is that we have not actually made the problem any more tractable, just allowed the agent to ignore its intractability by forcing the designer to solve the *harder* problem of optimisation over agent designs. As Tash (1996)[pp 26] says "the entire burden of decision theoretic computation has not been removed, but merely shifted to the designer".

As an example from (Russell and Subramaniam 1995), consider again our chess agent, and suppose the architecture has a total program memory of 8 megabytes. There are $2^{2^{26}} \approx 10^{10,100,000}$ possible programs that the architecture can represent. To find the bounded optimal chess computer requires the designer search through this space to find the small number of programs which play legal chess and for each program compute the expected value of this program over all possible legal chess positions. This is a harder problem than simply finding the optimal policy, as it requires both search through the huge space of possible programs *and* the evaluation of each program on the huge space of possible chess positions.

Further, the designer cannot, as is assumed in the traditional approach to agent design, simplify the design problem and guarantee agent optimality by requiring that the agent internally use rational decision making processes without first solving the full agent design problem to prove such a design is optimal. Thus, the optimality of an agent can only be assessed from a viewpoint external to the agent, and specifically not bound by its resource constraints. As Tash (1996) says, "ultimately, judgement of the [bounded] rationality of the agent rests in the hands of the designer". This is a severe blow to the traditional approach to agent design which attempts to produce rational agents by requiring that they make decisions using the same formalised optimality criteria as the designer would use to judge it, i.e. for bounded rational agents $\textit{Rational Algorithm} \not\Rightarrow \textit{Rational Agent}$

Note, this does not mean that an agent which makes rational, i.e. utility maximising, decisions on the basis of its current subjective beliefs is necessarily sub-optimal. Only that is is up to the agent designer to prove that this is the most effective use of the agent's resources. Indeed, it is clear that in certain circumstances a rational design will be bounded optimal. For example, when the optimal policy is simple enough and the agent powerful enough for it to store or compute the policy on-line with its available resources.

Thus, not only is the designer solely responsible for ensuring the agent design is rational, but they must solve an intractable design problem to do so!

2.4 Designing bounded rational agents

The above analysis has shown that when designing agents we must take careful account of their computational limitations, but provides no advice on how to design computationally limited agents. So far all we seem to have achieved is to show that the traditional agent design method which assumes calculative rationality is not generally effective for realistic computationally limited agents. So how can we solve the bounded agent design problem, given that explicitly optimising over all possible agent programs is generally impractical?

Deciding how to answer this question is at the heart of the recent deliberative/reactive and earlier declarative/procedural debates in Artificial Intelligence. Note, learning is somewhat orthogonal to this debate as one can learn both reactive procedures and abstract models.

Simply put, the deliberative/declarative position is that the best way to generate a good policy is for the agent to *explicitly* reason about the approximate quality of possible policies with respect to the given performance criteria. Thus, deliberationists advocate retaining the traditional top-down design philosophy with modifications for tractability.

The reactive/procedural position, as exemplified by Brooks (1991) or Arkin (1998), is more pragmatic saying simply that so long as the agent function generates good actions how it got there is irrelevant. Reactivist's then point out that as explicit representations can easily deviate from reality they are an unnecessary source of error which we should deliberately avoid (both for the designer and agent). Instead they suggest we should develop good agent functions based directly upon the agent's actual operating environment using incremental bottom up design techniques.

Like many researchers I remain neutral in this debate. It is clear that deliberative systems have significant problems due to their intractability and the need to keep the representations grounded. However, it is also clear that reactive systems present the designer with a significantly harder design task and may have problems with excessive internal state requirements when there are insufficient local environmental queues to *simply* "determine actions that have no irreversibly bad downstream effects" (Kirsh 1991). For example, a reactive implementation of a chess agent would require exponential amounts of memory to hold the optimal policy.

Between the reactive and deliberative extremes lie a number of hybrid techniques. For example, one can use explicit representations at design time to find high quality approximate policies, and then use compilation techniques (Russell and Wefald 1989, pp 40) to turn these into a, hopefully more compact and efficient, agent function, which may be purely reactive. This is the approach commonly used in control theory, and has recently been suggested for POMDPs (Boutilier, Dean, and Hanks 1999) and MDPs. Alternatively, one can construct a hybrid system which combines deliberative and reactive components within a single system to get the benefits of both without their drawbacks. A common approach in this vein is to use high speed reactive components to provide robust low level action executors which are managed by a slower deliberative component to achieve high level goals. Examples of this approach include the CIRCA system (Musliner, Durfee, and Shin 1993) and Sloman's (1999) CogAff architecture. Thus reactive implementations and explicit representations are not mutually exclusive.

Due to their different strengths and weakness I believe that the choice of agent design depends on a number of factors, including; (1) the structure of the problem (e.g. can good actions be determined based on local observations), (2) the quality of the designers information (i.e. do we have accurate domain models), and (3) the agent's abilities (i.e. do we have enough time for complex deliberation). For example, in relatively simple well understood problems directly constructing a reactive controller can be very effective. However, it may be that direct implementation of an optimal policy requires too much space making deliberative or explicit representations more efficient.

What is clear from this discussion is that whilst reactive procedures can be very effective in certain situations, there remain cases where deliberative methods are still king, namely when easily grounded domain models with sufficient predictive power are available *and* locally observable environmental queues alone are insufficient to simply identify actions which provide the required performance level.

Examples of this type of situation abound anywhere an agent may be required to solve a combinatorial problem as a sub-task of its operation. The classic examples in this vein are traditional board games such as chess or Othello, here the models are exact and very easily grounded, and (with our current knowledge) the best action is a very complex function of the current board state. Thus, it is not surprising that much work in traditional real-time AI has concentrated on game playing systems (Good 1968; Hamilton and Garber 1997; Russell and Wefald 1989; Baum 1993). Note, that most game solvers contain some reactive components in the form of their opening and end-game books, but rely on deliberative procedures for the majority of the mid-game as (with our current state of knowledge) reactive implementations would require excessive memory to encode the lookup tables.

Another classic example is high level route planning. High level models (maps) are reasonably accurate and relatively easily grounded. However the presence of dead-ends means local action choice can be misleading. In this case the utility of deliberative methods is less clear. Brooks (1991) describes a reactive robot called "Toto" which performs high-level path planning by performing what is essentially a potential field calculation on a physically encoded topological map. The reactive controller then uses this potential field as an additional local observation to avoid dead-ends. This is a good example of how compilation can be used to turn a deliberative specification (presumably Djkstra's algorithm) into a fast distributed reactive component – though in this case one would expect a deliberative system to be simpler, faster, and more powerful, in that one can easily extend the deliberative version to solve relatively large externally provided maps.

Thus deliberative methods still have some utility in on-line contexts— though likely only as a component in a larger hybrid system. However, in order to use deliberative components one must still solve the problem of ensuring they are bounded rational, i.e. provide the maximal action selection performance for the expended computational effort.

Chapter 3

Designing deliberative on-line agents

The preceding discussion has shown that deliberative methods still have their place in on-line contexts – providing they can be modified to operate within the agent's time and space limitations[1]. Thus the main focus in the sub-field of real-time AI is developing deliberative algorithms with the necessary space and time performance. The general approach is to meet the agent's resource limitations by sacrificing decision quality through approximate solution methods. Thus, the key to developing effective deliberative on-line agents is developing efficient complexity limiting approximations and solution methods.

Two main approaches have been used to obtain resource limited deliberation;

- Bounded deliberation, where the deliberation component is limited in some way, such as by limiting search depth, so its resource-usage and solution quality can be accurately predicted, and worst case performance guarantees given.
- Dynamic deliberation, where the deliberative component is designed so it can dynamically control its operation to utilise all the available resource to maximise agent performance.

These two approaches have different advantages for different contexts. The run-time performance guarantees of bounded deliberative components mean they can be used with conventional real-time system design and scheduling techniques, with formal correctness guarantees. This is important for low level mission critical applications where a response must be guaranteed within strict very short time limits (<100 ms). Hence they have found most use in aerospace robotics contexts (Myers 1996). Dynamic deliberation is used where there is high variability in problem difficulty and resource availability. Thus, it tends to be used for less mission critical high level optimisation tasks where longer time frames and harder problems provide variability for it to exploit.

[1] Note, even a deliberative sub-component will have to make some less stringent resource guarantees to be useful.

3.1 Bounded deliberation

Ingrand et al.'s Procedural Reasoning System (PRS) (Ingrand, Georgeff, and Rao 1992; Ingrand and Georgeff 1990; Georgeff and Ingrand 1989) is one example of a bounded deliberation system. This uses a pre-compiled hierarchical plan-fragment library coupled with a continuously updated world model to dynamically select actions. The amount of deliberation is bounded because, i) the depth of the plan hierarchy is bounded, ii) the plan-fragment library is designed to ensure only a bounded subset of the plan fragments will be applicable in any current state, iii) plan-fragment matching and choice are implemented as bounded time procedures. Thus the amount of work required to find a complete plan decomposition for the current situation is always bounded. Firby's (1989) Reactive Action Package (RAP) planning system achieves bounded deliberation in a similar way – though it allows plan fragments to be generalised to be arbitrary programs written in their special language.

Mok (1990) presents an alternative for forward chaining production systems, which provides a way of computing strict bounds on the execution time of database updates by imposing limitations on the rule types. Specifically, they require that, (i) rules only assign constants to variables, (ii) only one rule is fired at a time, and (iii) all rules are pairwise compatible in the sense that two rules cannot both be enabled and attempt to assign different values to the same variables. These bounds can then be used to design and debug production systems to ensure bounded real-time operation. Chen and Cheng (1994) and Tsai and Cheng (1994) present extensions of this technique which firstly significantly speed up the bound computation and secondly present rule-base transformation techniques which allow bound calculation(s) to be efficiently performed for the more expressive OPS-5 language (Tsai and Cheng 1994).

The most common method of bounding deliberation used in practise is to impose a fixed depth bound on search based deliberative procedures. This is the approach used in model predictive control (MPC) (Mayne, Rawlings, Rao, and Scokaert 2000) to control constrained non-linear systems. Bounded, fixed depth lookahead is also the most common way of implementing Korf's (1990) Real-Time A* (RTA*) algorithm. MPC and RTA* are discussed in more detail in Chapter 4 as examples of the incremental search approach to on-line decision making.

3.2 Dynamic deliberation

Dynamic deliberation provides a straightforward way of incorporating into deliberative systems the flexibility to perform trade-offs between decision quality and computational complexity across a range of problem difficulties and time pressures. The ability to dynamically adjust computational effort based on the availability of computational resources has been studied extensively in the AI community since the mid 1980s. These efforts have led to the development of a variety of techniques such as *anytime algorithms* (Dean and Boddy 1988), *imprecise computation* (Liu, Lin, Shih, Yu, Chung, and Zhao 1991), *progressive reasoning* (Mouaddib

and Zilberstein 1997), *design-to-time* (Garvey, Humphrey, and Lesser 1994), and *flexible computation* (Horvitz 1990). The working notes of the 1996 AAAI Fall Symposium on Flexible Computation (Horvitz and Zilbertstein 1996) or the Garvey and Lesser's (1994) review paper offer a good sample of such techniques and applications.

What all these techniques have in common is the realisation that; (i) low deliberation cost sub-optimal or incomplete solutions can result in enhanced system performance, and (ii) the appropriate level of computational expenditure is context dependent. Hence it is useful for the system to dynamically trade the result quality against deliberation cost. A dynamic deliberation system consists of two parts:

1. The base-level computation(s)– which actually make action selection decisions. To be usable for dynamic deliberation the base-level must be able to trade increases in computational resource for increased utility results.

2. The meta-level controller– which attempts to increase the expected utility of computation by custom-tailoring base-level decision making procedures to the specific problem and context. In particular, we wish to make the computational expenditure sensitive to time pressure, the value of likely results, and the relative improvement in outcome quality with more computation.

One can view the dynamic deliberation approach to agent design as an attempt to simplify the intractable bounded-agent design problem by moving to a higher level of abstraction, where we optimise over a small set of complex base-level computations rather than individual architecture-level instructions. This allows us to simplify the bounded agent-design problem by splitting it into two stages, 1) designing (a set of) base-level computations which provide sufficiently high action selection quality for a given resource use, 2) designing a meta-controller which optimises the (sequence of) base-level computations to execute in the current situation. In restricting attention to such designs we have inherently limited the range of agent programs considered and hence abandoned the idea of proving the optimality of our agent design in the interests of simplifying the design problem and improving design modularity. The best we can do as designers is say that this design *seems good in some looser sense*, such as being best with respect to the set(s) of base-level computations considered.

3.2.1 Flexible base-level solvers

The efficiency of dynamic deliberation hinges on the flexibility of the base level to trade increases in computational resource usage for increased utility results. Thus, the first step in designing a dynamic deliberative agent is to design the base-level computation(s). We can then go on to optimise the design of the meta-controller – and then most likely iteratively revise the base-level and meta-level routines so they interact more effectively. The usefulness of a base-level design depends on two main factors, 1) its ability to produce (near) bounded-optimal

solutions over a range of resource allocations, 2) the complexity of its associated meta-control problem which depends on the granularity of the base-level computations.

Broadly speaking, dynamic deliberation systems have tended to use meta-control at two levels of granularity, the *strategic* or macro-level and the *structural*-level. At the macro-level the agent must select one of a small pre-defined set of deliberation strategies (macros) and determine the resources to allocate to it. Most of the meta-level control research in the literature is concerned with macro-level decisions, such as deciding on the amount of time to allocate to an anytime or contract algorithm (Horvitz 1990; Zilberstein and Russell 1996; Garvey, Humphrey, and Lesser 1994). The structural-level concerns finer grained decisions about which individual computational step to perform next. This is the approach studied in this thesis. In some respects all work on selecting the best node to expand next in a search routine is a type of structural-level meta-control. Russell and Wefald (1989) and Baum and Smith (1997) use decision theoretic techniques to develop meta-control algorithms for structural-level computations. The later chapters of the thesis discuss Russell and Wefald's and Baum and Smith's work in more detail.

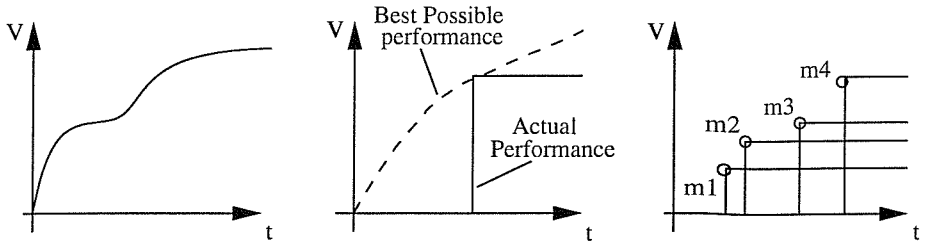
Irrespective of their granularity the base-level computations can be classified as one of three types depending on how they provide variable solution quality with variable resource usage, viz,

1. Anytime Algorithms / Flexible Computations – which can be interrupted at any time to provide a solution whose quality improves monotonically with increasing resource allocation.
2. Contract Algorithms (Russell and Zilberstein 1991) – which require the resource allocation to be determined before their activation and may not produce a solution until the end of the contracted allocation.
3. Multiple Methods – here the base-level contains many possible solution techniques based upon different approximations with varying resource usage and solution qualities.

We can abstractly represent these different models in terms of performance profiles, Figure 3.1 – which represent of the performance of a base-level component in terms of its inputs, including resource allocation, and the resulting solution value (quality). (Performance profiles are discussed in detail in Section 3.3.11.1.)

3.2.2 Anytime algorithms

Anytime algorithms are by far the most popular basis for dynamic deliberation in AI because they offer the most scheduling flexibility—significantly simplifying the meta-control problem. For example, given a single anytime algorithm with an obvious external deadline, e.g. the end of the previously executing action, the optimal meta-control policy is to simply run the algorithm until the deadline, irrespective of the algorithms performance or the likely deadline times. In



(a) Anytime Algorithm (b) Contract Algorithm (c) Multiple Methods

FIGURE 3.1: The Base-Level Solvers Performance Profiles – (a) Anytime algorithms are smooth lines. (b) Contract algorithms: The dashed line represents the smoothly varying upper bound on the algorithms performance, and the solid line the performance of a particular execution, which is a step function. (c) Multiple methods: Possible performances are a set of step functions, each one representing the actual performance of one of the available methods.

general however, as will be shown in Section 3.3.11.2, optimally scheduling anytime algorithms is a hard meta-control problem, especially when making a decision requires the execution of many interacting sub-tasks.

Much work in real-time AI has focused on producing anytime algorithms (and more generally any-resource algorithms (Musliner 1993)) for a variety of problems types. Zilberstein and Russell (1996) show that in many cases developing anytime algorithms does not require radical changes to AI programming techniques – which inherently use approximation techniques to avoid intractability. Example techniques include:

- Various forward search techniques such as, iterative deepening (Korf 1985a), truncated A* or anytime A* (Hansen, Zilberstein, and Danilchenko 1997).
- Asymptotically correct inference algorithms such as, approximate query answering (Vrbsky and Liu 1995) for set queries, bounded conditioning (Horvitz, Suermondt, and Cooper 1989) and state space abstraction (Wellman and Liu 1994) for belief network queries.
- Successive approximation numerical optimisation methods such as; Newton's method, value iteration and policy iteration (Howard 1960).
- Adaptive algorithms such as provably approximately correct (PAC) learning.
- Randomised algorithms such as Monte Carlo estimation, and (when they store the best solution so far) randomised optimisation techniques such as genetic algorithms and simulated annealing.

3.2.3 Contract algorithms

Contract algorithms are similar to anytime algorithms in that result quality monotonically increases with resource allocation (as shown in Figure 3.1(b)) but they are non-interruptible

in that they must be told in advance the amount of resource they will receive, and may only produce a result after this much resource has been used. Contract algorithms are more restrictive and harder to manage from a meta-control point of view as both solution quality or resource availability may be uncertain but a prior commitment to resource usage must still be made. There are many reasons why an algorithm may be non-interruptible. One general case is for meta-managed algorithms which given a resource bound manage their own internal operation to maximise performance, resulting in an overall system with a contract flavour. For example, managing the choice of map resolution in a route finding system or the search depth in a depth-limited search results in a contract algorithm as improving the solution once one resolution or depth has finished requires that the process run to completion at a higher resolution/deeper depth.

The contract model is less restrictive than the anytime model – any anytime algorithm can be managed as a contract algorithm but not vice-versa. However, Zilberstein, Charpillet, and Chassaing (2003) show that we can turn any contract algorithm into an anytime one at the expense of some wasted effort by saving the best result so far and consecutively running the algorithm with increasing resource limits. Indeed, this is a common way to develop interruptible anytime algorithms from non-interruptible algorithms, e.g. the iterative deepening searches used in game systems and Korf's IDA* (Korf 1985a). Zilberstein, Charpillet, and Chassaing (2003) show that in the case of an unknown deadline doubling the resource allocation for consecutive runs results in the minimal overhead. Specifically, the anytime version requires at most 4 times the resource to obtain the same result quality as a single run of the original contract algorithm with the optimal resource allocation.

Related to the ideas of contract algorithms and anytime algorithms are imprecise computations (Liu, Lin, Shih, Yu, Chung, and Zhao 1991). Imprecise computations consist of an uninterruptible (contract style) *mandatory* part and an interruptible (anytime style) *optional* part which increases the output quality with more resource. There has been much work (Liu, Lin, Shih, Yu, Chung, and Zhao 1991; Chung, Liu, and Lin 1990) on finding optimal polynomial time meta-control systems for specific types of problem in this area, for example where all optional or mandatory parts have the same resource requirements.

3.2.4 Multiple methods

A final alternative to developing anytime or contract algorithms is to develop more than one solution to the base level problem, where each solution is designed to have different resource usage and solution quality. As multiple methods make no assumptions about the interruptibility of the solver they are obviously the most general model, and hence the hardest meta-control problem. Indeed, both contract and anytime algorithms can be thought of as multiple methods where a set of discrete resource allocations and output performances define the set of possible base-level methods. In "design-to-time" scheduling Garvey, Humphrey, and Lesser (1994) present a system for meta-control of complex base level solvers which consist of interacting

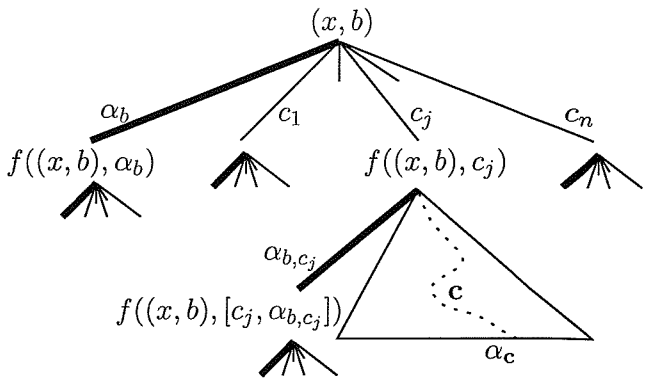


FIGURE 3.2: The meta-level decision problem is to make a sequence of choices based upon the agent's current beliefs b about whether to execute the current default action α_b (i.e. the one the base-level solver believes is best) or one of the possible computations $\{c_1, \ldots, c_n\}$. After executing a computation the meta-level may choose to execute a further sequence of computations $\mathbf{c}$ before the new default action $\alpha_{\mathbf{c}}$. (Note, for brevity $f((x,b),\mathbf{T}) \rightarrow (x',b')$ denotes the combined effect on (x,b) of the action sequence $\mathbf{T}$.)

sub-components for which multiple-methods are available. An alternative, simpler, approach to multiple methods is "progressive processing" (Mouaddib and Zilberstein 1997). Here the meta-control problem is made more tractable by requiring the sub-components to form a hierarchy with all but one of the components in each layer being optional, i.e. having an alternative null implementation.

3.3 Dynamic meta-control

Given a flexible base-level solver, the meta-control problem is to decide how to best manage the base-level so as to maximise the agent's performance. This is the core problem which must be solved in any effective dynamic deliberation system and much of the work on real-time AI has focused upon it. This section reviews the issues associated with meta-control of deliberation and develops the basic tools and methods we can use to construct effective meta-control systems.

The meta-control problem is a control problem just like the on-line planning problem, so we can use normative decision theoretic methods to analyse and solve it. To do so we must first develop a model of the problem the meta-level controller must solve, e.g. which computation to perform next, how long to deliberate for and when to act, and how these decisions influence the agent's ability to perform its task. We can then use this model to identify an (approximately) optimal meta-control policy which can be implemented in the agent.

3.3.1 Modelling meta-control

Simply put, the meta-control problem is one of making a sequence of choices between base-level computations and execution of the base-level's current default action in such a way that

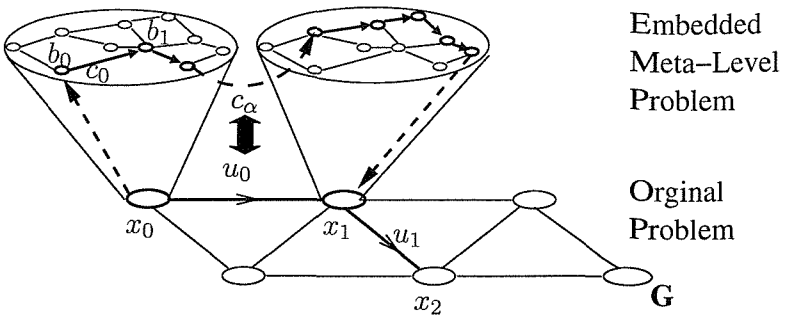


FIGURE 3.3: The meta-level planning problem is embedded within the original problem such that at each new state the agent has to find a trajectory from its current beliefs through belief space which ends in it executing the computation c_α which generates the "right" external action u_0. This belief space and the agent's meta-control policy are fixed for all base-level states, though the initial belief state may vary depending on the observations the agent receives.

we maximise the expected utility of the trajectory the agent traverses. Figure 3.2 (adapted from (Russell and Wefald 1989, p 63)) illustrates this problem. We can model this problem by simply extending the bounded agent design problem of Section 2.3 Def. 2.8 to include:

1. The set of possible base-level computations $\mathcal{L}'$, which consists of the normal base-level computations, $\{c_1, \ldots, c_n\}$, and the special computation, c_α, which causes the agent to execute its current default action, $\alpha \in \mathcal{U}$,

2. The base-level solver itself, represented as a meta-architecture, M', which executes base-level computations, $c \in \mathcal{L}'$, given the agent's current information, (y, b), to generate external actions and update the agent's internal state, $M'(y, b; c) \rightarrow (u, b')$.

3. A meta-level policy, π, which selects which base level computation to execute based upon the agent's currently available information, i.e. $\pi(y, b) \rightarrow c$.

Thus, the meta-level design problem is to design a meta-level policy π which selects base-level computations which maximise the agent's overall expected utility. In fact, this is just the bounded agent design problem at a larger level of granularity with the meta-level policy, π, representing the agents program, l, which the designer must optimise. However, the reduced set of possible computations makes this problem simpler than the bounded agent design problem. In order to perform this optimisation it is more useful to think of the meta-policy as the solution to a path planning problem defined over the space of the agent's beliefs with computations as state transitions. As shown in Figure 3.3 the meta-level planning problem is embedded within, and is a sub-routine of, the original on-line planning problem (defined over physical states and actions).

As with the agent design problem itself computing an optimal meta-policy can be expressed in terms of computing the *value* of the different possible meta-policies. A computational actions primary effect is to change the agent's internal state, however it will also have some external side-effects which cause changes in the external state (modelled using the external action the

computation generates) such as consuming time or other resources. Thus, as for any other action, the future value of a computational action is defined as the value of the trajectory the agent follows from the combined belief and physical state, (b', x'), the computation puts it in. That is,

$$V_{c,\pi}(x, b, t; E, R) = R(T([c, \pi]; E, R, x, b, t)), \quad (3.1)$$

$$= r(x, u, t) + V_{\pi}(f_x(x, u, t), b', t{+}1; E, R), \quad (3.2)$$

where $(u, b') = M'(f_y(x), b; c)$ are the agent's external action and internal state after executing the computation c in state (x, b), and $[c, \pi]$ denotes the policy which initially executes the computation (sequence) c and then operates according to the agent's normal program which implements the policy π.

Given this definition the optimal meta-level value function is given in the normal way by the Bellman optimality equation (2.10),

$$V^*(x, b, t; E, R) = \max_{c \in \mathcal{L}'}[V_{c,\pi^*}(x, b, t; E, R)], \quad (3.3)$$

$$= \max_{c \in \mathcal{L}'}[r(x, u, t) + V^*(f_x(x, u, t), b', t{+}1; E, R)], \quad (3.4)$$

from which the optimal meta-control policy can be extracted by greedy value maximisation.

As indicated in Figures 3.2 and 3.3 this computation consists of two sub-problems,

- the original problem of computing and optimising the value of the external actions executed, and
- the additional embedded meta-control problem of computing and optimising the value of the sequence of computations executed, which depends on the value of the actions executed.

Thus, the meta-level design problem is as hard as the original on-line planning problem because it includes it as a sub-problem. Thus, we cannot as some researchers, e.g. Hacking (1967), have suggested assume that the meta-level problem is easier than the original one, but must acknowledge that meta-control may itself require extensive computational resources which will themselves need to be controlled. Thus we must face the problem of requiring meta-meta-control, meta-meta-meta-control etc., and analytic regress.

3.3.2 Meta-meta-control and infinite regress

Decision theory requires that the meta-control system pick the computation for execution which has maximum expected utility. Since computations are deterministic, in the worse case a deliberative meta-control procedure could identify the maximal utility computation by

performing each of the computations itself and computing the utility of the output – leading to the contradictory situation where rationality requires meta-control to execute a base-level computation to decide if it is worth executing. To resolve this contradiction we could use a meta-meta-controller to select which meta-control computations to apply, but the same arguments apply in this case, requiring a meta-meta-meta-controller, etc.. Eventually, this regression must terminate or it will cost more than executing computations without meta-control.

This analysis allows us to draw the following important conclusions,

- Meta-control *cannot* have complete access to the internal state, b, or exact models of the base-level computations it is controlling but must use simplified abstract models.
- Meta-control *must* be advantageous such that the combined meta-controller and base-level solver require less computation than the base-level solver without meta-control.

Notice, that as optimal meta-control only requires accurate information about possible computation (sequences) *values*, using abstract base-level models does not mean meta-control is necessarily sub-optimal. Indeed, as discussed above, one of the main advantages of anytime/contract algorithms is their simple tractable abstraction in terms of their performance profile which allows optimal meta-control in some circumstances.

Thus, one of the main tasks when designing meta-control procedures for computationally limited agents is identifying tractable base-level abstractions which give adequate performance.

3.3.3 Separable computational cost

One of the main sources of complexity in the meta-control problem is that the policy value depends not only on the external actions executed but also on the sequence of computations executed to choose each action, i.e. $V_{\mathbf{c1},\alpha,\pi} \neq V_{\mathbf{c2},\alpha,\pi}$. This model allows us to capture all problem constraints and trade-offs between computation and action, at the expense of a significant increase in problem complexity as we must optimise directly over all possible computations and actions.

In many cases this complexity is unnecessary as most computations have only a few *external* effects, such as consumption of time or power, upon which the reward function can depend. Hence, one way to simplify the meta-control problem is to assume that the immediate reward of a sequence of computations followed by an action execution, i.e. $R(T([\mathbf{c}, u]; x, b, t))$, can be decomposed into the addition of the *intrinsic* value of the action, r^i, and the *computational cost* of the computations, C. That is,

$$R(T([\mathbf{c}, u]; x, b, t)) = r^i(x, u) - C(x, u, t, |\mathbf{c}|). \tag{3.5}$$

The intrinsic value of an action is immediate reward from executing the action in the state at the start of the sequence. The cost of a computational sequence is the reduction in the actions

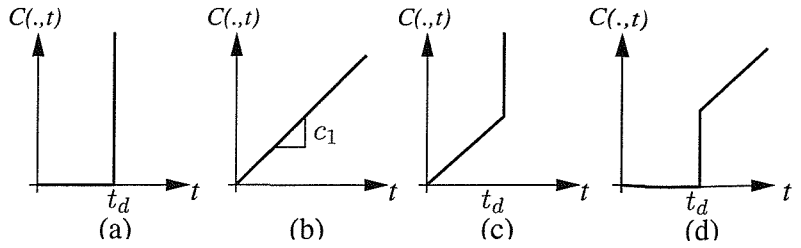


FIGURE 3.4: This diagram (adapted from (Horvitz 1990, p 39)) shows how a separable time cost function can be used to represent many of the prototypical types of resource limitations we would wish to model including, (a) deadline where results after t_d are useless so the cost goes to infinity, (b) urgency with each computational step costing c_1, (c) urgent-deadline, and (d) delayed urgency where computation up to the deadline is free.

immediate reward due to executing the computational sequence before the action. Note that in general, the computational cost can depend on any external effect of the computations, but for simplicity we have assumed it depends only on the total execution time, denoted by $|\mathbf{c}|$.

Whilst not totally general, as discussed in Horvitz (1988) and shown in Figure 3.4, this decomposition captures many of the types of resource dependence we are interested in.

Applying this decomposition to the value computation, (3.2), we obtain,

$$V_{\mathbf{c},u,\pi}(x,b,t) = r^i(x,u) + V_\pi(x',b',t') - C(x,u,t,|\mathbf{c}|). \tag{3.6}$$

where (x',b',t') are the current state, agent beliefs and time after $[\mathbf{c},u]$. Applying this decomposition recursively to $V_\pi(x,b',t')$ allows us to split it into two largely independent parts,

$$V_\pi(x,b,t) = \sum_{j=0..\infty} \left[r^i(x_j,u_j) - C(x_j,u_j,t_j,|\mathbf{c}_j|)\right] \tag{3.7}$$

$$= V^i_\pi(x,b,t) - C_\pi(x,b,t) \tag{3.8}$$

where V^i_π, C_π are the intrinsic value and computational cost of the policy, and $\mathbf{c}_j, u_j, x_{j+1}, t_{j+1}$ are respectively the j'th; computational sequence, action executed at the end of this sequence, this actions resulting state, and arrival time.

A separable time cost allows us to separate the direct external effects of a computational sequence, which are captured in the computational cost, from its internal effects, which are captured through its effects on action choice. This is a significant simplification allowing the value calculation to be split into two independent parts,

(1) calculating the policies intrinsic value, $V^i_\pi(x,b,t)$, which depends only on the external actions the agent executes and the states they put it in, and

(2) calculating the policies computational cost, $C_\pi(x,b,t)$, which depends only on the time spent on computation in each state.

Unfortunately, time cost alone does not provide the degree of simplification we require as the action executed still depends on the agent's exact internal state b. Thus to complete this simplification we need an abstract model of the agent's internal state and how sequences of computations alter it.

3.3.4 The meta-control Markov decision problem (MDP)

We require a model of the base-levels operation which is much simpler to analyse and solve. One way to perform this simplification is to use a *state-aggregation* abstraction (Holete, Preez, Zimmer, and MacDonald 1996) to approximate the detailed model by mapping sets of original states with the same important *features* into one abstract state. In this abstract space the discarded information means we cannot be sure exactly what state an action is being executed in so its effects will be *uncertain*. Thus, we must use a stochastic model for the actions effects. Note, this (additional) uncertainty is an artifact of the state-aggregation abstraction and not inherent to the problem.

Given an abstract model of this form, the meta-level design problem is now to find an optimal meta-policy which specifies which action or computation to perform in each abstract state. We can treat this as a Markov Decision Problem (MDP) (Puterman 1994). In a MDP we must find an optimal state→action (computation) mapping when the successor state after action execution cannot be predicted in advance (using this model) but is known with certainty after the action is executed. In addition, to obey the Markov property the successor state transition must be conditionally independent of the path taken to that state given the current state and action, i.e. the likely successor states depend only on the current state and action. For a more complete introduction to MDPs and their solution methods the interested reader is advised to see Puterman's (1994) book or any introductory text on stochastic dynamic programming, such as (Bertsekas and Shreve 1978), or reinforcement learning (Sutton and Barto 1998).

Note, in general state aggregation does not result in a true MDP. This is because the successor state actually depends on the current base-level state, which can be identified from the trajectory taken to the state using the non-abstracted domain model. Thus, the abstract successor state transition can depend on the trajectory as well as the current state and action, violating the Markov requirement. For the sake of tractability we ignore this subtlety and assume the abstract state transitions are Markov.

In this case the approximate meta-control MDP is given by,

- A feature detector, $f_\phi(b) \rightarrow \phi$, which maps the agent's internal state b to some abstract feature space, Φ. This abstract feature space is the state space for the MDP,
- A set of abstract actions $\{c_\alpha, c_1, \ldots, c_n\}$ corresponding to the set of all possible computations, including the special computation c_α which causes the agent to execute it's default action, $\alpha \in \mathcal{U}$,

- A reward function $r^m(\phi, c)$. A separable time cost is assumed so c_α's reward is the executed action's *intrinsic* reward, $r^m(\phi, c_\alpha) = r^m(\phi, \alpha) = r^i(x, \alpha)$, and all the other purely computational actions' rewards are their negative time costs, $r^m(\phi, c) = -C(x, t, |c|)$. Note, to ensure $r^m(\phi, c)$ is a Markov reward function the computational cost cannot depend on the final physical action executed as in (3.5), so we assume $\forall u, C(x, u, t, |\mathbf{c}|) = C(x, t, |\mathbf{c}|)$.

- A model of the effects of the computations in the abstract feature space represented as a stochastic transition function, $\Pr(\phi'|\phi, c)$, which gives the probability that the result of executing computation c in an internal state with feature vector ϕ is one with feature vector ϕ'.

  This can be computed off-line by sampling a large number of possible belief states, recording their feature values and using Bayes rule to compute,

$$\Pr(\phi'|\phi, c) = \frac{\Pr(\phi', \phi|c)}{\Pr(\phi)} = \frac{\sum_{b \in \mathcal{B}} \Pr(\phi', \phi|c, b) \Pr(b)}{\Pr(\phi)} \tag{3.9}$$

$$= \frac{\sum_{b \in \mathcal{B}} \Pr(\phi'|b, c) \Pr(\phi|b) \Pr(b)}{\Pr(\phi)} \tag{3.10}$$

  where we have used the fact that $\phi = f_\phi(b)$ and $\phi' = f_\phi(M'(b; c))$ hence ϕ' and ϕ are deterministically defined by and conditionally independent given b, c.

The separability of the objective function means the expected value of any abstract belief state under a meta-policy π which begins with computation c is given by,

$$V_\pi(\phi) = r^m(\phi, c) + \sum_{\phi'} \Pr(\phi'|\phi, c) V_\pi(\phi'). \tag{3.11}$$

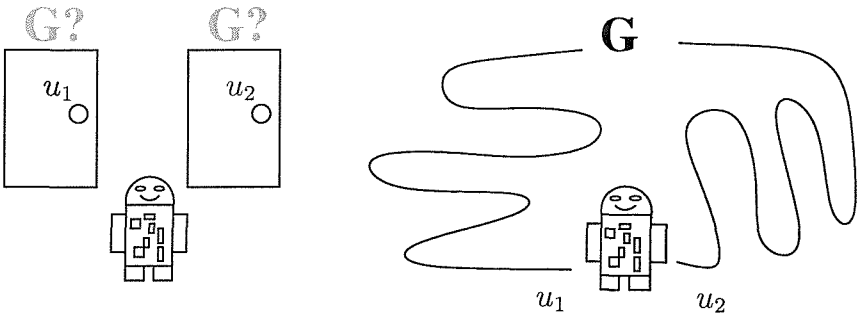
This is the stochastic version of the Bellman equation. In the particular case of the meta-level MDP we distinguish between the special action execution computations which have an intrinsic value, and normal computations to give,

$$V_\pi(\phi) = \sum_{\phi'} \Pr(\phi'|\phi, c) V_\pi(\phi') + \begin{cases} r^i(x, \alpha_\phi) & \text{if } c{=}c_\alpha \\ -C(x, t, |c|) & \text{otherwise} \end{cases} \tag{3.12}$$

As in the deterministic case the optimal value function, V^*, must satisfy the Bellman optimality equation,

$$V^*(\phi) = \max_c \left[r^m(\phi, c) + \sum_{\phi'} \Pr(\phi'|\phi, c) V^*(\phi') \right]. \tag{3.13}$$

Given a value function satisfying this equation the optimal meta-policy can be extracted in the usual fashion. This recurrence is the basis of many algorithms for solving MDPs. An extensive literature exists on the efficient solution of MDPs based upon prior models using off-line dynamic programming methods (Puterman 1994; Bertsekas and Shreve 1978; Hansen



(a) Value of Information (b) Value of Computation

FIGURE 3.5: These problems adapted from (Tash 1996) present two example cases where more information may be valuable. In (a) Robbie is unsure whether the goal lies behind door u_1 or u_2 so information about this is valuable, in (b) Robbie knows where the goal is but is unsure whether path u_1 or u_2 is shorter so spending resources computing the paths lengths may be valuable.

and Zilberstein 2001b) and on-line (Sutton and Barto 1998; Barto, Baradtke, and Singh 1995) methods, or even simultaneously learning the MDP model whilst solving it using reinforcement learning techniques such as Q-learning (Peng and Williams 1995). The MDP model is common whenever we abstract a deterministic sequential decision problem for tractability, as we will see later many abstract meta-control formulations can be modelled in this way.

3.3.5 The Value-estimate abstraction

Defining the right feature detector is critical to the effectiveness of the meta-control MDP as the meta-control decisions are based solely upon the current abstract state. Thus, the feature detector should capture the features of the agent's current beliefs which are important for making good meta-level decisions – that is the features which are important for predicting the quality of the agent's decisions and how the alternative base-level computations improve these decisions. However, the feature detector should also provide a sufficient abstraction to make the meta-level design problem tractable and the meta-policy implementable. Thus, the base-level abstraction must trade-off meta-level decision quality against meta-level policy tractability.

The most important features of the agent's internal state are those which affect which actions it selects for execution. If the agent is rational, we can treat these selections as being based upon the agent's current *subjective* assessments of the expected utility of the action. Notice, as was the case when defining a rational agent in Section 2.3 we do not require that the base-level explicitly represent such value estimates, only that it make decisions as if it did, so we can deduce them for the meta-level design. Therefore, one of the most important features of a rational agent's internal state are the *subjective solution value assessments its decisions imply.*

If computation improves the agent's action choices by updating its internal state then these subjective value assessments must be incorrect in some way. Thus, we can then view the agent's internal state as encoding the agent's current partial state of information about the

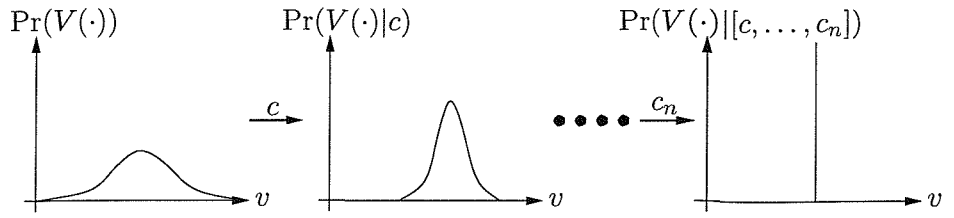


FIGURE 3.6: The effect of computation is to converge the agent's estimate of the true value of a possible solution hence reducing the uncertainty of the estimate, until (given sufficient free computation) its estimate is certain.

value of possible solutions to its problem, i.e. which are the best actions to perform in this environment. A computation is then essentially a type of observation, where the agent obtains additional information by manipulating its prior information (provided by its designer or computed previously), and making observations. Figure 3.5 illustrates this view where either direct observation of the world in problem (a) or computation in problem (b) are beneficial because they both have the effect of improving Robbie's value estimates and hence his action choices.

Now, as shown in Figure 3.6, assuming the computations are correct, the effect of each individual computation is to converge the agent's value estimates towards some unknown *final* (or true) value – which is the value the agent would calculate given sufficient time to compute an exact valuation. The effect of a computation therefore depends not only on the agent's current value estimate, $\hat{\mathbf{V}}$, but also how close this estimate is to the unknown true value, i.e. if a value estimate is correct applying computation to refine this estimate will have no effect. Of course, the agent does not know the true value towards which its computations are converging—if it did it wouldn't need to perform the computations.

However, the agent may have some knowledge of how close it is to the correct value, say in the form of bounds on the true value. In the most general terms we can represent *all* the agent's current beliefs about the true value in terms of a joint probability distribution over final values, $\Pr(\mathbf{V}|b) = \Pr(\hat{\mathbf{V}}|b, c_\infty)$. Given this distribution, a rational agent will select the *Bayes action*. This is the action which maximises the expected value w.r.t. this distribution, i.e. $\alpha_b = \text{argmax}_u \hat{\mathbf{V}}(u|b)$ where $\hat{\mathbf{V}} = \mathrm{E}\{\mathbf{V}|b\}$ is the agent's current value estimate.

Thus, we see that the joint probability distribution over final values represents a good abstract feature set for the meta-control problem which captures important features of both the agent's eventual action selections and the effects of computations. An abstract base-level model using value distributions consists of:

- Π the set of possible solutions.
- $\hat{V}_\pi$ a value estimate for each possible solution, $\pi \in \Pi$, $\hat{\mathbf{V}}$ is used to denote the vector containing a value estimate for each solution.
- A feature detector, $f_\phi(b) \rightarrow \phi$, which maps the agent's internal state b to some abstract feature space, Φ,

- $p_{\mathbf{V}} = \Pr(\mathbf{V}|\phi) = \Pr(\hat{\mathbf{V}}|f_\phi(b), c_\infty)$, a distribution over future value vectors parameterised by the feature set.

  $\Pr(\mathbf{V}|\phi)$ is essentially a function mapping from abstract features to value distributions. It can be learnt off-line by in a similar way to $\Pr(\phi'|\phi, c)$ by sampling a large number of belief states, recording their feature values and *true* solution values, and then using Bayes rule to compute, $\Pr(\mathbf{V}|\phi) = \Pr(\mathbf{V}, \phi)/\Pr(\phi)$. Note, computing the true solution values for each training example can represent a significant overhead for this approach.

- An abstract model of the effects of the computations represented as a stochastic transition function, $\Pr(p'_{\mathbf{V}}|p_{\mathbf{V}}, c)$. This is a second order distribution over distributions giving the probability that the result of updating the distribution of value estimates, $p_{\mathbf{V}}$, with computation, c, is the new distribution, $p'_{\mathbf{V}}$. As before this can be computed off-line using (3.10).

3.3.6 The subjective value equivalence assumption

So far, we have treated the value distribution as just an alternative abstract base-level model with a readily interpretable semantics and overly complex model learning requirement. However, the main power of the value distribution abstraction is that, when combined with the separable time cost assumption, it allows us completely separate meta-policy construction from base-level solution quality computation. This is achieved by making the following assumption.

Assumption 3.1 (Subjective value equivalence). *The agent's subjective value distribution is assumed to be a consistent, reliable, and accurate representation of the actual uncertainty in the value the agent will realise. Hence the agent can treat its subjective belief that it will obtain a certain expected value as equivalent to actually obtaining that value.*

Thus, we can use these subjective estimates directly for the intrinsic value of an action – *without having to compute the actual value of the agent's trajectory after the action.* Following Tash (1996) we call an agent which makes its decisions on the basis of the subjective value equivalence assumption *locally rational* as it considers maximising locally derived subjective value estimates as an end in its own right, without explicitly considering the longer term implications of its decisions. Clearly, the performance of such an agent depends critically on the accuracy of its subjective value distributions as predictors of the actual future value the agent will realise. This in turn depends on the accuracy of the agent's initial value distributions and the correctness of the computational mechanisms manipulating them.

This brings to light a subtlety which we have glossed over so far: What is the "true value" towards which the agents value estimates are converging and upon which the agent bases action selections? By definition the value of a situation is the "total objective value accumulated by the future trajectory induced by the policy" (Section 2.2.3 Def. 2.3). This puts us in something of a "chicken and egg" situation were we need to know the policy to compute the value function, but

we need to know the value function to define our locally rational policy. There are two possible solutions to this problem, either we compute our value function using one policy which we hope is close to the one the agent will implement and accept that this may lead to poor action choices, or we use reinforcement learning style techniques (Sutton and Barto 1998) to converge the value function and agent policy on-line towards a realistic representation of the agent's performance. For now we put aside this issue, leaving more detailed discussion until Chapter 4, and just assume that an appropriate value function is available.

3.3.7 The value-estimate MDP

The subjective value equivalence assumption allows us to treat the meta-level design problem independently of the original planning problem by allowing us to treat the value of a physical action as equivalent to its current expected subjective value. That is,

$$V_u(x,b,t) = \hat{\mathbf{V}}(u|b) = \mathrm{E}\{\mathbf{V}(u)|f_\phi(b)\} = \sum_{\mathbf{V}} \Pr(\mathbf{V}|\phi)\mathbf{V}(u) = \langle \mathbf{V}(u)|f_\phi(b)\rangle. \quad (3.14)$$

Note, in the remainder of this section it will be assumed that all values are computed assuming the default policy π, so this will not be explicitly stated in the subscript.

As the agent is locally rational, if it was forced to act now its default action would be to execute the action which it believes has maximum expected value, i.e. the Bayes action. Hence, the value of c_α, the computation which corresponds to executing the agent's current default action, is given by,

$$V_{c_\alpha}(x,b,t) = \max_u \hat{\mathbf{V}}(u|b) = \max_u \langle \mathbf{V}(u)|f_\phi(b)\rangle \quad (3.15)$$

Substituting this into the meta-policy value recursion (3.12) we get,

$$V_\pi(\phi) = V_{c,\pi}(\phi) = \begin{cases} \max_u \langle \mathbf{V}(u)|f_\phi(b)\rangle & \text{if } c{=}c_\alpha \\ \sum_{\phi'} \Pr(\phi'|\phi,c)V_\pi(\phi') - C(\phi,|c|) & \text{otherwise} \end{cases}, \quad (3.16)$$

where we have made the additional assumption that the abstract state ϕ is sufficient to identify the current computational cost so, $C(x,t,|c|) = C(\phi,|c|)$.

Comparing (3.12) and (3.16) we can see the true power of the value estimate abstraction and the subjective value assumption is that they allow us to terminate the value recursion by treating the action execution computation, c_α, as a deterministic transition to a terminal state with known value. Thus, we can design an approximately optimal meta-policy by solving the meta-level MDP directly.

3.3.8 Special Properties of the value-estimate MDP

It may be that this MDP is still too hard to solve completely, if for example the feature space Φ is too large. To reduce the complexity further we could use a smaller feature space or use one of the

specialised methods developed for the approximate solution of MDPs (Sutton 1990; Williams and Baird 1993; Baird III 1995; Gordon 1995; Poupart, Boutilier, Patrascu, and Schuurmans 2002). However, the value-estimate MDP has some special properties which we may exploit to develop specialised, efficient approximate solution techniques.

Consistency of value distributions. By assumption the intrinsic value of an action is independent of the agent's internal state. Thus, if a computation c changes the agent's internal state from ϕ to some unknown ϕ' then, by the standard axioms of probability, we can marginalise over ϕ' to obtain,

$$\Pr(\mathbf{V}|\phi) = \sum_{\phi'} \Pr(\mathbf{V}|\phi') \Pr(\phi'|\phi, c) = \mathrm{E}\left\{\Pr(\mathbf{V}|\phi')|\phi, c\right\}. \tag{3.17}$$

In words (3.17) says that the agent's prior value distribution should equal its expected posterior value distribution, hence the expected future probability mass at any point is constant. This consistency relation represents an additional rationality requirement which the agent's belief update procedures must meet, otherwise, as discussed in (Skyrms 1990), the agent could act irrationally. For example if a computation does not meet this requirement and increases all the action value estimates by x then the agent would, irrationally, be willing to pay up to x for this computation—even though it leaves the action choice unaltered and provides no useful information.

One important consequence of (3.17) is that *for any consistent belief update* the agent's prior expected value should equal its expected posterior expected value. That is,

$$\mathrm{E}\{\mathbf{V}|\phi\} = \sum_{\mathbf{V}} \mathbf{V} \sum_{\phi'} \Pr(\mathbf{V}|\phi') \Pr(\phi'|\phi, \mathbf{c}) = \mathrm{E}\left\{\mathrm{E}\left\{\mathbf{V}|\phi'\right\}|\phi, \mathbf{c}\right\}. \tag{3.18}$$

Monotonically increasing Bayes action value. By definition $\hat{V}_{c_\alpha} = \max_u \mathrm{E}\{\mathbf{V}(u)|\phi\}$ gives the expected value of executing the Bayes action now w.r.t. the agent's beliefs ϕ. Ignoring the cost of computation, the expectation of $\hat{V}_{c_\alpha}$ over future belief states, $\hat{V}^i_{c,c_\alpha} = \mathrm{E}\left\{\hat{V}^i_{c_\alpha}|\phi, c\right\} = \mathrm{E}\{\max_u \langle \mathbf{V}(u)|\phi'\rangle|\phi, c\}$, gives the expected posterior expected value of the Bayes action immediately after the computation, c. Now, by (3.18) $\mathrm{E}\{\mathbf{V}|\phi\} = \mathrm{E}\{\langle \mathbf{V}|\phi'\rangle|\phi, c\}$ hence if $\alpha = \mathrm{argmax}_u \langle \mathbf{V}(u)|\phi\rangle$ then $\hat{V}_{c_\alpha} = \hat{V}^i_{c,\alpha}$. This result allows us to compare $\hat{V}_{c_\alpha}$ and $\hat{V}^i_{c,c_\alpha}$ to obtain,

$$\hat{V}_{c_\alpha} = \hat{V}^i_{c,\alpha} = \max_u \mathrm{E}\left\{\langle \mathbf{V}(u)|\phi'\rangle|\phi, c\right\} \quad \leq \quad \mathrm{E}\left\{\max_u \langle \mathbf{V}(u)|\phi'\rangle|\phi, c\right\} = \hat{V}^i_{c,c_\alpha} \tag{3.19}$$

This follows from Jensen's inequality ($f(\mathrm{E}\{X\}) \leq \mathrm{E}\{f(X)\}$ for convex $f(\cdot)$) and the convexity of maximum. Notice, that equality holds if we can reverse the order of the max and outer expectation on the RHS, which can only occur if the maximising action is independent of $\langle \mathbf{V}(u)|\phi'\rangle$, i.e. is constant. Thus computation is worthless iff there exists a constant action k such that $\mathrm{argmax}_u \langle \mathbf{V}(u)|\phi'\rangle = k$ forall ϕ'. This derivation is essentially as given in (Skyrms

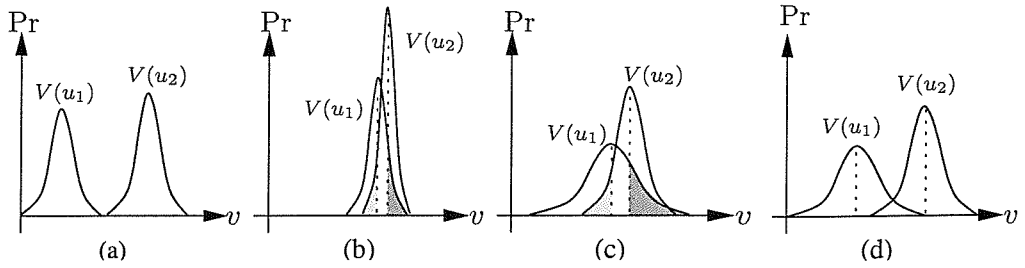


FIGURE 3.7: This diagram (adapted from (Russell and Wefald 1989, pp 76)) shows how the value of information depends on the agent's value uncertainty. The dark (and light) shaded regions indicate the areas where improved information about u_1's (resp. u_2's) value is useful. In (a) information is worthless as the agent's uncertainty doesn't affect its action choice. In (b) the possible change in the value of the agent's chosen action is small (as indicated by the narrowness of the shaded regions), so information has little value. In (c) however a large area exists where the action choice could change to realise significant additional value, so information is more valuable. Part (d) demonstrates a case where information about u_1 or u_2 alone has no value as it cannot change the action choice, but combined information about both may be valuable.

1990, p 92-100) which is a generalisation of Good's (1983) "dynamic probability" to the case where all that is known is that our value estimates will change.

To summarise, the preceding analysis has shown that the value-estimate MDP has the following important special properties,

P1 Performing additional *consistent* belief update computation *for free* cannot decrease the expected value of the Bayes action.

P2 An additional free consistent belief update computation can only *increase* the expected value of Bayes action if it could change the *final* Bayes action.

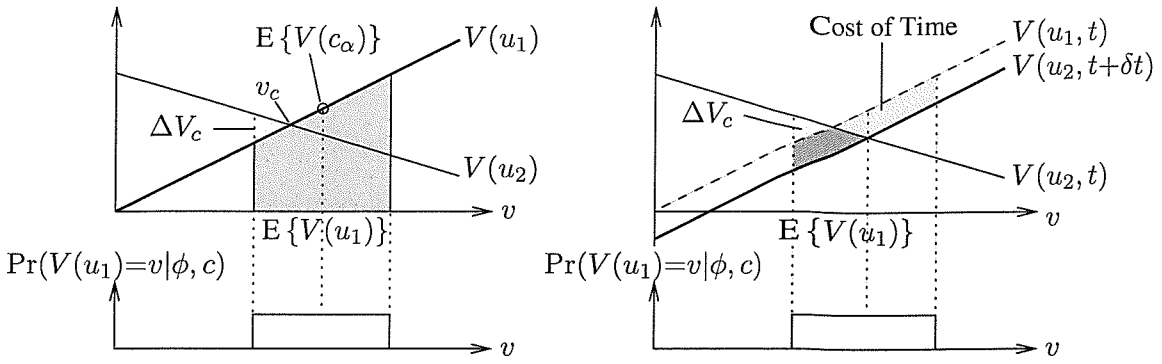
P3 Thus, only the final *expected value* of physical actions matters when assessing the value of a policy which finishes by executing the Bayes action.

P4 Only the changes in physical actions expected value that cause the Bayes action to change matter when assessing the marginal benefit of a "compute-then-act" policy compared to an "act-now" policy.

3.3.9 The value of computation/information

The difference between the value of the Bayes action before and after the computation is the marginal value of the computation, which is more generally called the value of information.

That the value of computation must always be positive is easiest to visualise if we consider the simplest case where an agent must choose between two actions, u_1 and u_2, such as the doors or paths faced by Robbie in Figure 3.5. Figure 3.7 shows four prototypical cases of value estimate uncertainty the agent may face. Part (a) demonstrates the case where the agent can be very



(a) The value of free computation

(b) The value of computation with time cost

FIGURE 3.8: These diagrams demonstrate the value of perfect information when there are only two actions u_1 and u_2 whose true values are assumed to be related by $V(u_2) = f(V(u_1))$ so they can be drawn on a 2D graph. Figure (a) shows the case when the computation, c, can completely resolve the uncertainty in $V(u_1)$ (and hence $V(u_2)$) without cost, figure (b) shows the same case when the time taken to perform the computation reduces $V(u_1)$ by a constant amount.

uncertain, but reducing this uncertainty has no value as it cannot change the agent's decisions. Parts (b) and (c) show that the marginal value of information about u_1 or u_2 also depends on the amount of improvement changing the Bayes action will realise, information has a higher marginal value in (c) as it results in a larger potential value improvement (as shown by the width of the shaded areas). Finally (d) demonstrates the case where we need to reduce the uncertainty of both u_1 and u_2 in order to realise any marginal value.

An alternative way of thinking about the same issues is illustrated in Figure 3.8. Here we assume the intrinsic value of u_2 can be represented as a function of the value of u_1, i.e. $V(u_2) = f(V(u_1))$, so we can represent the uncertainty in the agent's value estimates on a single 2D graph. This situation can occur if both action values depend on a single unknown parameter (such as the door behind which the goal lies). In this case, u_1 is the prior Bayes action as it has greater expected value w.r.t. the agent's current value uncertainty as indicated by the dark shaded region. Now, if the agent knew $V(u_1)$ was less than the value at the point where the two lines cross, v_c, then it could improve its action value by choosing action u_2. Thus as the total probability at $V(u_1) = x$ must remain constant, see (3.18), the probability of this occurring is given by the current value distribution for $V(u_1)$. Such a change can only improve the value of the Bayes action, with the extent of improvement given by the light shaded area, labelled ΔV_c.

3.3.10 Simplifying the value-estimate MDP (i) Meta-greedy policies

As discussed in Section 2.2.3 an optimal policy can be constructed for any sequential decision problem using only local information by greedily selecting the action with maximal optimal future value, i.e. $\alpha^* = \text{argmax}_u[r(u) + V^*(u)]$. As maximisation is unaffected by addition

(1) Identify the Bayes action w.r.t. our current beliefs, $\alpha = \text{argmax}_u V^i(u)$

(2) Identify the computation c^* which has maximal expected value of computation, $\Delta V(c) = V_c^i - C(|c|) - V_\alpha$,

(3) If $\Delta V(c^*) \geq 0$ then execute c^* otherwise execute u^*.

FIGURE 3.9: The META-GREEDY algorithm

of a constant, like $-\hat{V}^i_{c_\alpha}$ say, the same technique can be used with marginal values. Thus, the META-GREEDY algorithm presented in Figure 3.9 is optimal if $\Delta V(c) = \Delta V^*(c)$ and bounded-optimal with respect to the base-level if the calculation of $\Delta V(c)$ is cost free. An alternative way to analyse META-GREEDY is to think of $\Delta V(c) = V(c) - V^i_\alpha$ is as a measure of the sensitivity of the Bayes action value to possible value distribution refinements, so META-GREEDY is a hill-climbing algorithm which performs the most sensitive computations first. This meta-control algorithm, originally proposed by Good (1968), is found in various forms throughout the meta-control/bounded optimality literature, see for example (Horvitz 1988; Russell and Wefald 1989).

Unfortunately, calculating the optimal $\Delta V^*(c)$ requires that the agent solve the value-estimate MDP—which is exactly what we are trying to avoid. Thus, one approach to simplifying the value-estimate MDP is to use the special properties discussed above to develop efficiently computable approximations which can be used in place of $\Delta V^*(c)$ in the META-GREEDY algorithm. The hope is that if these approximations are sufficiently close to the true $\Delta V^*(c)$ then META-GREEDY will be optimal w.r.t. the approximations and likely near the true optimum.

Before we continue further we need to clarify some idiosyncrasies in terminology. As originally defined by Howard (1966) and in conventional usage $\Delta\hat{V}_{\mathbf{c}}$ and $\Delta\hat{V}_{c_\infty}$ are referred to as the "expected value of computation" (EVC) (or more generally information if $\mathbf{c}$ represents a general observation) and the "value of perfect information" (VPI) respectively, see for example (Good 1983; Russell and Wefald 1989; Horvitz 1990). This terminology may cause some confusion as these quantities are actually *marginal* values between the meta-policies $[c_\alpha]$ and respectively $[c, \pi]$ and $[c_\infty, \pi]$, where π is the agent's normal policy. For tradition, I will maintain this inconsistency and use the term "future value" to indicate when referring to the true future value, i.e. $\hat{V}_c$ or $\hat{V}_{c_\infty}$.

We start the process of developing an approximation for $V^*(c)$ by noting that the monotonicity of $\hat{V}^i_{c_\alpha}$ w.r.t. computational effort (property **P1**) allows us to define the following upper and lower bounds on the optimal value function,

$$\hat{V}^i_{c_\alpha}(\phi) \;\leq\; V^*(\phi) \;\leq\; \hat{V}^i_{c_\infty, c_\alpha}(\phi), \tag{3.20}$$

from which as $V^*(\phi, c) = V_{c,\pi^*}(\phi) = \mathrm{E}\{V^*(\phi')|c, \phi\} - C(\phi, |c|)$ by (3.12) we obtain,

$$\hat{V}^i_{c,c_\alpha}(\phi) - C(\phi, |c|) \;\leq\; V^*(\phi, c) \;\leq\; \hat{V}^i_{c_\infty, c_\alpha}(\phi) - C(\phi, |c|). \tag{3.21}$$

Russell and Wefald (1989) call the lower bound, $\hat{V}^i_{c,c_\alpha}(\phi) - C(\phi, |c|)$, a *myopic* approximation as it short-sightedly only considers the *immediate* effect of the computation on the Bayes action value, ignoring the possible longer term benefits when its effects are combined with those of later computations. Continuing this analogy I call the upper bound a *hyperopic* approximation as it considers all possible effects of the computation after an infeasibility large amount of further computation.

The upper and lower bound computations can be simplified using property **P3** to write $\hat{V}^i_{c,c_\alpha}(\phi)$ directly in terms of the distribution of posterior expected values $\hat{\mathbf{V}}' = \langle \mathbf{V}|\phi' \rangle$,

$$\hat{V}^i_{c,c_\alpha}(\phi) = \mathrm{E}\left\{V^i_{c,c_\alpha}(\phi)\right\} = \sum_{\hat{\mathbf{V}}'} \Pr(\hat{\mathbf{V}}'|\phi, c) \max_u \hat{\mathbf{V}}'(u). \tag{3.22}$$

Thus, we can compute simple bounds for the optimal value of computation using only knowledge of the resulting distribution of the agent's expected future value estimates. The computation of the upper bound, $\hat{V}^i_{c_\infty,c_\alpha}(\phi)$, can be simplified further by noting that by definition $\mathrm{p}_{\mathbf{V}}(\mathbf{V}) = \Pr(\hat{\mathbf{V}}'{=}\mathbf{V}|\phi, c_\infty)$, hence,

$$\begin{aligned} \hat{V}^i_{c_\infty,c_\alpha}(\phi) = \mathrm{E}\left\{V^i_{c_\infty,c_\alpha}(\phi)\right\} &= \sum_{\hat{\mathbf{V}}'} \Pr(\hat{\mathbf{V}}'|\phi, c_\infty) \max_u \hat{\mathbf{V}}'(u) && (3.23) \\ &= \sum_{\hat{\mathbf{V}}} \mathrm{p}_{\mathbf{V}}(\hat{\mathbf{V}}) \max_u \hat{\mathbf{V}}(u) && (3.24) \end{aligned}$$

The computation of the bounds can be simplified further if we work in terms of the *marginal* intrinsic value of computing then acting w.r.t. acting now, $\Delta\hat{V}^i_c = \hat{V}^i_c - \hat{V}^i_{c_\alpha}$. The advantage of the marginal values is that property **P4** tells us that if the posterior Bayes action equals the prior Bayes action the marginal change is zero. Hence to compute the marginal value we only need to consider the cases where the Bayes action changes (as shown by the shaded areas in Figure 3.7 and light shaded area in Figure 3.8). Formally,

$$\begin{aligned} \Delta\hat{V}^i_{c,c_\alpha}(\phi) &= \hat{V}^i_{c,c_\alpha}(\phi) - \hat{V}^i_{c_\alpha}(\phi) && (3.25) \\ &= \sum_{\hat{\mathbf{V}}'} \Pr(\hat{\mathbf{V}}'|\phi, c) \max_u \hat{\mathbf{V}}'(u) - \max_u \sum_{\hat{\mathbf{V}}'} \Pr(\hat{\mathbf{V}}'|\phi, c)\hat{\mathbf{V}}'(u) && (3.26) \\ &= \sum_{\{\hat{\mathbf{V}}':\mathbf{V}'(\alpha)<\max_u \mathbf{V}'(u)\}} \Pr(\hat{\mathbf{V}}'|\phi, c) \left[\max_u \hat{\mathbf{V}}'(u) - \hat{\mathbf{V}}'(\alpha)\right] && (3.27) \end{aligned}$$

where, α denotes the Bayes action in original belief state, ϕ. A similar equation holds for $\Delta\hat{V}^i_{c_\infty,c_\alpha}$ with $\Pr(\hat{\mathbf{V}}'|\phi, c)$ replaced by $p_{\mathbf{V}}(\hat{\mathbf{V}})$,

$$\Delta\hat{V}^i_{c_\infty,c_\alpha}(\phi) = \sum_{\{\hat{\mathbf{V}}':\mathbf{V}'(\alpha)<\max_u \mathbf{V}'(u)\}} \mathrm{P}_{\mathbf{V}}(\hat{\mathbf{V}}') \left[\max_u \hat{\mathbf{V}}'(u) - \hat{\mathbf{V}}'(\alpha)\right] \tag{3.28}$$

(3.27) and (3.28) offer potentially large computational saving when the agent is reasonably certain of the best action.

As a final step this calculation may be simplified further if we know the ranges of value estimates where each action is maximal, then we can perform the summation in each region independently with the maximum operation replaced with the value of the fixed maximal value action. To see how this is useful consider again Figure 3.8(a), in this case the light shaded area labelled ΔV represents the marginal value of computation. It is clear from the figure that the Bayes action is unchanged unless $V(u_1) \leq v_c$, when its value is given by $V(u_2)$, thus we can compute the marginal value of computation using,

$$\Delta \hat{V}^i_{c,c_\alpha}(\phi) = \sum_{V(u_1) \leq v_c} \Pr(V(u_1)|\phi, c) \left[V(u_2) - V(u_1)\right]. \tag{3.29}$$

The lower bound on the marginal value, $\Delta V^i_{c,c_\alpha}$, is called the myopic expected value of computation (MEVC) as it ignores the effects of any computations after c and forms the basis of many decision theoretic meta-control systems, including Russell and Wefald (1989), Horvitz (1990). The upper bound, $\Delta V^i_{c_\infty,c_\alpha}$, is what Baum and Smith (1997) call the value of all expansions as it represents the value of getting perfect information about all action values and forms the basis of their Best Play for Imperfect Players (BPIP) (Baum and Smith 1995) meta-control algorithm. We will encounter these value of computations again later in Chapter 5 when we consider the search control problem.

3.3.10.1 Discussion

Equations (3.27) and (3.28) represent two reasonably efficiently computable marginal value of computation approximations which, as shown in (3.21), can be used to approximate $\Delta V^*(c)$ in META-GREEDY (Figure 3.9). The quality of these approximations obviously depends on how closely their inherent assumptions reflect the reality of the situation. $\Delta \hat{V}^i_{c_\infty,c_\alpha}$ will tend to provide a good approximation when the optimal meta-policy performs enough computations to remove almost all the uncertainty in the action values. $\Delta \hat{V}^i_{c,c_\alpha}$ will tend to produce good approximations when the optimal policy performs very few more computations. Note however that the accuracy of this short-sighted approximation can be improved, at some computational cost, by simply increasing the range of c's considered to include sequences of computations.

To actually use these bounds in META-GREEDY we require the following information,

- A function $\Pr(\hat{\mathbf{V}}'|\phi, c)$ describing the effects of a computation in terms of the subsequent distribution of new expected value estimates, and
- A separable time cost function $C(c)$.

To allow efficient bound computation $\Pr(\hat{\mathbf{V}}'|\phi, c)$ should be structured so it is easy to identify regions where the Bayes action changes. The basic method for developing such structured probability estimates is to build a factored model (Kim and Dean 2003) based upon the

observation that in most cases computation's are highly local, depending on and affecting only a small subset of the agent's internal state. These effects will in turn have localised influence on the agent's value distributions and Bayes action choice. For example, in a search framework expanding a node has the local effect of revising that nodes value estimate which in turn may affect the value estimates of other nodes which depend on it. Thus, by exploiting detailed information about the operation of the base-level we can decompose $\Pr(\hat{\mathbf{V}}'|\phi, c)$ in the following way,

(i) define a feature detector f_ϕ which abstracts the internal state b into a multi-dimensional vector of feature's ϕ,

(ii) construct a *factored* stochastic model of the effects of computation on the internal state, $\Pr(\phi'_c|\phi_c, c)$ where ϕ_c is the sub-set of dimensions of ϕ affected by computation c,

(iii) construct a *propagation function* which models the effects of these local changes on the value estimates, $\Pr(\hat{\mathbf{V}}'|\phi'_c, \phi)$. This can be deterministic or stochastic.

In general, learning a stochastic model requires a number of training examples proportional to its size, which is exponential in the dimensionality of its inputs. Thus the factored model also has the advantage of simplifying the learning process allowing us to produce more accurate models for much less training effort. As $\Pr(\hat{\mathbf{V}}'|\phi, c)$ must be evaluated by the meta-level for each computation considered at every decision point the main effort in developing META-GREEDY algorithms is in developing such efficient factored implementations. This approach to developing efficient value of computation approximations is discussed in more detail in Chapter 5.

3.3.11 Simplifying the value-estimate MDP (ii) Macro-Computations

Another way to simplify the value-estimate MDP is to move to a higher level of abstraction using *macro-computations*. A macro-computation, called a *computation strategy* by Horvitz (1990), is a recipe or procedure for achieving some high-level objective, such as reducing value estimate error, which is thought to be useful for solving the agent's problem. In terms of the meta-control model developed above a macro-computation is a fixed meta-control policy π^m which when started executes a sequence of computations to achieve some high-level objective. A macro represents a way of providing the agent with additional prior problem solving knowledge by compiling in information about how to solve common sub-problems.

In general, a macro consists of two parts, (i) an abstract model of its input/output performance which provides the information needed by the agent to decide when to use the macro, and (ii) its decomposition into lower level operations which specifies how to apply the macro in the current situation. The decomposition may be open-loop, where once started the macro applies actions irrespective of the agent's actual situation, or closed-loop where the macro adapts to the

agent's situation to provide more robust operation. The use of such higher-level abstractions has been extensively studied in the classical planning literature (Tate 1977; Korf 1985b) where it forms the basis of the large scale Hierarchical Task Network (HTN) planners. More recently macro-actions have been investigated as a way of scaling MDPs to larger problems (Dean and Lin 1995; Hauskrecht, Meuleau, Kaelbling, Dean, and Boutilier 1998; Parr 1998).

An important issue in modelling and control of macros due to their temporally extended nature is deciding when the macro-controller can control the macro by *monitoring* its performance to interrupt its execution early if necessary. As with the meta-level itself, allowing too fine grained monitoring makes the macro-control problem more complex (becoming as hard as the original control problem if we can switch between macro's after each individual action) and can result in reduced agent performance if the cost of monitoring is a significant computational cost. However, too coarse monitoring can result in reduced agent performance if we lose the opportunity to switch to a higher value alternative. The usefulness of monitoring and interruption depends on the macros run-time performance and particularly whether its output when interrupted early will help the agent make better decisions, that is whether the macro has anytime or contract characteristics (Section 3.2).

Given a set, $m \in \mathcal{M}$, of *interruptible* macros the basic macro-control problem is to choose which macro to execute and *how long* to execute it for before either generating a solution, i.e. executing the Bayes action, or *monitoring* the macro's performance to decide what to do next.

3.3.11.1 Modelling the macro-control problem

We follow Parr (1998) and model a macro as a pre-defined fixed policy π^m which is applicable over a connected sub-set of state-space (which may be the entire state-space) called the macro's *applicable region*. This is a totally general model allowing the macro to use any implementation it wants, such as a FSM or HAM (Parr 1998, Ch. 5), so long as its action choices are Markov.

In the case of macro-computations the preceding analysis suggests the most natural state space is the agent's abstract belief space, Φ, where the macro's policy represents choices between individual base-level computations. However, it is more usual in the real-time AI literature concerned with scheduling macro-computations (Dean and Boddy 1988; Zilberstein 1993; Boddy and Dean 1994) to represent macros at a higher level of abstraction in a factored, real valued, multi-dimensional, space called the macro's *quality* space, $\mathbf{q} \in \mathbb{R}^n$. This is a special abstraction of the agent's internal state which has a readily interpretable semantics as it assumes the *intrinsic value* of the final system output is a monotonically increasing function of the individual quality dimensions. Formally, this is equivalent to assuming that the Bayes action value $V^i_{c_\alpha}(\mathbf{q})$ is *input monotonic*,

Definition 3.1 (Input monotonicity). A function $f(\mathbf{q})$ is input monotonic iff,

$$\forall \mathbf{q}, \mathbf{p} \qquad \mathbf{p} > \mathbf{q} \implies f(\mathbf{p}) > f(\mathbf{q}) \tag{3.30}$$

In this abstract quality space, as the macro's effects are Markov, the most general macro model is a stochastic transition function, $\Pr(\mathbf{q}'|\mathbf{q}, m) \triangleq \Pr(\mathbf{q}'|\mathbf{q}, \pi^m(\mathbf{q}))$, which denotes the probability of getting a solution quality of $\mathbf{q}'$ when the current solution quality is $\mathbf{q}$ by running the macro, m, for a single time step. In the real-time AI literature this model is called a Dynamic Performance Profile (DPP). The DPP can be constructed in a number of ways; it can be learnt off-line or adaptively on-line during the agent's execution. If the macro policy encoded is known the DPP can be computed by averaging over the abstracted features as in (3.10). Alternatively, as Zilberstein (1993) points out, for macros which represent algorithms with known input/output performance, such as Newton's method which has known convergence properties, the DPP can be constructed directly.

To model the execution of the macro for longer durations the Performance Distribution Profile (PDP), $\Pr(\mathbf{q}'|\mathbf{q}, m, \tau)$ is used. This gives the probability of reaching the state $\mathbf{q}'$ after executing the macro m for τ steps starting in state $\mathbf{q}$. The PDP can be used to model the unmonitored execution of a macro for an extended period, τ, as a single macro-action, m_τ, whose transition function is $\Pr(\mathbf{q}'|\mathbf{q}, m_\tau) \triangleq \Pr(\mathbf{q}'|\mathbf{q}, m, \tau)$. The PDP can either be learnt directly in a similar to way to the DPP or calculated from the DPP by "unrolling" the individual macro transitions forward in time using a dynamic-programming type algorithm (based upon the forward Chapman-Kolmogorov equations (Gillespie 1992)),

$$\Pr(\mathbf{q}', \tau|\mathbf{q}, m) = \sum_s \Pr(\mathbf{q}'|s, m) \Pr(s, \tau-1|\mathbf{q}, m) \tag{3.31}$$

Applying this recursion forward in time and normalising for each τ gives $\Pr(\mathbf{q}'|\mathbf{q}, m, \tau)$.

Another model related to the PDP which is used in many macro-control algorithms is the expected performance profile (EPP). This gives a macro's expected output quality as a function of its input quality and the time allocated, i.e. $\hat{q}(\mathbf{q}, m, \tau) = \langle \mathbf{q}'|\mathbf{q}, m, \tau \rangle$. The performance profiles presented in Figure 3.1 are prototypical EPPs for the different algorithm types. The advantage of the EPP is its simple functional form can be used as an approximation for the full PDP when computing resource allocations. This amounts to a type of certainty equivalence assumption. The quality of this assumption, and hence any algorithm based upon it, depends on how well the EPP predicts the true outcome quality. Thus, the EPP should only be used when the outcome quality can be accurately predicted, that is when the PDP has low variance.

To complete the macro control problem model we also require a reward function. For simplicity and consistency we again assume a separable computational cost (Section 3.3.3). Thus the rewards for the Bayes action c_α and executing macro m for duration[2] τ are respectively,

$$\begin{aligned} r(\mathbf{q}, c_\alpha) &= r(\mathbf{q}, c_\alpha, 0) = V^i_{c_\alpha}(\mathbf{q}), \\ r(\mathbf{q}, m_\tau) &= r(\mathbf{q}, m, \tau) = -C(\mathbf{q}, \tau). \end{aligned} \tag{3.32}$$

[2]In general the reward for each destination state/time pair is given by the averaged reward over all paths to this state, $r(p', \tau|p, m) = \sum_s \Pr(p'|s, m) \Pr(s, \tau-1|p, m)[r(s, m) + r(s, \tau-1|p, m)]$.

3.3.11.2 Solving the macro-control problem

If we model extended duration macro executions as single compound actions, m_τ, the macro-control problem can be modelled as a MDP similar in form to the value estimate MDP of Section 3.3.7. This should not be too surprising as the macro-control problem is exactly the same problem, just modelled in a different abstract space. Thus, we can solve the macro-control problem using any of the techniques discussed previously, or any standard MDP solution technique. This approach to macro-control has the advantage of producing a guaranteed optimal policy, w.r.t. the set of macros, which can take account of monitoring costs (Horvitz 1990, p 101) and is adaptive to the macro's actual performance in the current situation.

Indeed, Hansen and Zilberstein (2001a) show how to solve this MDP using standard dynamic programming techniques for the simplified case of a single macro with an arbitrary performance profile and significant monitoring costs. Unfortunately, as with any MDP its computational cost is polynomial in the state space size so computing this policy may be prohibitively expensive, especially if it must be performed on-line. Hence, as for the value-estimate MDP, much work on macro-control has focused on developing tractable approximate solution methods which exploit the special properties of the macro-control MDP.

The meta-greedy algorithm The first macro-control methods proposed (Horvitz 1990) were based upon the META-GREEDY algorithm described in Section 3.3.10 and Figure 3.9 with the MEVC. In the case of macros the MEVC of running the macro, m, for a time τ is,

$$\Delta\hat{V}_{m,c_\alpha}(\mathbf{q};\tau) \triangleq [\hat{V}^i_{m_{0:\tau},c_\alpha}(\mathbf{q}) - C(\mathbf{q},\tau)] - \hat{V}^i_{c_\alpha}(\mathbf{q}), \tag{3.33}$$

$$= \sum_{\mathbf{q}'} \hat{V}^i_{c_\alpha}(\mathbf{q}')\Pr(\mathbf{q}'|\mathbf{q},m,\tau) - \hat{V}^i_{c_\alpha}(\mathbf{q}) - C(\mathbf{q},\tau), \tag{3.34}$$

where $\Pr(\mathbf{q}'|\mathbf{q},m,\tau)$ is the macro's PDP and $[m_{0:\tau},c_\alpha]$ indicates that the agent chooses actions according to the policy m for the first τ time steps and then executes the Bayes action.

It is easy to show that if calculating $\Delta\hat{V}_m(\mathbf{q};\tau)$ is cost free then META-GREEDY is optimal in the following sense,

Theorem 3.1. META-GREEDY *maximises the total value increment,* $\sum_{t=0}^{n}\Delta\hat{V}(\mathbf{q}_t;m_t,\tau_t)$ *iff the macro's marginal values are,*

(i) independent – *if executing* (m',τ') *in* $\mathbf{q}$ *gives* $\mathbf{q}'$ *then forall* $m \neq m'$, $\Delta\hat{V}(\mathbf{q};m,\tau) = \Delta\hat{V}(\mathbf{q}';m,\tau)$, *i.e. executing one macro doesn't affect the marginal value of another, and*

(ii) non-increasing – *if executing* (m,τ) *in* $\mathbf{q}$ *gives* $\mathbf{q}'$ *then* $\Delta\hat{V}(\mathbf{q};m,\tau) \geq \Delta\hat{V}(\mathbf{q}';m,\tau)$, *i.e. for a single macro the marginal values exhibit diminishing returns.*

Proof. Adapting the optimality proof for greedy allocation in the fractional knapsack problem. Aiming for proof by contradiction, assume META-GREEDY fails and w.l.o.g. that the optimal allocation is given by $\mathbf{A}=(a_1,a_2,\ldots,a_n)$ where $a_i=(m_j,1)$ indicates macro m_j is run for a one

time step. Since META-GREEDY fails, then either; (a) there exist $i<j$ such that $\Delta\hat{V}(\mathbf{q}_i;a_i) \geq \Delta\hat{V}(\mathbf{q}_j;a_j)$, or (b) there exist j and $(m,1) \notin \{a_j,\ldots,a_n\}$ such that $\Delta\hat{V}(\mathbf{q}_j;m,1) \geq \Delta\hat{V}(\mathbf{q}_j;a_j)$. In both cases the non-increasing assumption requires that a_j refer to a different macro than m,a_i, hence by the independence assumption; in case (a) we can re-order the allocations greedily without affecting their sum, and in case (b) we can greedily replace a_j by $(m,1)$ to *increase* the sum. Thus, META-GREEDY will have the same or greater value than the optimal allocation $\mathbf{A}$, contradicting its assumed non-optimality. □

This proof is a slight generalisation of the one usually given for the optimality of META-GREEDY, such as (Hansen and Zilberstein 2001a), which consider only a single macro.

Assuming a separable time cost, $\Delta V(\mathbf{q}_t;m,\tau)$ will meet the non-increasing requirement when $\hat{\mathbf{V}}^i(\mathbf{q}_t;m,\tau)$ is a concave function, i.e. exhibits non-increasing returns, and $C(\mathbf{q},|\tau|)$ is a convex function, i.e. exhibits non-decreasing penalties. Thus, we can say that META-GREEDY is optimal w.r.t. $\sum_t \Delta V(\cdot;c_t)$ in this case. Intuitively, these are the most likely cases in practise—as the macro converges towards the optimal solution the improvement in solution value with additional effort usually reduces and, as in the cases of Figure 3.4, usually some external deadline causes the cost of delay to increase with time.

The optimality of META-GREEDY is easiest to see in the single macro case where the independence requirement is trivially satisfied. This is the case most commonly studied in the meta-control literature (Russell and Subramaniam 1995; Hansen and Zilberstein 2001a), as it corresponds to deciding when to stop a single decision making algorithm. Figure 3.10 (from Horvitz (1990, p 45)) illustrates this for the case of concave intrinsic value and linear time cost. In this case the total solution value is maximal when $\Delta V^i(\mathbf{q};m,1) = \Delta C(1) = C(1)$, i.e. when $\Delta V^i(\mathbf{q};m,1) - C(1) = \Delta\hat{V}(\mathbf{q};m,1) = 0$. In fact as Hansen and Zilberstein (2001a) point out in the single macro case the non-diminishing requirement can be relaxed to the requirement that once $\Delta V(\mathbf{q}_t;m,a_t) \leq 0$ it remains so for all later times, $t' \geq t$. This still guarantees optimality because META-GREEDY stops when $\Delta V(\mathbf{q}_t;m,\tau_t)$ first becomes negative and continuing after this point can only decrease the summed value. This diagram also illustrates how increasing the cost of time reduces the optimal thinking time. We will encounter the single macro macro-control problem again in Chapter 6 when we consider the stopping problem.

Justifying the independence assumption in the multiple macro case is more difficult. However, independence does hold when the solution value is a linear function of the quality dimensions, i.e. $V(\mathbf{q}) = \sum_i w_i\mathbf{q}(i)$, and each macro, m_i, depends on and modifies only one of the quality dimensions $\mathbf{q}_i$, so $\Delta V(\mathbf{q};m_i) = w_i\langle\Delta\mathbf{q}(i)|\mathbf{q},m_i\rangle$. This can occur when each dimension corresponds to solution quality of an independent (sub-)problem the agent must solve.

An example of this situation is *continual computation* (Horvitz 2001; Parkes and Greenwald 2001). In this problem the agent does not know the problem, d_i, it will be asked to solve next after some time t. However, it has some knowledge of the distribution over the possible problems, $\Pr(d_i)$, which it can use to improve its performance by working on the likely

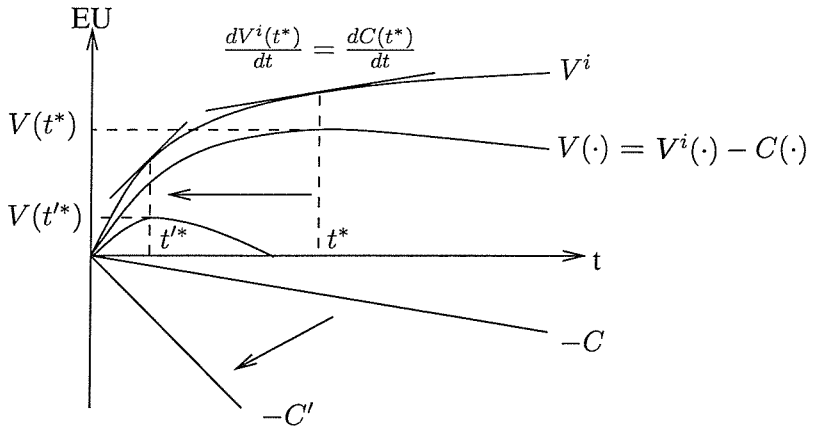


FIGURE 3.10: The optimal stopping problem is to choose when to stop deliberating so that the quality of the eventual decision is maximised. Assuming that benefit and cost are both convex functions of time this diagram clearly shows that this occurs when the benefit rate, dV/dt, equals the rate of cost dC/dt. (Note, $-C$ is shown to reduce graph clutter.)

problems in advance. An example of this type of problem is predictive pre-fetching of web-pages. If macro m_i works solely on problem d_i whose solution quality is quality dimension $\mathbf{q}(i)$, then the expected value of a pre-allocation which has output quality $\mathbf{q}'$ is $V^i(\mathbf{q}') = \sum_i \Pr(d_i)\mathbf{q}'(i)$. Hence, if each macro has diminishing returns the META-GREEDY allocation w.r.t. $\Delta V(\mathbf{q}_t; m_i, 1) = \Pr(d_i)\langle\Delta\mathbf{q}(i)|\mathbf{q}, m\rangle - C(1)$ generates an optimal pre-allocation.

Functional Composition. When independence does not hold META-GREEDY is no longer guaranteed to be optimal. However, in certain circumstances an optimal allocation can still be constructed without having to solve the full macro-control MDP. Zilberstein and Russell (1996) show how to construct an optimal macro-allocation for the special case where the final solution quality, $V(\mathbf{q})$, is given by an input monotonic (see Def. 3.1) function of some sub-set of the quality dimensions, i.e. $V(\mathbf{q}) = f_v(\mathbf{q}(d_1), \mathbf{q}(d_2), \ldots)$, and each of the $\mathbf{q}(d_1), \mathbf{q}(d_2), \ldots$ are recursively computed by macros with input monotonic expected performance profiles so $\langle\mathbf{q}(d_x)\rangle = q_x(\mathbf{q}(d_{x,0}), \ldots; \tau_x)$, where τ_x is a time allocation. Hence, $V(\mathbf{q}) = f_v(q_0(d_{0,0}, \mathbf{q}_{0,1}(\cdot); \tau_{0,0}), q_1(\cdot), \ldots)$.

This structure is representative of many agent problems where generating a solution requires the solution and combination of results from a hierarchy of sub-problems. Target tracking is one example of this type of problem where the quality of the low-level sensor influences the quality of the high-level which in turn influences the quality of the target prediction which influences the quality of the sensing strategies. Other examples include, literal hierarchical search where the quality of an abstract solution influences the quality of later detailed searches, object recognition problems where the quality of the initial features extracted influences the quality of the eventual recognition, and simple serial processing problems where the quality of solution to one task influences the solution quality of later tasks.

Zilberstein and Russell (1996) show that, whilst generating an optimal allocation is NP-hard in general, if the functional dependencies have a tree structure, i.e. each quality dimension occurs in only one function, then an optimal allocation can be computed using a bottom-up *local compilation* routine. Zilberstein and Russell (1996) prove this using an inductive proof. A simpler direct proof is as follows. Consider an arbitrary parent quality function of degree two[3] with child quality functions $q_l(\cdot), q_r(\cdot)$. Let $\mathbf{t}_l, \mathbf{t}_r$ denote the allocation to all the sub-functions of the two children and t the parent's allocation. If we make a certainty equivalence assumption and treat the EPP as representing the true output quality, then the optimal parent value maximising allocation in state $\mathbf{q}$ is given by,

$$q_p^*(\mathbf{q}, \tau) = \max_{t, \mathbf{t}_l, \mathbf{t}_r \geq 0} q_p(q_l(\mathbf{q}, \mathbf{t}_l), q_r(\mathbf{q}, \mathbf{t}_r); t) \tag{3.35}$$

subject to the constraint that $t + \sum_i \mathbf{t}_l(i) + \sum_i \mathbf{t}_r(i) = \tau$. Defining $t_l = \sum_i \mathbf{t}_l(i), t_r = \sum_i \mathbf{t}_r(i)$ we can decompose this to obtain,

$$q_p^*(\mathbf{q}, \tau) = \max_{t, t_l, t_r \geq 0} \max_{\mathbf{t}_l, \mathbf{t}_r} q_p(q_l(\mathbf{q}, \mathbf{t}_l), q_r(\mathbf{q}, \mathbf{t}_r); t). \tag{3.36}$$

As q_p is input monotonic and q_l, q_r are independent for a given l, r resource allocation q_p is maximised by maximising q_l and q_r independently. Thus,

$$q_p^*(\mathbf{q}, \tau) = \max_{t, t_l, t_r \geq 0} q_p(\max_{\mathbf{t}_l} q_l(\mathbf{q}, \mathbf{t}_l), \max_{\mathbf{t}_r} q_r(\mathbf{q}, \mathbf{t}_r); t) \tag{3.37}$$

$$= \max_{t, t_l, t_r \geq 0} q_p(q_l^*(\mathbf{q}, t_l), q_r^*(\mathbf{q}, t_r); t) \tag{3.38}$$

Hence, given optimal quality functions for its children, the optimal parent allocation can be computed by maximising over the total allocations to itself and its immediate children. Applying this process recursively to the children's quality functions, we see that an optimal quality function can be computed by recursive, dynamic programming type, bottom-up construction of optimal quality functions from the input qualities. In this way a compact representation of the optimal allocation policy can be generated by computing the optimal composite performance profile for each quality function offline and recording for each parent time allocation which child allocations generate the optimal parent quality. Then, for any deadline time the optimal allocation can be generated directly by traversing down this tree of optimal policies. An optimal macro execution schedule can be generated from this allocation by simply ensuring that child quality functions are executed before parent ones.

Zilberstein and Russell (1996) go on to show how the method can be extended to non-tree structured functional dependencies. Specifically, they consider the case when a quality function occurs in more than one parent function so the functional dependencies form a directed acyclic graph (DAG). They present two algorithms for this case which work by iteratively refining the repeated functions allocation and then using local-compilation to optimise the other allocations. The CONDITIONING-ALLOCATION guarantees optimal allocations but has

[3]The proof generalises readily to higher degrees.

exponential complexity, alternatively the TRADING-ALLOCATION algorithm is sub-optimal but polynomial time.

Garvey et al. (Garvey, Humphrey, and Lesser 1994; Garvey and Lesser 1993) address essentially the same problem in a multiple-methods framework in their "design-to-time" scheduling system. In this case however the possible functions and interdependencies are severely limited such that; (1) the quality functions can only be maximum or minimum, and (2) the only interdependencies allowed are *enables* and *hinders* constraints. *Enables* constraints require some of a maximising parents child methods (macros) be run in a certain order. *Hinders* constraints reduce the quality of result produced and/or increase the run-time of one method when the lowest quality method is used to compute the result of another. As maximum, minimum and hinders are all input monotonic local-compilation can be used to compute an optimal parent performance profile and child allocation. Garvey, Humphrey, and Lesser (1994) describe a branch and bound method to efficiently perform this computation. As they do not affect the allocation process the enables constraints are accounted for at the scheduling stage. Garvey et al. then go on to show how this approach can be extended to the case when some macros must finish before known deadlines.

3.4 Summary

This chapter has discussed the techniques which can be used to develop deliberative AI components which are suitable for on-line contexts. Two main approaches were presented, bounded deliberation and the more flexible and popular dynamic deliberation. Dynamic deliberation breaks the real-time deliberative component into two parts, the base-level solver and the meta-controller. Two granularities of base-level were identified, strategic (macro) and structural, which influence the type of meta-controller required. It was shown that in the most general case both methods require the meta-controller to solve a MDP. In general this MDP will be too hard to solve directly so simplifying assumptions must be made, such as separable time cost and subjective value equivalence. It was shown that subjective value distributions have many useful properties which can make meta-control simpler by allowing the development of tractable EVC approximations. Based upon these assumptions the META-GREEDY algorithm was developed and shown to be optimal in a wide variety of meta-control situations.

Chapter 4

Decision making and value estimation

This chapter represents the start of the actual implementation of dynamic deliberative on-line agents based upon the ideas discussed in the previous chapters. Specifically, this chapter is concerned with the design and implementation of the base-level solver. This is the part of the agent which solves the core on-line planning problem of using the available information to decide which physical action to execute next. In terms of the incremental decision model of Section 1.2 the base-level solver performs step (6) of Figure 1.2. Thus, this chapter will not be concerned directly with computational resource management, nor how the agent decides what information to gather (steps (2)-(5) of Figure 1.2), only how to make good decisions w.r.t. whatever information the agent currently has available.

Of course, the base-level solver must integrate usefully into the larger dynamic deliberation system so the wider resource management issues cannot be totally ignored. As discussed in Section 3.2.1 the usefulness of the base-level solver in a dynamic deliberation system depends on, (1) its ability to produce near bounded-optimal solutions over a range of resource allocations, (2) the difficulty of its associated meta-control problem. Thus, we should attempt to ensure the base-level solver makes the best decisions possible for a range of available information qualities and is structured in such as way as to ease the meta-control problem.

4.1 Explicitly rational base-level solvers

As discussed in Section 3.3.5 base-level solvers which make rational decisions, as well as being desirable from a decision theoretic point of view, are particularly well suited to efficient value-estimate based meta-control using the META-GREEDY algorithm and EVC approximations. Meta-control can be further simplified if the base-level solver is *explicitly rational.*

Definition 4.1 (Explicitly Rational). An explicitly rational system *explicitly calculates* a set of local value estimates, $\hat{\mathbf{V}}$, and makes decisions by greedily maximising this estimated value.

Explicit rationality saves the meta-controller having to learn and implement a function mapping from the agent's internal state to the implied value estimates. It is also useful for macro-control (Section 3.3.11) as the value-estimates can be used as a direct measure of solution quality, simplifying performance profile construction and macro monitoring.

Thus, we restrict attention to explicitly rational base-levels. This is not as great a restriction as would first appear, and in fact covers a large range of current agent design possibilities ranging from reactive systems based upon pre-complied value estimates, such as potential field methods and utility based behaviour arbitrators (Rosenblatt 2000) through systems which learn value estimates, such as temporal difference (Sutton 1988) and Q-Learning (Watkins 1989) to highly deliberative systems such as pure planning systems which compute value estimates on-line. In fact the only designs specifically excluded are systems with hardwired state→action mappings.

In the terminology of Section 3.3.6 an explicitly rational agent is locally rational and makes the subjective value assumption. Clearly, the performance of such an agent depends critically on the accuracy of its value estimates. Hence, the main design effort when constructing explicitly rational agents goes into ensuring its value estimate generation system produces the most accurate estimates possible. Thus, the main focus of this chapter is on methods and algorithms for efficiently generating value estimates.

4.1.1 The full-observability assumption

There is a slight change in terminology in this chapter, where values, V_π, and value estimates, $\hat{V}_\pi$, are defined without reference to the agent's internal state, b, but as functions of the external state, x, alone. This change has been made for the sake of simplicity and is equivalent to assuming full-observability. In a Markov domain the current state contains all the information necessary to predict the future effects of the agent's actions. Thus, if the agent's policy is modelled as a simple state→action mapping, so $\pi(f_y(x), b) = \pi(x)$, then the value function depends on the state and policy alone, i.e. $V_\pi(x, b) = V_\pi(x)$. Hence, defining V_π in terms of x alone is equivalent to assuming the agent's observations give it complete and correct information about the current external state, i.e. $f_y(x) = x$, so it can implement a state dependent policy. Further, it follows from Wittle's principle of irrelevant information (Whittle 1983, Theorem 3.1) that the optimal policy, π^*, is a function of x alone. So, in the optimal case, restricting the policy in this way has no adverse effect. Notice, that this assumption also implies that x contains complete information about the current objective function R.

4.1.2 Approximating the value function

By assumption, our computationally limited agent cannot represent or compute the optimal value function. Therefore the value estimates used by our explicitly rational agent must be based upon some simpler approximation to the true value where unimportant distinctions are ignored. In the

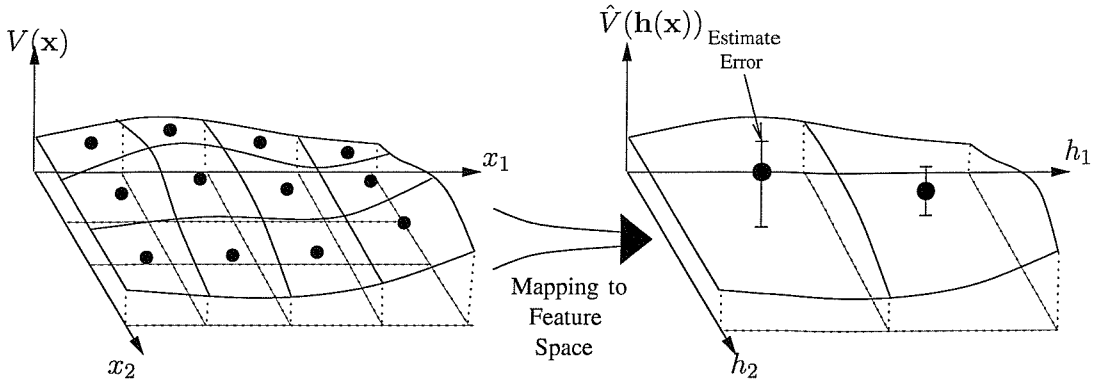


FIGURE 4.1: The complicated value function defined over state space, $V(\mathbf{x})$, is approximated by mapping it to a (usually smaller) feature space where the value function can be more simply represented $\hat{V}(\mathbf{h}(\mathbf{x}))$. A side-effect of this mapping is that errors are introduced into the value estimates. (Note, this is a type of state-aggregation abstraction).

most general terms, this simplification is achieved by decomposing the value function defined over state space, $V(x) \rightarrow \mathbb{R}$, into two parts;

(1) one or more feature detectors, $\{h_1, h_2, \ldots, h_n\}$, which map from state space to a usually smaller feature space, $\mathbf{h} \in \mathcal{H}$, and

(2) a simple estimation function, $\hat{V}(\mathbf{h}(x)) = \hat{V}(h_1(x), h_2(x), \ldots, h_n(x))$, which combines the feature detectors output to produce the desired value estimate.

Good examples of the feature detectors used in search algorithms are; state space connectivity, heuristic estimates and lower bounds. Their associated estimation functions are the value propagation functions such as the Bellman equations. Notice, that the distinction between a feature detector and estimation function is somewhat arbitrary as in many cases a value estimate for one state may be used as an input feature for another. This decomposition is illustrated in Figure 4.1. This figure also emphasises that, in general, the estimation function produces approximate estimates which include some error component. Special care may be necessary to account for these errors and stop them being magnified if one value estimate is used as an input feature for computing another.

Clearly, the correct choice of feature detectors and value estimation function are crucial to minimising both the error of the value estimates and their cost of computation. The feature space should preserve only the minimal amount of information necessary for the required reconstruction accuracy, and the estimate function should use the information provided by the feature detectors to produce the best estimates it can given its resource limits.

For a general function approximation problem defining good features is something of an art, termed *feature engineering* in the machine learning literature. However, for the special case of value function estimation we can use prior knowledge of the problem to identify good candidate features. For example, the state space connectivity itself is generally a good feature as the value of one state is strongly dependent on the values of its neighbours. Alternatively we can produce

features by *relaxing* (Pearl 1984, Ch. 4) the original state space to ignore unimportant details such as distant obstacles in navigation problems, or deleted pre-conditions in STRIPS (Fikes and Nilsson 1993) style planning (Bonet and Geffner 1999). The relaxed state space may be the feature space itself or may be used to define other features, such as bounds on the true value. State space relaxation and value bounds are commonly used for value function approximation in search based planning systems. Another general method of defining feature detectors is to define a set of basis functions (the feature detectors) which map the state space to some vector space where the value function is given in some simple functional form; such a weighted sum, $\hat{V}(x) = \mathbf{w}^{\mathsf{T}}.\mathbf{h}(x)$. This is the approach usually taken to develop tractable value function approximations in reinforcement learning systems.

Given a set of feature detectors there are two main approaches to developing a value estimation function;

(1) **Learning Approach**. Treat computing $\hat{V}(\mathbf{h}(x))$ as an inductive inference problem and use machine learning techniques to learn an approximation based upon some training set.

(2) **Inference Approach**. Use prior information about the properties of the value function and the feature vectors to define a procedure by which we can infer a value estimate from the feature values.

Of course, when estimates are themselves features these techniques may be combined to construct much more powerful *hybrid* estimation techniques. For example prior information may be used to constrain the types of learnt estimate function, or learnt estimates may be used as input features for an estimate function based upon prior information.

For value estimation prior knowledge of the agent's policy is particularly powerful because it allows us to impose constraints on the relationship between value estimates. Propagation or inference algorithms based upon these constraints can then be used to define the value estimation function. Indeed, as Mayer (1994) points out identification of such constraints (or domain invariants) and efficient propagation techniques is the key step in the development of many popular value estimation (and problem solving) algorithms. For example, if the agent uses a greedy policy then the value of one state must obey the Bellman equation with respect to its neighbours, i.e. $\hat{V}(x) = \max_u[r(x,u) + \hat{V}(x')]$. Thus, given a feature set which includes the local connectivity of a state and estimates for the value of some of these states, we can use the Bellman equation to propagate refined estimates to other states. Additional information about the feature set can be used to further refine the estimation process. For example the A* and branch and bound algorithms use the knowledge that estimates are lower bounds to avoid having to compute and propagate estimates which can have no effect on the current state's value.

Prior knowledge can be a powerful tool in designing simple and efficient value estimate functions, and hence powerful explicitly rational agents. Notice however that most of the existing techniques take no account of the errors in the value estimates they rely on and provide

no indication of the errors in their results. The main thrust of this section of the thesis is to investigate the usefulness of providing such error management using probabilistic inference techniques. The hope is that this will improve the agent's performance in two ways,

1. it will provide potentially better value estimates from the same feature set, as probabilistic inference allows us to represent more complex constraints, and
2. the probabilistic value estimates and information about the effects of the possible refinements can be used directly for value-estimate based meta-control purposes.

4.1.3 The semantics of the value estimate

Irrespective of the approach used to compute the value estimates we must be clear about what exactly the returned value means. In Section 2.2.3 Def. 2.3 the value of a situation is defined as the "total objective value accumulated by the future trajectory induced by the policy" from the situation. Thus a value is only defined *with respect to a given fixed policy* and one must be careful to always state the policy from which the value is derived.

That values only have meaning w.r.t. a given policy poses something of a problem when designing an explicitly rational agent. The rationality of its action selections is based upon the assumption that its value estimates are best effort approximations to the true value of the action *to the agent.* Thus, the value we need to estimate, V_π, depends on the agent policy, π, which for an explicitly rational agent is a greedy policy, $\pi_g(\hat{V})$, w.r.t. the approximate value function $\hat{V}$. Hence to justify the rationality of the greedy policy requires $\hat{V} \approx V_\pi = V_{\pi_g(\hat{V})}$. As mentioned earlier (Section 3.3.6) this puts us in something of a "chicken and egg" situation where we need to know the agent policy to define the values upon which the explicitly rational agent's policy is based.

Further, it is a fundamental theorem of dynamic programming that the optimal value function, V^*, is the *only* one whose values are correct w.r.t. greedy action selections, i.e. $V_{\pi_g(\hat{V})} = \hat{V}$ if and only if $\hat{V} = V^*$. Thus, if our agent is explicitly rational w.r.t. its value function then, unless it is optimal, this value function *cannot* be correct.

Thus, not only is a locally rational agent's best value function circularly defined in terms of itself but (unless the agent is optimal) it is a misleading representation of the agent's true future value. By assumption we are only concerned with bounded rational agents whose resource limitations are such that they cannot implement the optimal value function so we are forced to accept the fact that the agent's value function will be misleading, all we can try and do is ensure this does not lead to catastrophic mistakes.

How to avoid catastrophic mistakes is unclear but ensuring the estimated value function is as close as possible to the agent's true one seems like a good start. Too optimistic a value function may require our resource-limited agent to solve problems which are too hard for it, whereas

a pessimistic one may lead to lost opportunities where the agent could perform better than predicted. For example, consider a chess position where a queen sacrifice leads to a complex but guaranteed win in, say, 8 moves. This knowledge may be worse than useless to a resource limited agent if it makes a mistake a few moves into the sequence because it can only look 4 moves ahead and loses the game. In the reverse situation a pessimistic value function may lead the agent to accept a draw in a potentially win-able situation, because the value function assumes it cannot identify the winning line.

Breaking the circular dependency is easier, we simply design our agent so it computes value estimates based upon some policy, $\hat{\pi}$, which we believe is sufficiently close to the agent's true policy to ensure adequate performance, i.e. such that $V_{\hat{\pi}} \approx V_{\pi}$. This approximate policy can then be used as prior information to define the value constraints in an inference based value estimation process. Careful choice of the approximate policy allows us to construct high quality value estimates for low computational cost. As we will see later (Section 4.4.1) this is the approach used in RTA*, MPC and α-β search, which assume a policy consisting of greedy action selection inside some local region of state space and some default policy outside this region.

An approximate policy can also be used to compute the value estimates used to train a learnt approximation. However, a more popular alternative is to simultaneously learn the approximate policy and its associated value function in the light of the agent's experience in (a model of) its environment using reinforcement learning (RL) techniques (Sutton and Barto 1998). Providing, the agent makes locally rational action selections and the value function approximation system meets certain convergence requirements, it can be shown (Gordon 1995) that this technique will converge towards a policy which is within some constant factor of the optimal value function, where the factor is proportional to the error induced by the approximation.

4.2 Incremental search/Predictive control

As mentioned above the correct choice of feature detectors and value estimation function are crucial to the performance of an explicitly rational agent. In particular for a dynamic deliberative agent we require the value estimation process to be dynamic so the meta-controller can trade-off computational effort against value estimate quality. One way of implementing this trade-off is to define the feature set such that the agent can compute more and/or higher quality features with additional computational effort. If the value estimation function is defined so it can utilise the additional features then the meta-controller can control computational effort by controlling the feature generation process.

One way of providing such an adjustable feature set is by providing the agent with an incremental search capability. Incremental search is based fundamentally on the assumption that the planning problem is local such that;

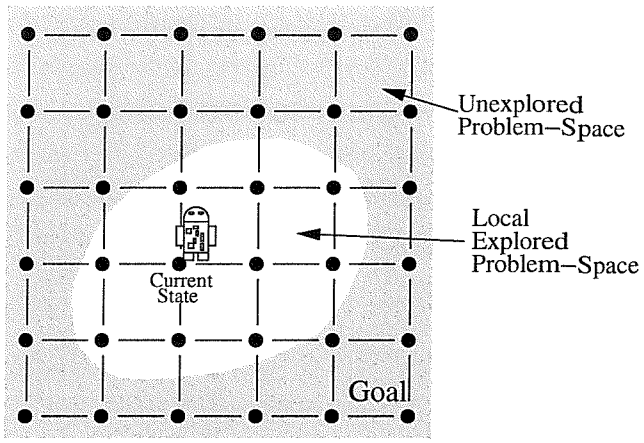


FIGURE 4.2: In a local domain the agent can decide what to do next based upon only the information contained in the local region about its current state. Hence only this local region needs to be explored.

i) when deciding what to do next, the quality of the eventual decision is dominated by a few relevant state-space features, and

ii) the most relevant features are found within the *local* neighbourhood of the current state, as shown in Figure 4.2.

Thus, for a local problem the most important features for decision making (and hence value estimation) can be obtained easily, and more importantly incrementally, by limited forward search through the local state space. Route finding in a maze is an example of a deliberately non-local problem where local information is of little use in finding a good solution. Navigation in obstacle free flat space is a example of a highly local problem as local information about the direction an action moves us is very useful in finding a good path to the goal.

An incremental search algorithm relies on two important types of feature,

1. The local connectivity of the state space, represented as a local state space graph. This is constructed incrementally using the agent's world model to identify the immediate neighbours of any state.

2. Estimates of relevant state-space features for the states in the local state space, represented as labels on the nodes and arcs of the local state space graph.

Thus, the input feature set for the value estimator in an incremental search decision making system is a *labelled graph*, γ, as shown in Figure 4.3, consisting of sets of states labelled with their local state features and actions transitioning between states labelled with their features. For the problems we are interested in the relevant state-space features will generally be action rewards and state future value estimates, though in general any local state or action feature could be used. Given these feature detectors the basic operations of an incremental search algorithm (and the corresponding steps in the generic incremental decision algorithm of Figure 1.2) are,

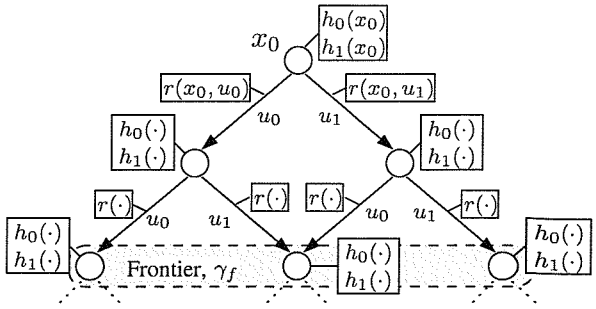


FIGURE 4.3: An example labelled local search space, γ, for input to a value estimator, $\hat{\mathbf{V}}(\gamma)$. This graph consists of a set of states with distinguished initial state x_0, represented as nodes labelled with their local feature sets, $h_0(\cdot), h_1(\cdot), \ldots$, and actions represented as arcs between nodes also labelled with their local feature sets (in this case the transitions immediate reward, $r(x, u)$). γ_f is the set of *frontier* states who have not yet been expanded to add their neighbours to γ.

1. **Output Prediction.** Incrementally generate a labelled local search space graph, γ. (steps (2)-(5) of Figure 1.2)

2. **Control Calculation.** Compute value estimates for a range of possible policies based upon this local search space graph, $\hat{V}(\pi|\gamma) = \hat{V}_\pi(\gamma)$, and select the best for execution, $\alpha = \text{argmax}_\pi \hat{V}(\pi|\gamma)$, (step (6) of Figure 1.2)

3. **Control and Feedback.** Execute some initial segment of the chosen policy (usually just the first action) (step (7) of Figure 1.2) and observe the output to identify the new current state (step (1)). Then repeat.

Additionally, incremental search may include an adaptation phase after feedback where the prediction model, i.e. world model and value estimate function, are modified in the light of experience to more closely represent the current problem or improve problem solving performance (step (8) of Figure 1.2). One simple method of adaptation which can result in significant efficiency gains is to retain the current local search space for later problems.

The basic incremental search algorithm is illustrated in Figure 4.4. The incremental aspect comes partly from the fact that the overall on-line planning problem is solved one action at a time, and partly because the local search space is incrementally extended to improve the information upon which the value estimates are based. Simply put, incremental search uses partial information about the local region around the current state extracted from an internal world model and local feature detectors to infer the required value function estimates.

Of course, few problems are truly local, and those that are would probably be better solved by a reactive controller. However, many problems exhibit some degree of locality, and others can be made so by careful choice of feature detectors. Locality can arise when the problem is episodic in some fashion, so actions before the start of the next episode have little effect on later decisions. Uncertainty about the future also tends to increase a problems locality as the long term effects of early decisions get "smeared out" by the domain uncertainties so only

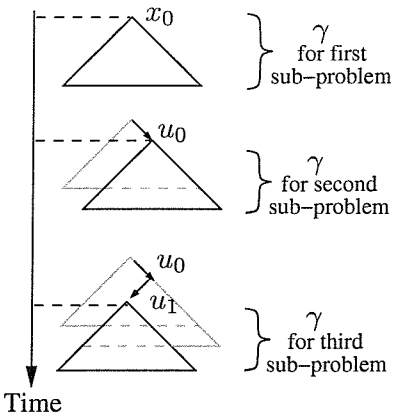


FIGURE 4.4: An example of the general incremental search process showing how the local region of state space explored, γ, is incrementally extended as needed to solve each consecutive sub-problem with only the initial portion of any local policy executed before feedback and control re-computation.

immediate consequences are important (this is one justification for the discount factor used in many MDP formulations (Puterman 1994)). A non-local problem can be made more local by carefully choosing the feature detectors to account for the most important non-local effects. For example, this can be achieved by computing state value estimates in a relaxed state space were only the most important non-local effects are modelled. This approach to improving problem locality is used in many incremental search algorithms. Finally, simply treating a problem as local tends to increase its locality as later decisions are less relevant to the current one because they can still be changed.

Given a local problem incremental search has a number of advantages which have made it popular for solving complex on-line control problems,

- Intuitively it focuses the control effort on the nearby problem where the information is more accurate, leaving until later the more uncertain future problems.
- As the policy is only computed for a small finite region of state space, general optimisation techniques and criteria can be used for control calculation, allowing it to be applied to arbitrarily complex non-linear control problems.
- Feedback provides robustness to external disturbances and model inaccuracies.
- Simple integration of on-line learning techniques to correct model inaccuracies by modifying the predictive world model or value estimate functions parameters.
- Time vs. Space trade-off. The quality of the value estimates can be increased using either more time to increase the size of the local search space, or more space to store more accurate local feature detectors, as shown in Figure 4.5.

Incremental search techniques have been widely studied in the on-line planning literature under a range of different names, such as real-time search (Korf 1990), on-line search (Koenig et al.

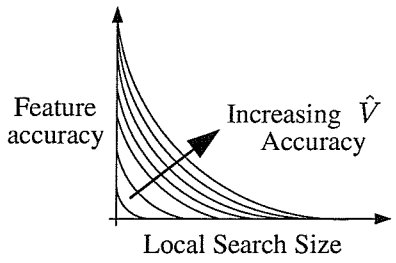


FIGURE 4.5: Space Time trade-off in incremental search.

1997) and agent-centred search (Koenig 2001). The best known version of incremental search is Korf's real time A* (RTA*) (Korf 1990), variants of which have been applied to a wide range of on-line problems including; STRIPS planning (Bonet et al. 1997), moving-target search (Koenig and Simmons 1995), robotic exploration (Thrun et al. 1998), and autonomous navigation in unknown terrain (Stentz and Hebert 1995). Incremental search is also widely studied in game-playing systems, where it forms the basis of the extremely successful *mini-max* search technique (Russell and Norvig 1995, Ch. 5). A variant of incremental search called model predictive control (MPC) or receding horizon control (RHC) has also recently become popular for the optimal control of (continuous) non-linear systems (Garcia, Prett, and Morari 1989; Camacho, Bordons, and Johnson 1999; Mayne, Rawlings, Rao, and Scokaert 2000). Finally, the use of incremental search as a basis for efficient meta-control has been studied in (Russell and Wefald 1989; Hansson and Mayer 1989; Pemberton 1995; Baum and Smith 1997).

4.3 Value estimation in incremental search

Implementation of a decision making system based upon incremental search consists of, (i) designing the world model and local feature detector set, (ii) designing the control calculation system, and (iii) designing the local search space construction meta-controller. In this dissertation it is assumed that the world model and local feature detector have already been provided as part of the problem description. This chapter is concerned with the control calculation or decision making system. Meta-controller design is addressed in subsequent chapters.

The problem the control calculation system must solve is what Pemberton (1995, p. 24) calls the *general incremental decision problem.*

Definition 4.2 (General incremental decision problem). Given a labelled local state graph, γ, choose the child of the root state, x_0, whose reward when summed with the rewards of a further sequence of incremental decisions is maximal.

The approach taken in this dissertation to solving this problem is to estimate and then maximise the value of this sequence of action decisions, V_π. As mentioned above there are three main ways to construct value estimators, either learn it from examples of the agent's supposed performance,

or use prior knowledge (about the agent's actions or the feature detectors) to design an inference mechanism, or to combine these methods into a hybrid approach which both learns state value estimates and uses prior knowledge to infer improved value estimates from the learnt ones. The remainder of this section discusses each of these value estimate construction methods in turn.

4.3.1 Simplifying assumptions

To simplify the value estimation problem we make the following assumptions,

1. The incremental search algorithm will only execute the *first* action of any control policy, hence in current state x_0 only the value estimates for the root's children, $V(x_0, u_1)$, $V(x_0, u_2)$, etc., need be computed, where $V(x_0, u) = V_{u,\pi}(x_0)$.
2. The agent's world model is accurate and correct, and its local feature detector for actions includes (an estimate of) the actions immediate reward, $r(x, u)$.
3. The optimal value function, V^*, is well-defined and finite for all states. Specifically, we restrict attention to multiple goal shortest path problems where;
   (a) all transition rewards are bounded negative, i.e. $r(x, u) \leq k < 0$, so the state graph has no non-negative reward cycles and $V^* < \infty$,
   (b) there are a set of terminal (or goal) states[1], $\mathcal{X}_g$, at least one of which is reachable from any state, so optimal trajectories will eventually terminate and $V^* > -\infty$.

The class of multi-goal shortest path problems is broader than it first appears as any problem with bounded finite transition rewards, i.e. $|r(x, u)| \leq k < \infty$, and no non-negative reward cycles can be transformed into this form without changing the optimal policy by simply subtracting k from every transition reward so $r'(x, u) = r(x, u) - k$. Note, an important side-effect of these assumptions is that a maximal reward path between any two nodes cannot contain a cycle, hence any optimal policy forms a forest over state space,[2] with each tree rooted in a goal state.

4.4 Inference approaches

4.4.1 The MINIMIN estimate

MINIMIN is the classic and simplest incremental search algorithm forming the basis for Korf's (1990) RTA* algorithm. The fundamental assumption underlying the MINIMIN algorithm is that, given a local search space γ the agent's true policy, π, can be approximated by a policy, π^*_γ, which maximises the combined value of its trajectory through the local search space, γ, and the

[1] Note, any state which has a zero cost self-transition is treated as a terminal state.
[2] If we allowed more than one optimal act per state then this would be a DAG.

estimated value of some default policy, V_{π^0}, from the frontier of the local search space. We can see this approximation policy as a type of greedy Bayes policy with respect to the local search space transition costs and the frontier value estimates. The value of π^*_γ is given by,

$$V_\pi(x) \approx V_{\pi^*_\gamma,\pi^0}(x) = \max_{\pi'} \left[R(T(\pi';x),0{:}n{-}1) + V_{\pi^0}(x_n)\right], \quad (4.1)$$

$$= \max_{\pi'} \left[\sum_{t=0}^{n-1} r(x_t,\pi'(x_t)) + V_{\pi^0}(x_n)\right], \quad (4.2)$$

subject to the constraints that; $x_0 = x$, π' is such that the trajectory always remains within γ, and $x_n \in \gamma_f$ where γ_f is the set of frontier states which includes any terminal states in γ. As an optimal policy cannot contain cycles it is guaranteed to pass through a frontier or terminal state in finite time for a finite γ, so $V_\pi(x)$ is always well-defined. This value equation can be re-written in recursive form to give,

$$V_{\pi^*_\gamma,\pi^0}(x) = \max_u \left[r(x,u) + V_{\pi^*_\gamma,\pi^0}(f_x(x,u))\right], \quad (4.3)$$

subject to the boundary condition that $V_{\pi^*_\gamma,\pi^0} = V_{\pi^0}(x)$ for all states $x \in \gamma_f$. This is just the normal optimal DP backup equation (2.10) restricted to the local search space γ with terminal costs given by V_{π^0}.

As the local search space increases in size then the MINIMIN value estimate $V_{\pi^*_\gamma,\pi^0}$ will tend towards the optimal value V_{π^*}, with the limiting case being when γ is the entire state space. Thus, we can treat MINIMIN as either computing the value of the policy $[\pi^*_\gamma,\pi^0]$ or as an approximation for π^*.

4.4.1.1 Implementation issues

To use MINIMIN for a particular problem we require that a state's local feature detectors provide an estimate of the state's future value under the default policy, i.e. $h_0(x) \approx V_{\pi^0}(x)$.

The child value estimates, $V(x_0,u)$, can be computed efficiently by noting that the value of any state, x, under $[\pi^*_\gamma,\pi^0]$ consists of two parts, the value of the best trajectory to a frontier node x_f, and the value from that node $V_{\pi^0}(x_f)$. Thus (4.1) becomes,

$$V_{\pi^*_\gamma,\pi^0}(x) = \max_{x_f\in\gamma_f} \left[\max_{\pi',\text{ s.t. } x_n=x_f} R(T(\pi';x),0{:}n{-}1) + V_{\pi^0}(x_f)\right], \quad (4.4)$$

$$= \max_{x_f\in\gamma_f} \left[r^*_\gamma(x,u_{x_f}) + V_{\pi^0}(x_f)\right], \quad (4.5)$$

where $r^*_\gamma(x,u_{x_f})$ is the value of the maximal reward path (MRP) from x to x_f. Thus the interior of the local search space can be abstracted away by treating the choices from x as being between a set of abstract actions, u_{x_f}, each of which transitions directly to a frontier node, x_f, for an immediate reward, $r^*_\gamma(x,u_{x_f})$. As a maximal reward path cannot contain cycles, the set of maximal reward paths from x to γ_f must form a tree, which can be computed efficiently using

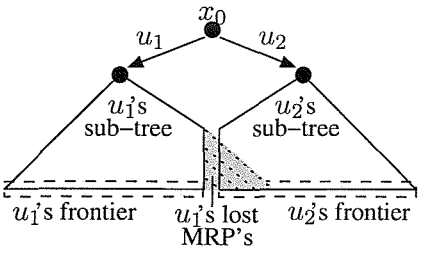


FIGURE 4.6: This diagram shows how the root's maximum reward path (MRP) tree can be used to split the local search space into disjoint sub-trees. The dark shaded region also shows how this decomposition may not include all the MRPs (and frontier states) reachable after u (potentially raising u's value estimate).

Djkstra's algorithm. Hence, x's MINIMIN value can be computed without computing the value of any other states in γ.

By definition, restricting attention to paths in a sub-tree of the roots maximal reward path tree cannot exclude the maximal reward path from the best sub-tree. Thus, only frontier nodes in a child's sub-tree of the root's MRP tree need be considered to guarantee the *best* child's value estimate is correct, though, as shown in Figure 4.6, this may result in pessimistic estimates for the other children. This can represent a significant saving in computational effort as each frontier state need only be considered once and the root's maximal reward path tree can be constructed during γ's construction.

4.5 Learning approaches

Learning approaches use training information about the feature→value mapping to learn a value estimate function. Therefore in this section we assume we have available from somewhere a large set of training examples, $\mathbf{D}$, consisting of mappings from γ_i to child value estimates $\{V_{\hat{\pi}}(x_i, u_1), V_{\hat{\pi}}(x_i, u_2), \ldots\}$, where $V_{\hat{\pi}}(x_i, u)$ indicates that the estimates have been computed w.r.t. some training policy $\hat{\pi}$, most likely the optimal policy π^*.

4.5.1 The gold standard estimate (GSE)

The simplest and most direct method for learning the value estimates is to treat the labelled local search space graph, γ, as a single input feature for the value estimator. Generally γ will not map to a unique set of child value estimates. Given this uncertainty normal decision theoretic arguments prove that the best possible estimate is the expected value given γ and the distribution of value estimates the agent is likely to encounter,

$$\hat{V}(u|\gamma) = \mathrm{E}\{V(u)|\gamma, \pi\} = \sum_{v \in \mathbb{R}} v \Pr(V_\pi(u){=}v|\gamma, \pi) = \sum_{x \in \mathcal{X}} V_\pi(x, u) \Pr(x|\gamma, \pi), \qquad (4.6)$$

where $V_\pi(x,u)$ is the value to the agent of (x,u) and $\Pr(x|\gamma,\pi)$ is the probability of being in state x given the agent has generated feature vector γ. The functional relationship, $V_\pi(x,u)=v$, enables the change of variable from v to x. Following Mayer (1994), we call this the gold standard estimate as it is the gold standard by which other estimates will be judged. Unfortunately, as the true distribution $\Pr(x|\gamma,\pi)$ and $V_\pi(x,u)$ are generally unknown we must use the training set to compute the sample GSE,

$$\hat{V}(u|\gamma) \approx \mathrm{E}\{V_{\hat{\pi}}(u)|\gamma,\mathbf{D}\} = \sum_{\{i:\gamma_i=\gamma\}} V_{\hat{\pi}}(x_i,u)\Pr(x_i|\gamma_i). \tag{4.7}$$

This is the best value estimate possible given *only* the information contained in the training set, $\mathbf{D}$, i.e. no additional prior information.

4.5.1.1 Implementation issues

$\hat{V}(u|\gamma)$ can be computed directly using (4.7) and stored in a lookup-table. Unfortunately, the size of this table is proportional to the number of possible feature vectors. This grows at an *exponential rate* with respect to the number of states in γ, i.e. $\mathcal{O}\left(\left(v_x v_u^{|\mathcal{U}|}\right)^n\right)$, where n is the maximum number of states in γ, v_x, v_n are the number of distinct feature values a state and action can have respectively and $|\mathcal{U}|$ is the number of actions per state. Notice, that the base of this exponential is usually quite large, e.g. for a problem with only, v_x=4, v_u=2, $|\mathcal{U}|$=4 then $v_x v_u^{|\mathcal{U}|}$=64 giving 1073741824 possible feature vectors for γ with 5 states. Fortunately, feature values are usually highly correlated (which is why inference approaches work) so many of these combinations will not occur in practise. However, the GSE still presents a significant computational burden both for the agent to simply store the look-up table and for the designer to generate enough training examples to fill it. For reliable estimates at least one, but preferably >10, training examples are required per table entry.

One way to reduce this burden at the expense of reduced estimate accuracy is to only use γ's most relevant features to compute the GSE. For a pure learning solution the relevant features can be found using machine learning techniques, such as independent/principle components analysis (ICA/PCA) (Hyvärinen, Karhunen, and Oja 2001) or ARD (Neal 1995). Alternatively, we can use prior knowledge to identify the relevant features to give a hybrid feature estimator. The MINIMIN analysis of Section 4.4.1.1 suggests that one such set of relevant features are frontier states feature values, sub-tree allocations and maximum reward path values from the root. Considering only this information reduces the lookup table size to $\mathcal{O}((v_x v_r)^{n_f})$, where v_r is the number of distinct maximum reward path values and n_f is the number of frontier states. This can be a significant complexity reduction. This value estimate is called GSE-frontier (GSE-F).

4.5.2 The Bayesian Problem Solver (BPS) estimates

The GSE provides the best possible value estimate, but is tremendously inefficient in terms of the training effort and run-time storage required for $\hat{V}(u|\gamma)$. In fact, the results in Section 4.15 show for the toy problems considered, the GSE requires only on the order of 10 states in γ to obtain near perfect performance. Unfortunately, obtaining this performance requires look-up tables 100s of mega-bytes in size. One source of complexity in the GSE is computing and storing the joint conditional probability table (CPT), $\Pr(V_\pi(u){=}v|\gamma, A)$, needed to compute $\mathrm{E}\{V(u)|\gamma, \pi\}$. Thus one approach to reducing the GSEs complexity is to develop more compact and efficient ways of using this CPT.

One standard way of simplifying a CPT is to structure it as a Bayesian network (BN) (Pearl 1988), where any independences between the CPT's component variables are used to decompose it into intersecting conditionally independent sub-sets of variables, called cliques. This decomposition allows the probability of a partial assignment to CPT variables to be found by marginalising over, i.e. integrating out, the the unknown variables in each clique independently. For example, if the global CPT is decomposed into three cliques as shown in Figure 4.7(b) then the probability of a partial assignment, e, is given by,

$$\begin{aligned}
\Pr(e) &= \sum_{\sigma} \Pr(\sigma, e) = \sum_{\{\sigma:e\}} \Pr(\sigma_{0\cup 1\cup 2}), && (4.8)\\
&= \sum_{\{\sigma:e\}} \Pr(\sigma_2|\sigma_{0\cup 1}) \Pr(\sigma_1|\sigma_0) \Pr(\sigma_0), && (4.9)\\
&= \sum_{\{\sigma:e\}} \Pr(\sigma_2|\sigma_{0\cap 2}) \Pr(\sigma_1|\sigma_{0\cap 1}) \Pr(\sigma_0), && (4.10)\\
&= \sum_{\{\sigma_0:e_0\}} \Pr(\sigma_0) \sum_{\{\sigma_2:e_2,\sigma_{0\cap 2}\}} \Pr(\sigma_2|\sigma_{0\cap 2}) \sum_{\{\sigma_1:e_1,\sigma_{0\cap 1}\}} \Pr(\sigma_1|\sigma_{0\cap 1}), && (4.11)
\end{aligned}$$

where, σ is a complete assignment, σ_i is the projection of σ onto the variables in clique i, $\sigma_{i\cup j}, \sigma_{i\cap j}$ are respectively the projection of σ onto the union and intersection of cliques i and j, and $\{\sigma_s{:}e\}$ denotes the set of all possible assignments to set s such that partial assignment e is true. This is essentially the junction tree algorithm (Pearl 1988) for inference in BN.

This approach of decomposing $\Pr(V(u)|\gamma, A)$ into a BN is the method used in Hansson and Mayer's (1989) Bayesian Problem Solver (BPS), which has been applied to single agent search (Mayer 1994) and scheduling problems (Hansson and Mayer 1994; Hansson 1998). The advantage of this decomposition is that the total size of the clique CPTs is much less than the full CPT, i.e. $\mathcal{O}(|c|(v_{cv})^{n_c})$ where $|c|$ is the number of *distinct* clique types, v_{cv} is the number of values for each clique variable and n_c is the number of variables in the clique. Notice, this is an exponential size reduction, potentially reducing both the training effort and run-time storage requirements by many orders of magnitude. For example in the case discussed for the GSE where each state has 64 values, decomposing a 5 state γ into 2 cliques of 3 states with a single state overlap reduces the storage to $2 * 64^3 = 52488$, a saving of over 20,000 times.

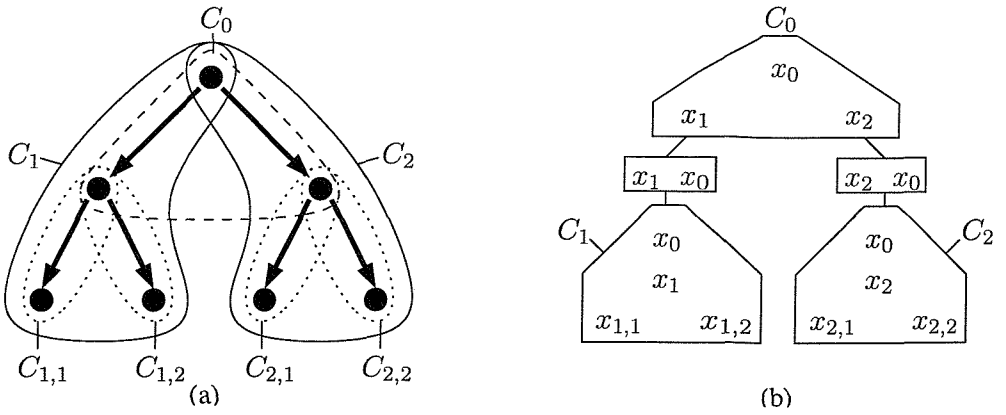


FIGURE 4.7: BPS(2)'s decomposition of γ into cliques. Part (a) shows the grouping of nodes into local cliques where all nodes are within 2 arcs of each other (or equivalently within 1 arc of the seed node), part (b) shows the resulting CPT decomposition into cliques and separators with redundant cliques removed.

4.5.3 Implementation issues

To construct a BN decomposition of $\Pr(V(u)|\gamma, A)$ when γ is known with certainty we need to introduce some additional explanatory variables to represent the underlying hidden local world state and capture the dependencies between cliques. As the quantities we wish to estimate are the state values, $V(x)$, these are the obvious candidates for the explanatory variables. Further, the functional dependence between $V(x)$ and x's successor values ensure $V(x)$ is conditionally independent of all other states values (though not necessarily their feature values). The roots child action value distributions can now be found using Bayes rule, $\Pr(V(u)|\gamma, \pi) = \Pr(V(u), \gamma|\pi)/\Pr(\gamma|\pi)$, where $\Pr(V(u), \gamma|\pi)$ can be computed using (4.11) and $\Pr(\gamma|\pi)$ is a normalising constant found by marginalisation, i.e. $\sum_v \Pr(V(u) = v, \gamma|\pi)$.

To construct the cliques we again invoke the locality assumption, which says that the most important influences on a state value are usually those nearest to it, to limit cliques to locally connected sub-sets of γ. Hansson and Mayer define a range of possible decompositions and associated estimation functions, BPS(1), BPS(2),..., BPS(d), based upon the clique size. Specifically, BPS(d) assumes that the variables associated with state x are conditionally independent given all states within d actions of x. Thus, all the variables within d actions of x form a clique centred on x. Figure 4.7(a) shows Hansson and Mayer's preferred BPS(2) clique decomposition for a tree structured γ. Notice, how adjacent cliques which are sub-sets of one another are collapsed into a single clique in Figure 4.7(b).

Finally, for efficiency two additional simplifying assumptions are made in BPS. The first is *position-independence* which assumes all cliques have the *same* CPT. This results in a further reduction in storage and training effort as only one clique CPT need be learnt which is of size $\mathcal{O}((v_{cv})^{n_c})$. The second assumption is that γ is tree structured. This assumption is necessary as the complexity of exact inference in a BN is proportional to the number of variables in the maximal cut-set of the network (Pearl 1988). For a tree this is simply the size of the largest

clique, however for a graph it may be much larger (up to the size of the complete graph in a fully connected network). Thus, the tree restriction ensures the complexity of computing the BPS estimate is proportional to the size of γ. A tree version of γ is constructed in BPS by forward search from the root state with states reached by more than one path duplicated. Note, this may introduce errors as two tree nodes which represent the same state may be assigned different values, and the tree may be exponentially larger than the graph (offsetting any reduction in computational complexity).

4.6 Hybrid approaches

Hybrid approaches combine aspects of both learning and inference. Learning is used to construct local state value estimators, $\hat{V}(x)$, and prior information used to construct inference techniques to combine the local state value estimates to produce improved global estimates. Hybrid methods consist of two sub-components,

1. **Local state value estimator.** This is usually a simple learnt feature→value estimate mapping, but may include some inference aspects.
2. **Combining, global state value estimator.** This is usually based upon prior information about the dependencies between state values, but may include some learning to compensate for biases in the learnt estimates.

As these two components are modular with respect to each other we discuss methods of implementing each in turn next.

Note, technically BPS is a hybrid method as it uses learning to construct the clique CPTs (which we can think of as probabilistic local value estimators) and prior information (in the form of the degree of locality assumed) to define the cliques inter-dependencies and hence the method used to combine local clique CPTs to compute the improved global value estimates. Also MINIMIN is easily hybridised using learnt functions for the frontier state's future value estimates so $V_{\pi 0}(x) = \hat{V}(x|\gamma)$. In essence this hybrid, called MM-CAL, *calibrates* MINIMIN's frontier value estimates using the training examples to remove any bias and hopefully improve its performance.

4.7 Generating local state value estimates

As discussed earlier (Section 4.1.2) we are primarily interested in probabilistic methods for combining the value estimates, so we treat the state $\hat{V}(x)$ learning problem as one of learning the state's value distribution, $\Pr(V(x)|\gamma)$. In this dissertation three methods for learning and generating the state value distributions have been tried based upon different independence assumptions for $V(x)$ w.r.t. the features in γ. These are; state feature estimators, clique feature estimators and parent state value dependent estimators.

4.7.1 State local value estimates

These are the simplest value estimates where it is assumed that the state value estimate depends only on the states individual feature set, so $\Pr(V(x)|\gamma) \approx \Pr(V(x)|\mathbf{h}(x))$.

4.7.2 Clique local value estimates

The next simplest value estimators, like BPS, invoke a locality assumption to assume that the state values depend on only the features of clique of nodes in the immediate neighbourhood of x, so $\Pr(V(x)|\gamma) \approx \Pr(V(x)|\gamma_x)$, where γ_x is a sub-set of γ centred around the state x. However, unlike BPS, computing this estimate does not require any probabilistic inference as $V(x)$ is assumed to be independent of its neighbours value estimates. In this dissertation only cliques which contain only x and its *parent* state are considered. So $\Pr(V(x')|\gamma) \approx \Pr(V(x')|\mathbf{h}(x'), \mathbf{h}(u), \mathbf{h}(x))$, where u is the action taken to get from the parent state x to the current state x'.

4.7.3 Parent value dependent estimates

The final value estimator performs a limited form of probabilistic inference where the value of state x' is assumed to depend only on its local features, $\mathbf{h}(x), \mathbf{h}(u)$ and the true value of its parent state x. As only a distribution over parent values is known under this assumption estimating $\Pr(V(x)|\gamma)$ becomes a problem of *Bayesian belief updating*, where the child states value distribution is given by,

$$\Pr(V(x')|\gamma) \approx \Pr(V'|\,\mathrm{p}_V, \mathbf{h}', \mathbf{h}_u) = k \Pr(\mathbf{h}'|V') \Pr(V'|\,\mathrm{p}_V, \mathbf{h}_u) \tag{4.12}$$

where k is a normalising constant given by $k = \int \Pr(\mathbf{h}'|V') \Pr(V'|\,\mathrm{p}_V, \mathbf{h}_u) dV'$. Note, for brevity the following abbreviations have been used; $V{=}V(x)$, $V'{=}V(x')$, $\mathrm{p}_V = \Pr(V(x))$, $\mathbf{h}'{=}\mathbf{h}(x)$ and $\mathbf{h}_u{=}\mathbf{h}(u)$. To complete this estimate we require two CPTs, the state features likelihood, $\Pr(\mathbf{h}'|V')$, and, the child's prior value distribution given the parent distribution and transition, $\Pr(V'|\,\mathrm{p}_V, \mathbf{h}_u)$.

The likelihood can be found by simply learning $\Pr(V, h)$ from which Bayes rule gives $\Pr(\mathbf{h}|V) = \Pr(V, \mathbf{h}) / \Pr(V)$. The child's prior distribution could also be learnt, but this CPT may be excessively large. Instead the approach taken in this work is to note that if $\pi(x){=}u$ then $V_\pi(x) = r(x,u) + V_\pi(x')$ and the child's prior distribution is known to be $\Pr(V') = \mathrm{p}_V(V{=}V'{+}r(u))$, where $r(u) = r(x,u)$. Assuming this happens with probability p_s then

$$\Pr(V'|\,\mathrm{p}_V, \mathbf{h}_u) = p_s\, \mathrm{p}_V(V'{+}r(u)) + (1 - p_s) \Pr(V'|\,\mathrm{p}_V, \mathbf{h}_u, \pi(x){\neq}u). \tag{4.13}$$

To estimate p_s we simply assume that all children have equal probability of being the correct successor so $p_s{=}1/|\mathcal{U}|$. For $\Pr(V'|\,\mathrm{p}_V, \mathbf{h}_u, \pi(x){\neq}u)$ we use either the unconditional value distribution, $\Pr(V)$, which pessimistically assumes we have no prior information about the child's distribution, or the state local value distribution, $\Pr(V|\mathbf{h}')$, which is less pessimistic but mathematically unsound as $\mathbf{h}'$ is used in both the prior and likelihood during the update calculation. These distributions have the advantage of requiring no additional learning as they can be computed directly from $\Pr(V,h)$ using $\Pr(V) = \sum_{\mathbf{h}} \Pr(V,\mathbf{h})$ and $\Pr(V|\mathbf{h}) = \Pr(V,\mathbf{h})/\Pr(\mathbf{h})$. Hence, we have either,

$$\Pr(V'|\,\mathrm{p}_V, \mathbf{h}', \mathbf{h}_u) = k\Pr(\mathbf{h}'|V')\left[p_s\,\mathrm{p}_V(V'{+}r(u)) + (1-p_s)\Pr(V)\right], \quad (4.14)$$

or,

$$\Pr(V'|\,\mathrm{p}_V, \mathbf{h}', \mathbf{h}_u) = k\Pr(\mathbf{h}'|V')\left[p_s\,\mathrm{p}_V(V'{+}r(u)) + (1-p_s)\Pr(V|\mathbf{h}')\right]. \quad (4.15)$$

4.8 Combining the local state value estimates

Given probabilistic estimates for the state values in the local search graph, γ, the combination component combines these estimates to compute more accurate global value estimates for the root's children $V(x_0,u_1), V(x_0,u_2)$, etc. Efficiently performing this combination relies upon making useful assumptions about dependencies between value estimates, usually based upon some assumption about the policy the agent will execute (such as that it is greedy w.r.t. some approximate value function).

4.8.1 The E{MRP} estimator

One key issue for value estimation which has not yet been discussed is the incremental nature of the agent's operation and in particular the dependence of its future action choices on the additional information gathered before the choice. Thus, knowing when and what additional information the agent will have in the future can be critical to accurately computing a decisions value and hence making good decisions. In this section we develop the *expected maximum reward path* (E{MRP}) estimator, which provides the agent with a limited ability to consider the value of additional information.

As an example of when taking account of future additional information can be valuable, consider again Figure 3.5 where Robbie must choose behind which of two doors lies a goal. If the goal has unit value then without additional information Robbie's expected decision value is 0.5. However, if he knows that by the time the choice must be made he will have identified the correct door his expected decision value assuming the additional information is 1. Thus, Robbie's best decision when choosing between a guaranteed reward of 0.55, say, and choosing a door, depends critically on knowing what additional information he has at the door choice point.

Thus an incremental search agent needs to consider the value of the information (Section 3.3.9) it will have available in the future when deciding on the best current action. Doing so requires firstly predicting when and what additional information the agent will gather and secondly using this prediction to compute how valuable the information will be for improving its future action choices. As control of information gathering is a meta-control task whereas action choice is a decision making one consideration of additional information requires the decision making system model and solve the combined embedded meta-control/original problem of Section 3.3.1 Figure 3.3. In that section this model was deemed intractable so again we cannot require the agent solve it directly but must instead develop some tractable approximation.

The learning and inference methods discussed earlier represent two such approximations. Learning methods avoid considering the value of additional information by assuming this value is already included in the training sets value estimates. The MINIMIN approximation simply assumes there are no more incremental decisions and hence no additional information, allowing it treat the current frontier state value estimates and the policy derived from them as fixed.

4.8.1.1 Modelling additional information

To model the acquisition of additional information and how this information effects the value of a rational agent's decisions we use the value estimate abstraction discussed in Section 3.3.5. This relies on the following assumptions,

1. the agent's current set of value estimates, $\hat{\mathbf{V}}$, are assumed to be unbiased estimates of the true value given the agent's current beliefs b, i.e. $\hat{\mathbf{V}} = \mathrm{E}\{\mathbf{V}|b\}$.
2. the effect of additional information, i, is modelled as a conditional distribution over future value estimates, $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, i)$,
3. in any situation the agent selects the Bayes action w.r.t. its beliefs at that time, so $\pi(x) = \mathrm{argmax}_u \hat{V}(x, u)$.

To apply this model to the problem of predicting the effects of additional information on the value of future action choices, consider the case shown in Figure 4.8, where in some future state x the agent has to select one of the actions $\{u_1, u_2, ..., u_n\}$. Assume the agent knows that by the time it reaches x its new set of value estimates will be $\hat{\mathbf{V}}$ with probability $\Pr(\hat{\mathbf{V}})$. As the agent is explicitly rational it also knows that when its estimates are $\hat{\mathbf{V}}$ a potentially better estimate for the expected value of its policy from x can be inferred using the standard dynamic programming backup equation,

$$V(x) = \max_u [r(x, u) + V(f_x(x, u))] = \max_u \hat{V}(x, u). \tag{4.16}$$

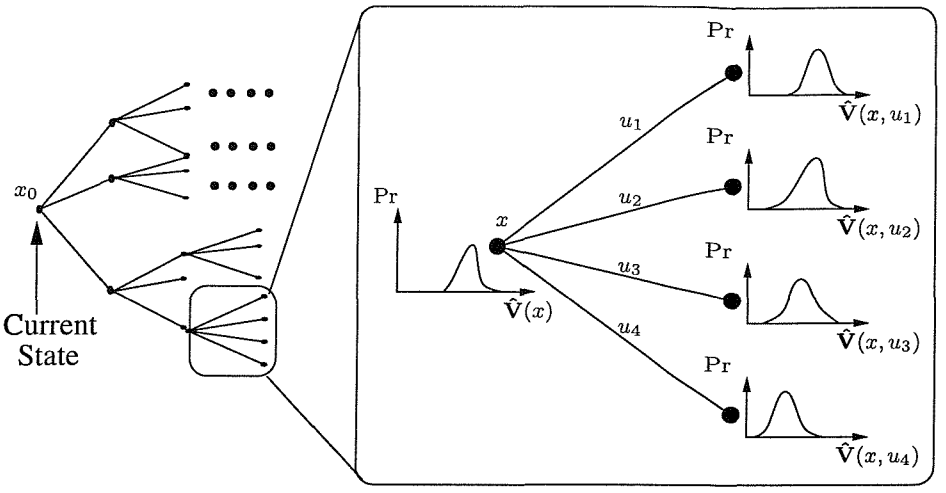


FIGURE 4.8: Given distributions for the expected value of the different outcomes of a future decision, $\hat{V}(x, u_i)$, the agent's inference procedure allows it to determine the expected value of the decision itself, $\hat{V}(x)$ (under certain simplifying assumptions outlined in the main text).

Hence, the total probability that $V(x)=v$ can be found by summing probabilities for all the cases where x has the same backed-up value,

$$\Pr(V(x)=v) = \sum_{\hat{\mathbf{V}}} \Pr\left(\hat{\mathbf{V}}, \left[\max_u \hat{V}(x,u)\right]=v\right), \tag{4.17}$$

$$= \Pr\left(\left[\max_u \hat{V}(x,u)\right]=v\right). \tag{4.18}$$

Notice, this calculation implicitly assumes that the $\hat{\mathbf{V}}$s (of x's successors) *are correct* so $\Pr(\hat{\mathbf{V}})$ gives the actual probability that the world is such that the agent will obtain the indicated value. In this case, as the agent will actually pick the maximum value action (4.18) gives the probability that the world is such that the action the agent picks will realise the value v. Integrating both sides to transform from probability density function (PDF) to cumulative density function (CDF) we obtain;

$$\Pr(V(x)<v) = \Pr\left(\left[\max_u \hat{V}(x,u)\right]<v\right). \tag{4.19}$$

Now, the maximum of a set can only be less than v if all the members of the set are individually less than x, so the max can be replaced with an AND and the inequality brought inside to give;

$$\Pr(V(x)<v) = \Pr\left(\underset{u}{\text{AND}}\left[\hat{V}(x,u) < v\right]\right), \tag{4.20}$$

$$= \Pr(\hat{V}(x,u_1)<v, \hat{V}(x,u_2)<v, \ldots). \tag{4.21}$$

Equation (4.21) is the general equation for inferring the Bayes action's value distribution given a joint distribution over the future values of the possible action choices.

In the case considered here the agent can use (4.21) to compute an improved estimate for x's value distribution given $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, i)$ which represents its current belief about the likely effect

of the additional information it will acquire given its current value estimates. So,

$$\Pr(V(x)<v|\hat{\mathbf{V}},i) = \Pr(\hat{V}'(x,u_1)<v,\hat{V}'(x,u_2)<v,\ldots|\hat{\mathbf{V}},i). \tag{4.22}$$

4.8.1.2 The last incremental decision assumption

Equation (4.21) provides a way of inferring an improved value distribution for a single state, x, assuming Bayes action selection and that at x the agent's estimated action values reflect the true expected values. To correctly model the effects of additional information we need to extend this model to cater for differing information availabilities, and hence different estimated value function distributions, at different future states. Unfortunately, it is unclear how to do this efficiently.

The problem is that the agent's future policy depends on both its current state *and* its current estimated value function (because the agent selects actions based upon its estimated value function). Further, the effect of additional information on agent's future value estimates depends on its current estimates, as discussed in Section 3.3.5. Thus, predicting value of the agent's future policy requires the changes in the state *and* estimated value function be modelled together. As future estimated value functions are only known stochastically finding the best current action is equivalent to solving a MDP defined over the combined space of states and value function estimates. The problem is that the size of this MDP is $\mathcal{O}\left(((v_x)^n)^n\right)$ making it infeasible to compute a solution on-line.

In essence by asking the agent to optimise its current actions with respect to the unknown outcome of future information gathering operations, i.e. observations, we are asking the agent to solve a partially observable Markov decision problem (POMDP) (Sondik 1978) with the true state values the hidden world state. Current state of the art POMDP systems can only solve problems with, at most, 10s of physical states, (Littman, Cassandra, and Kaebling 1995; Kaelbling, Littman, and Cassandra 1997; Zhang and Zhang 2001), so attempting to solve the state value function POMDP in real-time is clearly infeasible.

To avoid this problem we simply assume that the current decision is the *last incremental decision* (Pemberton 1995) such that for every subsequent decision the value estimates are fixed, allowing the remainder of the policy to be executed open-loop. It is assumed that the values become fixed because immediately after this decision the agent obtains its last piece of additional information, i, which allows it to fix a states value where the probability of any particular value is given by the current future value distribution, $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}},i)$. The advantage of this assumption is that only one stochastic transition must be considered – the one caused by the information gathered immediately after this decision is made which resolves all the local search space's future values exactly. Hence, under this assumption the action selection MDP is only $\mathcal{O}((v_x)^n)$.

Fixing the value function for all future decisions also allows us to compute the value of the Bayes policy (i.e. greedy value maximisation given γ and the frontier value estimates) from any state recursively backward from the frontier states using, $V(x) = \max_u(r(x,u) + V(x))$, or forward using MINIMIN's approach,

$$V(x) = \max_{x_f \in \gamma_f} [r^*_\gamma(x, u_{x_f}) + \hat{V}(x_f)] = \max_{x_f \in \gamma_f} [\hat{V}(x, u_{x_f})], \tag{4.23}$$

where as before γ_f is the set of frontier states in local search space γ and u_{x_f} is an abstract action representing transition to frontier state x_f by the best possible route whose reward is $r^*_\gamma(x, u_{x_f})$. Hence, an improved value distribution can be computed for any state using,

$$\Pr(V(x)<v|\hat{\mathbf{V}}, i) = \Pr\left(\underset{x_f \in \gamma_f}{\text{AND}} \left[r^*_\gamma(x, u_{x_f}) + \hat{V}'(x_f) < v\right] \middle| \hat{\mathbf{V}}, i\right). \tag{4.24}$$

Notice, that only information about the distribution of the frontier states future value estimates is required for this computation, so it can be simplified by only predicting future values for these states.

4.8.1.3 The frontier value independence assumption

Equation (4.24) could be used directly to infer child value estimates by explicit enumeration over the space of possible frontier state estimated values. Unfortunately, this space is very large, $\mathcal{O}(v_x^{n_f})$, making actually performing this computation prohibitively expensive. Further, the design effort required in simply learning and storing the conditional probability table, $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, i)$, even when restricted to frontier states only, makes this direct implementation problematic.

To simplify this computation further additional assumptions about the structure of $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, i)$ are required. The structure of (4.24) itself suggests one simplifying assumption is to treat the frontier states values as independent. This assumption allows us to treat $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, i)$ as a factored model where $\prod_{x \in \gamma_f} \Pr(V(x)|\hat{\mathbf{V}}, i)$. This is the frontier value independence assumption and has two significant benefits. Firstly, it allows (4.24) to be rewritten as,

$$\Pr(V(x)<v|\hat{\mathbf{V}}, i) = \prod_{x_f \in \gamma_f} \Pr(r^*_\gamma(x, u_{x_f}) + \hat{V}'(x_f) < v|\hat{\mathbf{V}}, i). \tag{4.25}$$

Which, because the $\Pr(r^*_\gamma(x, u_{x_f}) + \hat{V}'(x_f) < v|\hat{\mathbf{V}}, i)$ can be computed independently and combined later reduces the computational effort to $\mathcal{O}(v_x n_f)$. Secondly, learning and predicting the distribution of future value estimates is much simplified as only individual states' value distributions are required (hence allowing the use of the state value estimate generating methods developed in Section 4.7).

Of course, it is unlikely that the frontier value independence assumption holds in practise, especially if the state space is a graph so nearby frontier nodes may have future trajectories

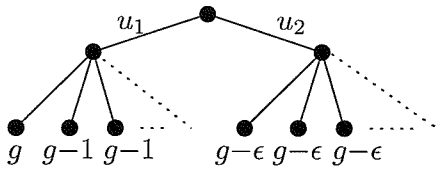


FIGURE 4.9: Assuming all frontier nodes have identical independently distributed future values drawn from $\Pr(V)$ this diagram shows the worst case situation for MINIMIN where it picks the action u_1 as it has the frontier node with the highest expected value whereas E{MRP} picks the alternative action which has n frontier nodes each of which is worse than g by ϵ.

in common. For example in a grid navigation problem states near the goal will be on the agent's future trajectory from many states, making their values highly dependent. One way of avoiding this independence assumption without increasing the inference complexity suggested by Russell and Wefald (1989, p. 148) uses the inequality $\Pr(A, B, \ldots) \leq \min[\Pr(A), \Pr(B), \ldots]$ to define the alternative approximation,

$$\Pr(V(x)<v|\hat{\mathbf{V}}, i) \leq \min_{x_f \in \gamma_f} \Pr(r^*_\gamma(x, u_{x_f}) + \hat{V}'(x_f) < v|\hat{\mathbf{V}}, i). \tag{4.26}$$

Unfortunately this approximation was found to give very poor performance so frontier value independence is assumed in the remainder of this thesis.

In either case the E{MRP} estimate for value of the agent's policy from state x is found by taking the average of x's updated value distribution. In the terms of the notation used in Section 3.3.7 this estimate is equivalent to computing a *myopic* value estimate for state x assuming a policy consisting of a computation, c_i, which generates the information, i, immediately followed by executing the Bayes action w.r.t. the resulting value estimates,

$$\text{E\{MRP\}} = \hat{V}^i_{c_i, c_\alpha}(x) = \text{E}\left\{V(x)|\hat{\mathbf{V}}, i\right\} \tag{4.27}$$

4.8.2 Analysis

The last incremental decision and frontier value independence assumptions represent a pair of strong assumptions which in combination enable the efficient inference of state value estimates which take some account of the predicted effects of later information gathering on the operation of an incremental search agent. Whether these assumptions are justified and whether the additional informations effects accounted for are significant enough to justify the computational complexity are two of the questions which the empirical analysis at the end of this chapter attempts to answer.

Pemberton (1995, Ch. 5) provides an analysis which shows that the benefits of the E{MRP} approximation compared to the simpler MINIMIN estimator are severely limited. This analysis proceeds from the worst case situation for MINIMIN shown in Figure 4.9. In this case the expected value of the MINIMIN choice, E{MM}, is simply $\text{E\{MM\}} = g + \text{E}\{V\}$. As the frontier values are all independent identically distributed (i.i.d.) the combined CDF for the frontier nodes

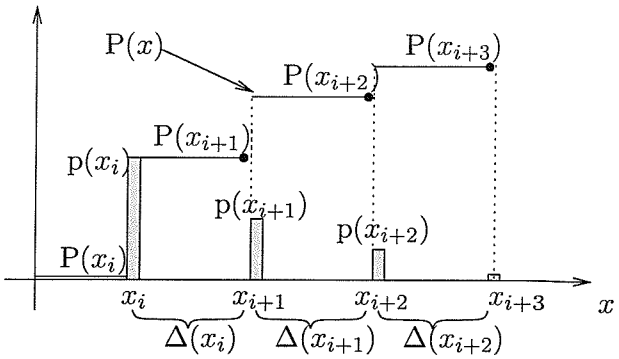


FIGURE 4.10: The staircase representation of a probability density function, $\mathrm{p}(x_i)$, and its cumulative density function, $\mathrm{P}(x_i)$.

below u_2 is given by, $\prod_i \Pr(V_i{<}y) = (\Pr(V{<}y))^n$ so E{MRP}'s expected value is given by,

$$\mathrm{E}\{\mathrm{MRP}\} = g - \epsilon + \int_{-\infty}^{\infty} y \frac{d(\Pr(V{<}y))^n}{dy} dy \tag{4.28}$$

Now, $\lim_{n\to\infty}(\Pr(V{<}y))^n \rightarrow \mathcal{H}(y{-}y^+)$ where y^+ is an upper-bound on V and $\mathcal{H}$ is the Heaviside step function, so in the limit E{MRP}'s advantage is E{MRP}-E{MM}$=-\epsilon + y^+ -$ E$\{V\}$. Results from the field of extreme value statistics indicate that whilst the exact rate at which this limit is reached with increasing n depends on the exact shape of $\mathrm{p}_V(y)$, in general it is reached more rapidly for less peaked, heavier tailed distributions. Thus E{MRP}'s advantage depends strongly on having few good nodes in the MINIMIN sub-tree and many good but sub-optimal frontier nodes in the E{MRP} sub-tree (so n is large). Of course, this situation is unlikely to occur in practise so in most cases E{MRP}'s advantage would be much lower. This analysis also shows that E{MRP}'s advantage reduces as the state value estimate's accuracy improves, i.e. becomes more peaked.

4.8.2.1 Implementation issues

To implement the E{MRP} estimator we require a representation for the probability distributions which makes the implementation of the probability product of (4.25) as simple as possible. Baum and Smith (1997) suggest this is simplest if the CDF has the piece-wise constant "staircase" shape shown in Figure 4.10. This implies that the PDF is a discrete set of impulses, or spikes. A discrete V automatically produces such a staircase distribution and one can be constructed as an approximation for any continuous V. In this way the product can be implemented using a merge sort style algorithm in $\mathcal{O}(n_s n_f \ln(n_f))$ time, where n_f is the number of frontier nodes and n_s is the number of spikes per frontier distribution. If necessary the cost of inference can be controlled by approximating the P_V with a distribution with less spikes, by for example using the algorithms in (Smith 1992).

The local state future value distributions, $\Pr(V'(x)|\hat{\mathbf{V}}, i)$, depend on the additional information, i, which must be estimated from γ. Hence, $\Pr(V'(x)|\hat{\mathbf{V}}, i) \approx \Pr(V'(x)|\gamma)$. As in the GSE

and BPS cases this CPT is too large to learn or store so we approximate it by assuming that the amount of information is independent of the node's position in γ. Hence, a single CPT can be used for all x's, which can be learnt by one of the local methods described in Section 4.7.

Correctly computing E{MRP} requires $\mathcal{O}(|\mathcal{U}|n_s n_f \ln(n_f))$ effort to compute the improved global value estimates for the roots children and $\mathcal{O}(|\gamma|)$ effort to compute the $r^*_\gamma(x_0, x_f)$. By considering only the frontier nodes in each child's sub-tree of the roots MRP tree, as was suggested for MINIMIN, this effort can be reduced to $\mathcal{O}(n_s n_f \ln(n_f) + |\gamma|)$, at the expense of more pessimistic estimates.

A further efficiency improvement suggested by Pemberton (1995) is based on the observation that (4.25) effectively filters out the effects of any distribution below the point where the *best* few nodes CDFs are near zero, i.e. their lower bounds. Thus, an effective approximation is to use only the k *best* frontier nodes distributions, reducing inference effort to $\mathcal{O}(n_s k \ln(k))$. This value estimator, called k-E{MRP}, also has the benefits of weakening the independence assumptions upon which the E{MRP} estimate relies as only the k best nodes need be independent and enabling the use of branch and bound techniques to find these best k frontier nodes. However, decision making performance may reduce as only k frontier nodes are considered.

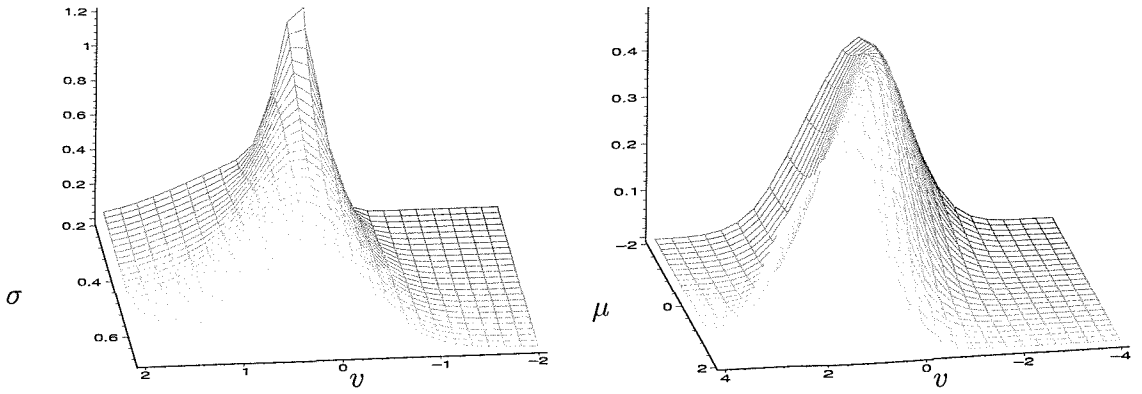
4.8.3 The approximate E{MRP} estimator

Even with all the approximations and simplifications described above at $\mathcal{O}(n_s n_f \ln(n_f) + |\gamma|)$ time complexity the E{MRP} estimator may be too expensive to use. The time cost of propagation is particularly important if we wish to use the estimates as part of the meta-control procedure to decide how to best extend γ as updated E{MRP} estimates will be required after each change to γ. This section presents the development of an new approximation, called $\sim$E{MRP}, which uses parameterised distributions to reduce the cost of computing improved value estimates to $\mathcal{O}(n_f + |\gamma|)$ or a very low constant overhead per node expansion.

4.8.3.1 The normal approximation

The key to this approximation is to replace the arbitrary distributions used in the E{MRP} estimator by parameterised distributions which can be propagated through (4.25) analytically. To minimise the error introduced by this approximation the parameterised distribution should approximate the true distribution, $\Pr(V'(x)|\gamma)$, with as little error as possible *and* propagate through (4.25) with little additional error.

It was decided to use a Gaussian for the parameterised distribution. This choice can be justified intuitively by appeal to the central limit theorem. If we assume the distributions parameters capture the important features upon which the true value depends, then the changes introduced by additional information can be thought of as small additional random fluctuations. If the



(a) (b)

FIGURE 4.11: Graph of $p_{\min(V_1,V_2)}$ for V_1, V_2 drawn from $\mathcal{N}_{0,1}$ and $\mathcal{N}_{\mu,\sigma}$ normal distributions. In (a) $\mu = 0$ and σ varies, in (b) $\sigma = 1$ and μ varies.

estimator is good and there are a large number of these fluctuations then by the central limit theorem the net effect will be to give a normal distribution. As shown in Figure 4.12 and Figure 4.13 empirically derived state local future value distributions have a strongly Gaussian feel, bearing out this intuition.

Figure 4.11 shows the result of propagating a pair of normal distributions through (4.25). As this diagram shows the result of propagating a pair of normal distributions only deviates significantly from normality when the distributions have very different standard deviations. Further, as Figure 4.11(b) demonstrates the filtering effect of (4.25) means that only the *better* distribution has any effect when they are more than a few standard deviations apart. Therefore, we would expect that in most circumstances the output of propagating a pair of normal distributions can be well approximated by another normal distribution. This is an important property as it significantly simplifies the use of our approximate propagation routine by allowing additional distributions to be incrementally included as they are encountered.

4.8.3.2 Derivation of the propagation equations

In this section the detailed mathematical derivation of an approximation for (4.25) for normal distributions is presented. We begin by defining the basic functions we shall be using and some of their useful properties.

The standard normal distribution, $\mathcal{N}(x)$, and its CDF, $\Phi(x)$, are defined as,

$$\mathcal{N}(x) \triangleq \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \qquad \Phi(x) \triangleq \int_{-\infty}^{x} \mathcal{N}(t)dt = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt \tag{4.29}$$

Using these definitions the normal distribution with mean μ and standard deviation σ and its associated CDF are given by,

$$\mathcal{N}_{\mu,\sigma}(x) = \frac{e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}}{\sqrt{2\pi}\sigma} = \frac{1}{\sigma}\mathcal{N}\left(\frac{x-\mu}{\sigma}\right), \qquad \int_{-\infty}^{x} \mathcal{N}_{\mu,\sigma}(t)dt = \Phi\left(\frac{x-\mu}{\sigma}\right). \quad (4.30)$$

To approximate (4.25) assume each of the the frontier states, $x_i \in \gamma_f$, future value estimates are normally distributed with known mean and standard deviation, so $\Pr(V(x_i){=}v) = \mathcal{N}_{\mu_i,\sigma_i}(v)$. The improved global value distribution for some interior state, x, is given by substituting these distributions into (4.25). Noting that when $V(x)$ is normally distributed then $\Pr(V(x){<}v) = \Phi(v)$ this gives,

$$\Pr(V(x){<}v) = \prod_i \Pr(V(x_i){<}v) = \prod_i \Phi\left(\frac{v-\mu_i}{\sigma_i}\right). \quad (4.31)$$

Initially, assume γ has only two frontier states, x_0 and x_1 then x's PDF is given by,

$$\mathrm{p}_V(v) = \frac{d\prod_{i=0,1}\Phi\left(\frac{v-\mu_i}{\sigma_i}\right)}{dv} = \mathcal{N}_{\mu_0,\sigma_0}(v)\Phi\left(\frac{v-\mu_1}{\sigma_1}\right) + \mathcal{N}_{\mu_1,\sigma_1}(v)\Phi\left(\frac{v-\mu_1}{\sigma_1}\right). \quad (4.32)$$

In order to find the properties of this distribution, i.e. its mean and standard deviation, we use its moment generating function, which is defined by $M(z) \triangleq \mathrm{E}\left\{e^{zX}\right\} = \int_{-\infty}^{\infty} e^{zx}\,\mathrm{p}_X(x)dz$ and has the useful property that $\mathrm{E}\left\{X^n\right\} = \frac{d^n M(0)}{dx^n}$, i.e. the nth moment of X is given by the nth derivative of $M(z)$ evaluated at 0. For $\mathrm{p}_V(v)$ the moment generating function is given by,

$$M(z) = \int_{-\infty}^{\infty} e^{zv}\left[\frac{1}{\sigma_0}\mathcal{N}\left(\frac{v-\mu_0}{\sigma_0}\right)\Phi\left(\frac{v-\mu_1}{\sigma_1}\right) + \frac{1}{\sigma_1}\mathcal{N}\left(\frac{v-\mu_1}{\sigma_1}\right)\Phi\left(\frac{v-\mu_0}{\sigma_0}\right)\right]dz. \quad (4.33)$$

Bringing the exponential inside and combining it with the normal distribution gives,

$$\begin{aligned} M(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} &\frac{1}{\sigma_0}\exp\left(zv - \frac{1}{2}\left(\frac{v-\mu_0}{\sigma_0}\right)^2\right)\Phi\left(\frac{v-\mu_1}{\sigma_1}\right) \\ &+ \frac{1}{\sigma_1}\exp\left(zv - \frac{1}{2}\left(\frac{v-\mu_1}{\sigma_1}\right)^2\right)\Phi\left(\frac{v-\mu_0}{\sigma_0}\right)dz. \end{aligned} \quad (4.34)$$

Denote the integral of the first additive term as $I_{0,1}$ and the second as $I_{1,0}$. The two integrals have symmetric terms so consider $I_{0,1}$ first. Completing the square gives,

$$I_{0,1} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\frac{1}{\sigma_0}\exp\left(zv - \frac{1}{2\sigma_0^2}(v^2 - 2v\mu_0 + \mu_0^2)\right)\Phi\left(\frac{v-\mu_1}{\sigma_1}\right)dz, \quad (4.35)$$

$$= \exp\left(\mu_0 z + \frac{\sigma_0^2 z^2}{2}\right)\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma_0}\exp\left[-\frac{1}{2}\left(\frac{v-\sigma_0^2 z+\mu_0}{\sigma_0}\right)^2\right]\Phi\left(\frac{v-\mu_1}{\sigma_1}\right)dz. \quad (4.36)$$

Using the substitution $w = \frac{x-\sigma_0^2 z+\mu_0}{\sigma_0}$ we obtain,

$$I_{0,1} = \exp\left(\mu_0 z + \frac{\sigma_0^2 z^2}{2}\right) \int_{-\infty}^{\infty} \frac{e^{-\frac{1}{2}u^2}}{\sqrt{2\pi}} \Phi\left(\frac{w\sigma_0 + \sigma_0^2 + \mu_0 + \mu_1}{\sigma_1}\right) dz. \quad (4.37)$$

To perform this integration we require the following integral identity,

$$\int_{-\infty}^{\infty}\int_{-\infty}^{ax+b} f(x)g(y)dydx = \int_{-\infty}^{\infty}\int_{-\infty}^{\frac{b}{\sqrt{1+a^2}}} f\left(\frac{x'+ay'}{\sqrt{1+a^2}}\right) g\left(\frac{y'-ax'}{\sqrt{1+a^2}}\right) dy'dx', \quad (4.38)$$

where $x' = \frac{x-ay}{\sqrt{1+a^2}}$ and $y' = \frac{ax+y}{\sqrt{1+a^2}}$. This identity follows because the substitutions are equivalent to rotating the integral axes parallel to $ax + b$ where the upper bound is constant. Using this identity the following useful definite integral can be derived,

$$\begin{aligned}
\int_{-\infty}^{\infty} \mathcal{N}(x)\Phi(ax+b)dx &= \int_{-\infty}^{\infty}\int_{-\infty}^{ax+b} \frac{1}{2\pi} e^{-\frac{1}{2}x^2} e^{-\frac{1}{2}y^2} dydx, \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\frac{b}{\sqrt{1+a^2}}} \frac{1}{2\pi} e^{-\frac{1}{2}\left(\frac{x'+ay'}{\sqrt{1+a^2}}\right)^2 - \frac{1}{2}\left(\frac{x'+ay'}{\sqrt{1+a^2}}\right)^2} dy'dx', \\
&= \int_{-\infty}^{\infty} \frac{e^{-\frac{1}{2}x'^2}}{\sqrt{2\pi}} dx' \int_{-\infty}^{\frac{b}{\sqrt{1+a^2}}} \frac{e^{-\frac{1}{2}y'^2}}{\sqrt{2\pi}} dy' = \Phi\left(\frac{b}{\sqrt{1+a^2}}\right), \quad (4.39)
\end{aligned}$$

where the last step follows because $\int_{-\infty}^{\infty} \mathcal{N}(x)dx = 1$. Using this identity (4.37) becomes,

$$I_{0,1} = \exp\left(\mu_0 z + \frac{\sigma_0^2 z^2}{2}\right) \Phi\left(\frac{\frac{\sigma_0^2 z+\mu_0-\mu_1}{\sigma_1}}{\sqrt{1+(\sigma_0/\sigma_1)^2}}\right), = \exp\left(\mu_0 z + \frac{\sigma_0^2 z^2}{2}\right) \Phi\left(\frac{\sigma_0^2 z+\mu_0-\mu_1}{\sqrt{\sigma_0^2+\sigma_1^2}}\right). \quad (4.40)$$

Substituting this and its symmetrical partner for $I_{1,0}$ into (4.34) we obtain the final definition of p_V's generating function,

$$M(z) = \exp\left(\mu_0 z + \frac{\sigma_0^2 z^2}{2}\right) \Phi\left(\frac{\sigma_0^2 z+\mu_0-\mu_1}{\sqrt{\sigma_0^2+\sigma_1^2}}\right) + \exp\left(\mu_1 z + \frac{\sigma_1^2 z^2}{2}\right) \Phi\left(\frac{\sigma_1^2 z+\mu_1-\mu_0}{\sqrt{\sigma_0^2+\sigma_1^2}}\right). \quad (4.41)$$

Using this definition after some mathematical effort we find p_V's first and second moments simplify to,

$$\dot{M}(0) = \mu_1 + (\mu_0 - \mu 1)\Phi\left(\frac{\mu_0-\mu_1}{\sqrt{\sigma_0^2+\sigma_1^2}}\right) + \frac{\sqrt{\sigma_0^2+\sigma_1^2}}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\mu_0-\mu_1}{\sqrt{\sigma_0^2+\sigma_1^2}}\right)^2}, \quad (4.42)$$

$$\begin{aligned}
\ddot{M}(0) = {} & \sigma_1^2 + \mu_1^2 + (\sigma_0^2 - \sigma_1^2 + \mu_0^2 - \mu_1^2)\Phi\left(\frac{\mu_0-\mu_1}{\sqrt{\sigma_0^2+\sigma_1^2}}\right) \\
& + \frac{e^{-\frac{1}{2}\left(\frac{\mu_0-\mu_1}{\sqrt{\sigma_0^2+\sigma_1^2}}\right)^2}}{\sqrt{2\pi}\sqrt{\sigma_0^2+\sigma_1^2}} \left[2(\mu_0\sigma_0^2 + \mu_1\sigma_1^2) - \left(\frac{(\mu_0-\mu_1)(\sigma_0^4-\sigma_1^4)}{\sigma_0^2+\sigma_1^2}\right)\right]. \quad (4.43)
\end{aligned}$$

Using these equations p_V's updated mean and variance can be computed in the usual way as

$\mu = \dot{M}(0)$, $\sigma^2 = \ddot{M}(0) - \mu^2$. Hence, as the best (in terms of minimising Kullback-Leiber divergence) normal approximation for any distribution has the distributions mean and variance, we can optimally approximate the propagation of two normal distributions to some interior node with another normal distribution given by $\mathrm{p}_V(v) \approx \mathcal{N}_{\mu,\sigma}(v)$. This is the constant time normal distribution propagation method we have been seeking.

4.8.4 Implementation issues

By extracting common terms (4.42) and (4.43) can be evaluated in constant time, requiring only two function evaluations (an exp and a Φ) and a few arithmetic operations. Using lookup tables for the function evaluations this allows us to reduce the propagation overhead to tens of machine operations. Further, the product of (4.25) can be decomposed into $n_f - 1$ binary products. Recursively applying the normal distribution propagation procedure with the same decomposition allows us to compute the combined effect of all n_f frontier states in linear, $\mathcal{O}(n_f)$, time.

If necessary, the computational effort can be further reduced as for the E{MRP} estimator by only considering frontier states in each child's sub-tree and/or only propagating the k best frontier states values, to give the $\sim k$-E{MRP} estimator.

4.9 Experimental analysis

This chapter has presented five main methods for implementing the decision making component of an on-line planning agent; MINIMIN, GSE, BPS, E{MRP}, $\sim$E{MRP}. Except for GSE all these methods make significant simplifying assumptions to increase their efficiency. This section empirically compares these methods to determine if the assumptions are justified and assess the relative merits of each approach with regard solely to decision making performance.

4.9.1 General methodology

To ensure the comparisons are as fair as possible, in this and all subsequent experimental analysis, we make what Hansson (1998) has called the "opaque state bias" assumption. Under this assumption decision making methods are not allowed direct access to the current state and decision problem but only the indirect information provided by the feature detectors contained in the labelled local search graph. This assumption prevents the use of non-scalable methods which condition on the true current state and enforces a consistent information regime to allow easy comparison of different algorithms. Notice that only the *problem specific* information is restricted. The decision making system can utilise any amount of domain specific information, such as learnt correlations between feature vectors and true values, to improve its performance.

Indeed, one of the aims of these experiments is to assess what type of domain specific information is most useful.

In the experiments reported in this thesis the available features are;

1. the state connectivity for non-frontier states of γ,
2. the transition rewards, $r(x, u)$, for transitions within γ, and
3. a domain specific heuristic estimate of the value of the optimal trajectory from the state, $h(x)$. For the two experimental domains considered here the heuristics are,
   - **N-Puzzle** – the Manhattan distance heuristic (Doran and Michie 1966), this is the sum over all tiles of the Manhattan distance between the tiles current and goal locations.
   - **Grid** – the Manhattan free space heuristic, this is the Manhattan distance between the state and the *nearest* goal.

In this chapter to focus solely on decision making performance all experiments will be conducted using the same set of labelled search graphs.

4.9.2 Training

Except for MINIMIN, all the decision making systems described in this chapter require some off-line training to learn a local feature set to value estimate (distribution) mapping function. This section describes the training methodology used to perform this learning. Due to the size of the problem spaces the agent will work with the basic approach taken in this work is to build a training set by randomly generating a representative set of example problems, computing the feature values and their associated true values, and then learning from this set.

Clearly, to be useful the training set should be representative of the distribution of problems the agent will need to solve. This distribution depends on the agent's actual decision making abilities as the problems an incremental search agent encounters later in its "life" depend on its earlier decisions (and recursively on the training itself). Of course, at training time the exact agent implementation is unknown. So a hopefully realistic generic incremental search agent is used to generate training examples. Specifically, the training set is generated in the following way:

1. Generate a start state generated at uniform random from the domains problem space.
2. Run a RTA* style incremental search[3] from this state for a limited number of steps, recording a random sample ($\approx$1%) of the states expanded. These are the training examples.

[3]This uses fixed depth α-pruned γ construction and MINIMIN decision making with the heuristic values as frontier state value estimates.

3. Post-process the training example states to compute the (near) *optimal* values for the states children, $V^*(x,u)$, using a time limited IDA* search (Korf 1985a).

The result of this training process is a set of training examples consisting of sampled states and their associated children's estimated optimal value, $\{V^*(x,u_1), V^*(x,u_2)\ldots\}$. Notice, that this training regime computes the optimal value from each state as an approximation for the agent specific value. As mentioned in Section 4.1.2 this may be over-optimistic causing a reduction in agent performance. This is especially true for the E{MRP} estimators where using the distribution of V^* to represent the distribution of new values is equivalent to assuming that after this decision the agent gets *perfect information* about the frontier states true values.

4.9.2.1 Results

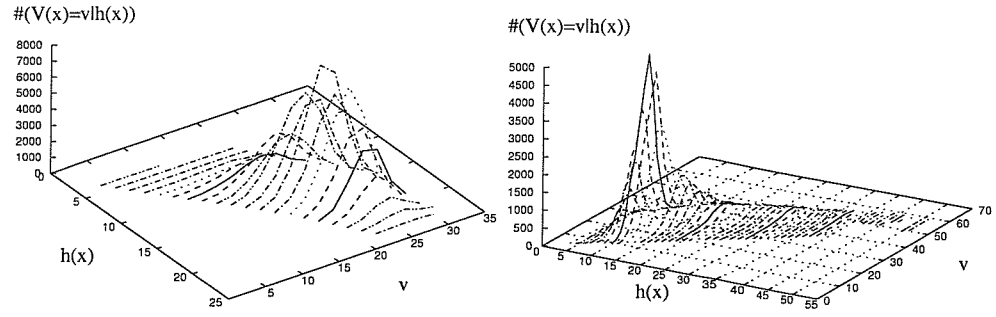
The results of training the state local value estimator based on the state's heuristic value for the 4 problem domains considered are presented in Figure 4.12 for the puzzle domains, and Figure 4.13 for the grid domains. Examining these distributions we note that;

- In all the domains the distributions have Gaussian shape with a single central peak and relatively symmetric tails. This justifies to some extent the normal approximation used in the ~E{MRP} estimator.
- In all the domains the heuristic values are quite poor predictors of the true value, as seen by the wide spread of the true value distributions. The average variance is ≈10 for the grid problems and ≈4 for the puzzle problems. These are both more than twice the average transition costs of 5 and 1 respectively.
- As one would expect in all cases the heuristics are significantly biased and underestimate the true goal distance, as seen by the fact that the heuristic value is always at or below the lower bound of the true value distribution. This is most significant in the grid domains where the heuristics underestimate the mean true value by a factor of ≈5.

The poor quality of the heuristics as future value estimators proves the necessity for value estimate techniques to improve the decision making quality. The significant bias of the heuristic estimates also suggests that MINIMIN's decision making could be significantly improved by simply calibrating its frontier future value estimates to remove this bias. In later experiments we test this hypothesis using the MM-CAL hybrid value estimator.

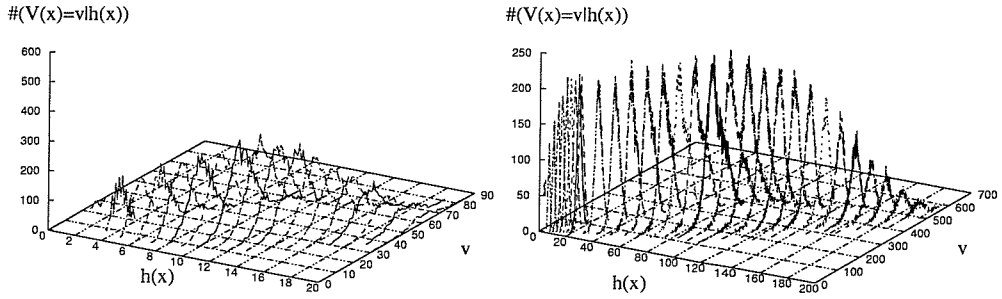
4.9.3 Independence tests

The E{MRP} estimators make strong assumptions about the independence of frontier states future values. The validity of these assumptions was tested directly by computing the correlation coefficient between pairs of frontier states true values. The procedure used was;



(a) 8-Puzzle (b) 15-Puzzle

FIGURE 4.12: Counts of the true value of states with given heuristic value, for (a) the 8-Puzzle and (b) 15-Puzzle. Sample states generated using complete enumeration for the 8-puzzle, and a depth 10 RTA* search from 6,000 uniform sampled start states recording 0.5% of expanded states for the 15-puzzle.



(a) 10x10 Grid (b) 100x100 Grid

FIGURE 4.13: Counts of the true value of states with given heuristic value, for (a) the 10x10 grid, and (b) 100x100 grid, both with action costs selected uniform randomly from the range [1,10]. Sample states generated using complete enumeration for 50,000 uniform randomly sampled problems for 10x10 grid and a depth 10 RTA* search from 50,000 uniform randomly sampled start states for the 100x100 grid with 10% of expanded states recorded for training.

(1) generate a problem sampled at uniform random from the domain problem space,

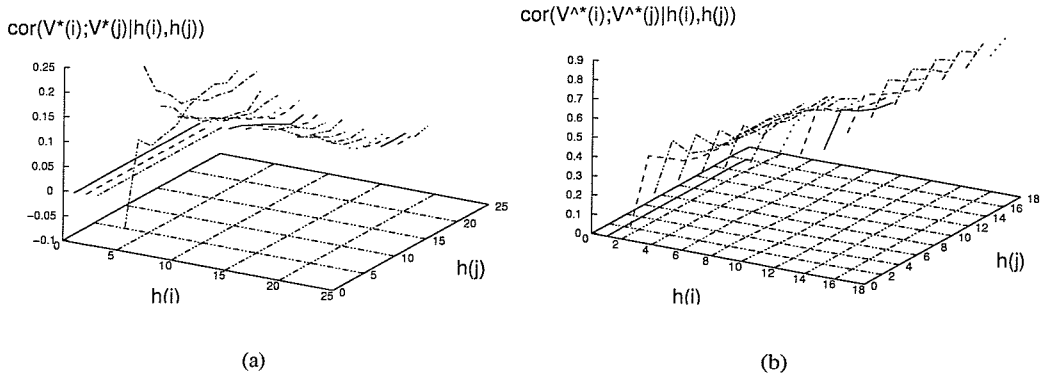
(2) compute the near optimal value of all states in the problem using time limited IDA*(Korf 1985a),

(3) choose at uniform random a start state,

(4) generate a fixed depth local search space for each depth up to depth 10,

(5) for each pair of frontier states at each depth record their optimal values, binned according to the state's heuristic values.

The correlation coefficients were then calculated from the stored information. As this procedure requires the computation of the optimal value for all states in the problem space it could only be conducted for the smaller 8-puzzle and 10x10 grid problems. The results of this analysis,



(a) (b)

FIGURE 4.14: These graphs plot the correlation coefficients between pairs of frontier states true values, where the state pairs have been binned according to their heuristic values. Plot (a) gives the results for the 8-puzzle and (b) the 10x10 grid.

presented in Figure 4.14, clearly show that for the 8-puzzle the frontier state correlations are quite low, $\leq$0.25, hence in this domain the independence assumption is not too unrealistic. They also show that frontier states with similar low heuristic values tend to be better correlated. This is what we would expect if similar paths are taken by nearby states in proximity to the goal.

In contrast, Figure 4.14(b) shows that for the 10x10 grid frontier states correlations can be quite high, being almost perfectly correlated for large heuristic values. This is actually an artifact of the domain's limited size and top right goal location which means there are very few states with large heuristic values, specifically 1 with heuristic value 19, 2 with heuristic value 18, etc. Thus, results for heuristic values above $\approx$12 should be discarded to remove this artifact. However, the high correlations, $\geq$ 0.3, of the remaining states indicate that the independence assumption may be unrealistic in this case.

4.9.4 Testing the decision making quality

This section presents an experimental comparison of the different decision making methods performance profiles, which gives the decision quality as a function of the local search space size. The criteria chosen for assessing decision quality is the expected move error, which for a set of starting states $x \in \mathcal{X}_s$ is defined as $\sum_{x \in \mathcal{X}_s} \Pr(x)(V^*(x) - V^*(\alpha)) = \mathrm{E}\{V^*(x) - V^*(x, \alpha)\}$, where α is the action chosen by the decision making method. For explicitly rational agents α is given by $\alpha = \mathrm{argmax}_u \hat{V}(u|\gamma)$. In words the expected error is the expected loss in reward from using this decision making method for this decision then following an optimal trajectory, rather than just following an optimal trajectory directly. Notice, that this measure is concerned only with decision making quality *not* value estimate quality. This criterion was chosen as it directly measures the quality of the decision making systems most important output—its decisions. It also takes account of the importance of the decision as highly sub-optimal decisions weigh more highly than slightly sub-optimal ones. This is important in the grid domain where the loss from sub-optimal decisions can range from 0 to 18.

As the shape of the local search space can be critical to decision making performance two sets of experiments were conducted with the local search space was constructed using either full-width fixed depth search or A* search (Pearl 1984). The full-width fixed depth tests allow for simple analysis and comparison with the systems for which only full-width results are available, namely the GSE, GSE-F and BPS systems. The A* local search spaces should be more representative of the types of search space constructed by a meta-control system, and hence give a better indication of true decision making performance.

All the experiments in this section were conducted using the following procedure:

1. Generate a problem and initial state sampled at uniform random from the domain problem space.
2. Generate local search spaces of increasing size or depth up to some bound using either the fixed-depth or A* generation procedures.
3. Run the decision making system on the local search space, recording the action it would have chosen and other summary information.
4. Compute the optimal solution and its value for each child of the initial state.
5. Use the computed optimal values to compute and record the decision making systems expected move error.

All algorithms (except BPS(2)) were implemented and tested within a single generic C++ incremental search framework using the same graph construction, heuristic evaluation, and performance assessment code to minimise implementation specific artifacts. The algorithms tested were;

Minimin MINIMIN using heuristic values for frontier state values,[4] and only considering frontier states in a states MRP sub-tree for computing the states value.

MM-Cal (1-E{MRP}) MINIMIN with calibrated frontier state values, $\mathrm{E}\{V(x)|h(x)\}$.

GSE The gold standard estimator.

GSE-F The gold standard estimator, using only frontier states r^* and $h(x)$ information.

E{MRP} E{MRP} using state local value estimators, $\Pr(V(x)|h(x))$, and the MINIMIN efficiency hack of only considering frontier states in a states MRP sub-tree.

~E{MRP} The ~E{MRP} version of E{MRP}.

k-E{MRP} E{MRP} propagating only the k frontier states with best expected value. (Note, only results for 5-E{MRP} are presented.)

BPS(2) Mayer's (1994) published BPS(2) results. Only available for the 8-puzzle.

[4]Note, this is an *irrational* decision making method as the heuristic values are strongly biased estimators of the future value of a frontier node.

4.9.4.1 Fixed depth local search spaces

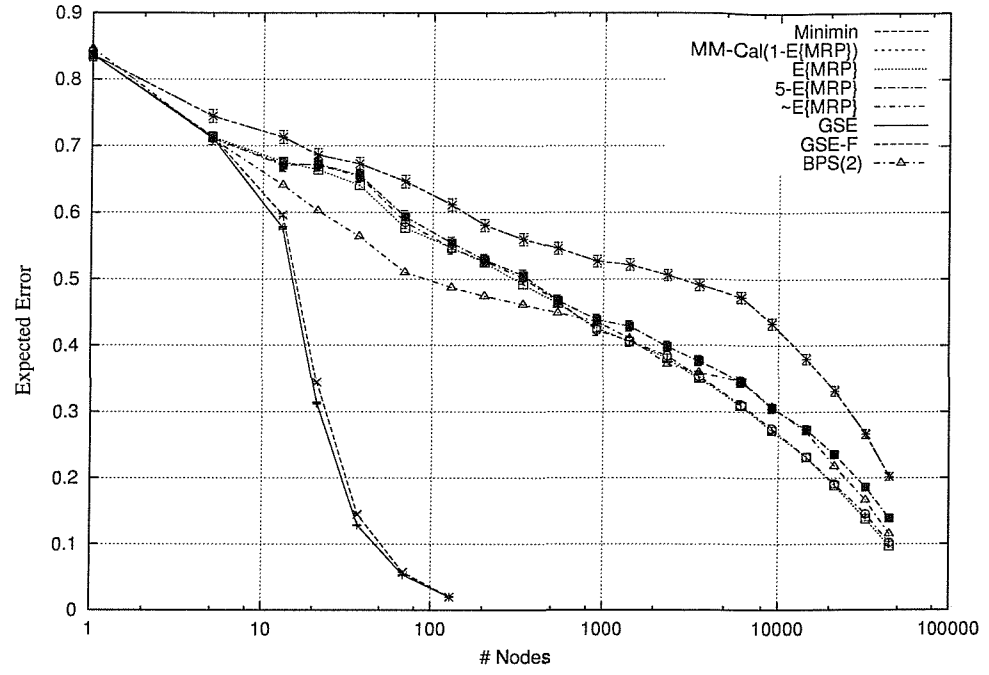
The results of the fixed depth experiments are presented in Figure 4.15 for the puzzle domains and Figure 4.16 for the grid domains. The notable features of these results are;

- The good performance of BPS, E{MRP}, 5-E{MRP} and ~E{MRP}, relative to MINIMIN.
- The high performance, particularly for low numbers of states of MM-CAL on the grid problems.
- The near indistinguishable performance of E{MRP} and ~E{MRP}.
- The extreme performance of the GSE and GSE-F algorithms, which, on the 8-puzzle, can equal that of any other algorithm with almost 1,000 times fewer states.
- The rapidly improving performance of all the algorithms for deep searches on the 8-puzzle and 10x10 grid.

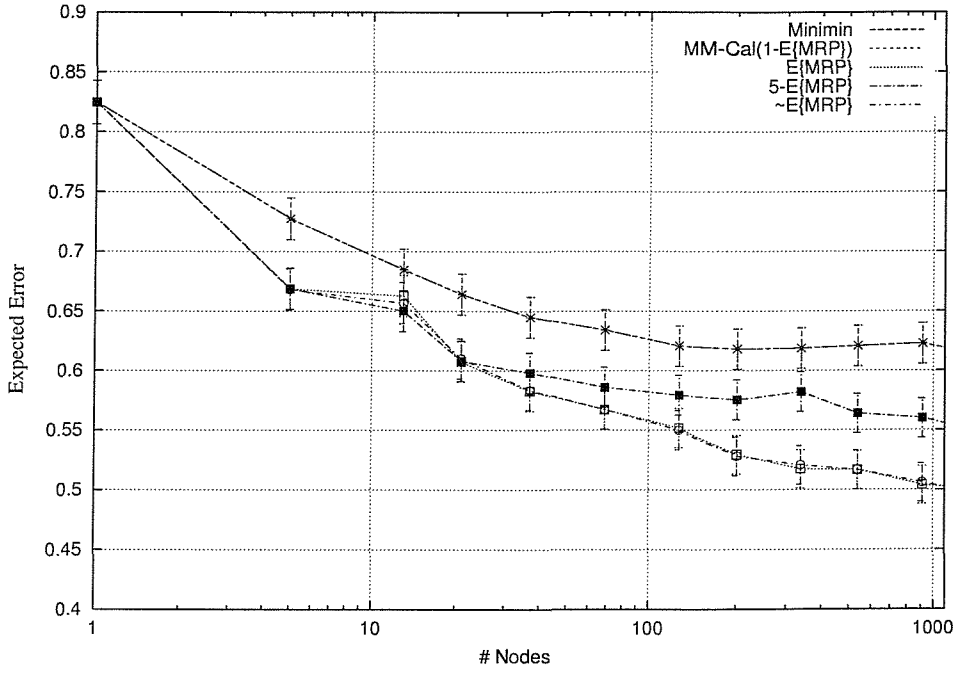
The most important result of these experiments is that it is possible to significantly exceed the decision making quality of a naive decision making method such as MINIMIN. The improvements are most pronounced on the puzzle problems (Figure 4.15), where BPS(2) offers the best advantages exhibiting a consistent 0.1 reduction in expected error for all search depths greater than 3 on the 8-puzzle. Viewed in the other orientation BPS(2) reaches an error of 0.5 at depth 6 for 90 states whereas MINIMIN reaches the same performance level at depth 13 for 2,000 states. Thus, BPS requires significantly less effort, over 20 times less in the best case, for the same level of performance.

Analysis of the E{MRP} algorithms. Figure 4.15(a) shows that the E{MRP} algorithms are less advantageous than BPS at lower depths, requiring an additional 2 or 3 levels of search to reach BPS performance levels for depths <10 or 1,000 states. However, at higher depths they perform as well as BPS, even slightly exceeding it for very deep searches. It is hypothesised that the reduced performance of E{MRP} for shallow depth searches arises because the E{MRP} algorithms only utilise information from frontier states when constructing their estimates. For shallow searches, when the frontier states value estimates are likely to be unreliable, BPS can use the interior states feature values to improve its final estimate quality, whereas E{MRP} estimates must rely solely on the frontier state information. As the search space gets larger the important, lowest value, frontier estimates will tend to get more accurate so ignoring interior states becomes less important. The decreasing deviation between the GSE and GSE-F algorithms for increasing search depth on the 8-puzzle bears out this hypothesis to some extent.

On the grid domains the advantages of the E{MRP} algorithms relative to MINIMIN are equally pronounced with consistent improvements of 0.4 in error or 7 levels of search on the 10x10 grid

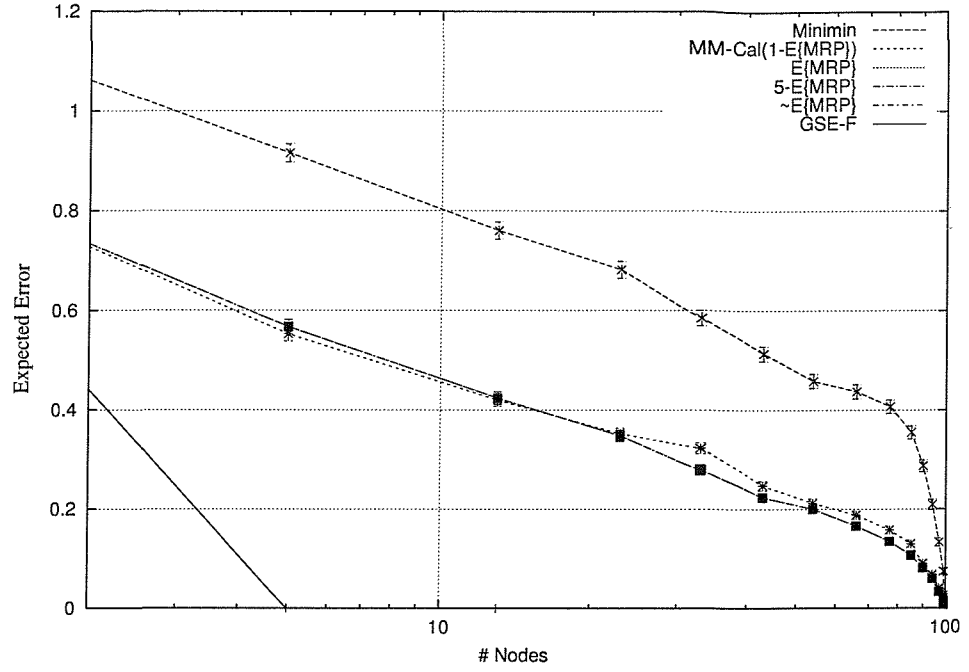


(a)

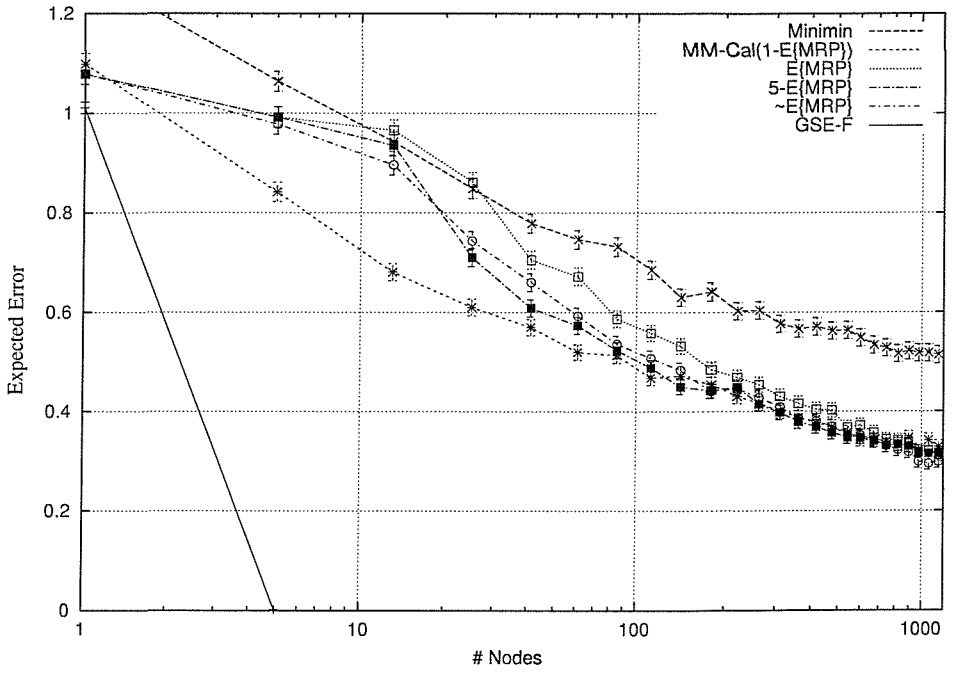


(b)

FIGURE 4.15: Decision making methods expected performance profiles for fixed depth search on the (a) 8-Puzzle and (b) 15-Puzzle. Each point represents a search depth. The BPS(2) results are taken from (Mayer 1994, Fig. 3-9). All other results are averages over 10,000 problems for the 8-puzzle and 3,000 problems for the 15-puzzle. The error bars indicate the mean's sample variance. Note the log-scale on the x-axis, and non-zero y-axis in (b).



(a)



(b)

FIGURE 4.16: Decision making methods expected performance profiles for fixed depth local search spaces on the (a) 10x10 grid and (b) 100x100 grid. Each point represents a search depth. The results are averages over 10,000 problems with error bars indicating the mean's sample variance. Note, the log-scale on the x-axis.

and more than *30 levels* of search (or about 7 times fewer states) on the 100x100 grid. This is inspite of the invalidity of the E{MRP} estimates due to the high correlations between frontier value estimates. However the fact that MM-CAL algorithm performs equally well or better on these problems indicates that the performance is due more to the frontier value estimate calibration than the probabilistic propagation. This is not too surprising as in these domains the heuristic estimates underestimate the true costs by about a factor of 5. Hence, MINIMIN puts excessive importance on the reward of the path taken to a node, allowing it to get mislead into preferring a decision which leads to a frontier node with good reward to it but poor reward from it. Calibration corrects this problem, hence improving MINIMIN's performance. This emphasises the importance of correct value estimate calibration for decision making performance. Note, this effect is not apparent in the puzzle domains as the heuristics estimates are only about 0.6 times the true cost so calibration will not generally change MINIMIN's decision.

The reduced performance of E{MRP} relative to MM-CAL and 5-E{MRP} on the 100x100 grid (Figure 4.16(b)) for low search depths, is assumed to be an indication of the break-down of the E{MRP}'s independence assumptions because performance appears to be *inversely* related to the degree of independence assumed.

The near indistinguishable performance of E{MRP} and ~E{MRP} indicate that, at least on these domains, the approximations introduced by ~E{MRP} do not significantly reduce performance. In fact, the superior performance of ~E{MRP} on the 100x100 grid (Figure 4.16(b)), at low search depths, indicates that these approximations can actually be beneficial. This is somewhat surprising given that ~E{MRP} approximates the frontier value functions with a Gaussian and uses an approximate propagation routine, both of which should introduce errors which would reduce the final estimate accuracy. To explain this superiority two possibilities suggest themselves: the first is that the normal approximation for the state value estimates makes the computed estimates less sensitive to the sampling noise in these distributions shown in Figure 4.13(b). The second possibility is that somehow the approximations reduce the effect of the independence assumptions, and hence indirectly improve performance in this domain. In either case the fact that ~E{MRP} does not significantly reduce performance is encouraging given the reduction in computational complexity it provides.

The performance of 5-E{MRP} is less encouraging as it performs worse than ~E{MRP} despite requiring significantly more computational effort.

Analysis of the GSE algorithms. The extremely high performance of the GSE algorithms indicates that there is a *lot* of information contained in the local search space not used by the other decision making algorithms. Further, the near identical performance of GSE-F indicates that most of this useful information is contained in the frontier states, (at least for larger search spaces). Unfortunately, the GSE is *not* the way to extract this information. Its tables required 8hr of training and 100Mb just for the fixed width local search spaces on the relatively small 8-puzzle. The frontier only table took 400Mb and 2 days of training to get a *partial* table for the

10x10 grid. The required the table sizes made it impossible to implement these algorithms for the larger problems.

The pitfalls of toy problems. The improving performance of all the algorithms for deep searches on the 8-puzzle and 10x10 grid should serve as a warning to be wary of using toy problems to assess performance. These performance improvements are due to the finite size of the problem space meaning the local search space has a high probability of encountering what (Mayer 1994) calls *beacon* states. These are states whose heuristic values are obvious signposts to the goal location. In the puzzle domains the beacons are states with heuristic values less than 5, as from these states greedy action selection is almost certain to optimally lead to the goal. In the 10x10 grid the effect is due simply to the local search space including a significant portion of the very small 100 state problem space, so any remaining un-explored paths are almost sure to lead to the goal.

4.9.4.2 A* generated, variable depth, local search spaces

When attempting to identify the best local decision it is very likely that improved information about the true value of some frontier states will be more important than others, improving decision quality more rapidly for less effort. An effective meta-controller will expand these states first, creating a variable depth local search space. The experiments in this section test the ability of the decision making methods to cope with such variable depth local search spaces. They have been conducted using the same methodology as above, except an A* search terminated after a given number of nodes have been expanded is used to construct the local search space. The assumption was that A* should produce search spaces similar to those produced by a good meta-controller.

For these experiments an additional decision making method called ANTI-THRASH has been introduced. As the name implies the aim of this algorithm is to overcome A*'s well known thrashing problem where the node A* expands next jumps seemingly at random over the set of frontier states. Usually, thrashing is seen as a problem for maintaining locality of reference for efficient memory caching in A* implementations. However, as A*'s selection criteria is identical to MINIMIN's, thrashing can also present a problem for MINIMIN decision making. The problem is that because the heuristic estimates are *optimistic* one node can appear better than another either because it is actually better or because it hasn't been expanded as deeply and is less well informed. This causes A* to have a risk seeking behaviour where it ignores deeper better informed frontier states if some less well-informed node that has some, no matter how small, chance of being optimal. For generating complete solutions this is fine as it guarantees the solution produced is optimal as all possibly better paths have already been explored and disproven. However, for local decision making the high probability of the MINIMIN node being misleading significantly reduces decision quality. ANTI-THRASH attempts to make decision

making less risk seeking by only changing the best node if the new MINIMIN frontier node is *deeper* in the search tree, and hence better informed.

The results of the A* experiments are presented in Figure 4.17 for the puzzle domains and Figure 4.18 for the grid domains. The most notable features of these results are;

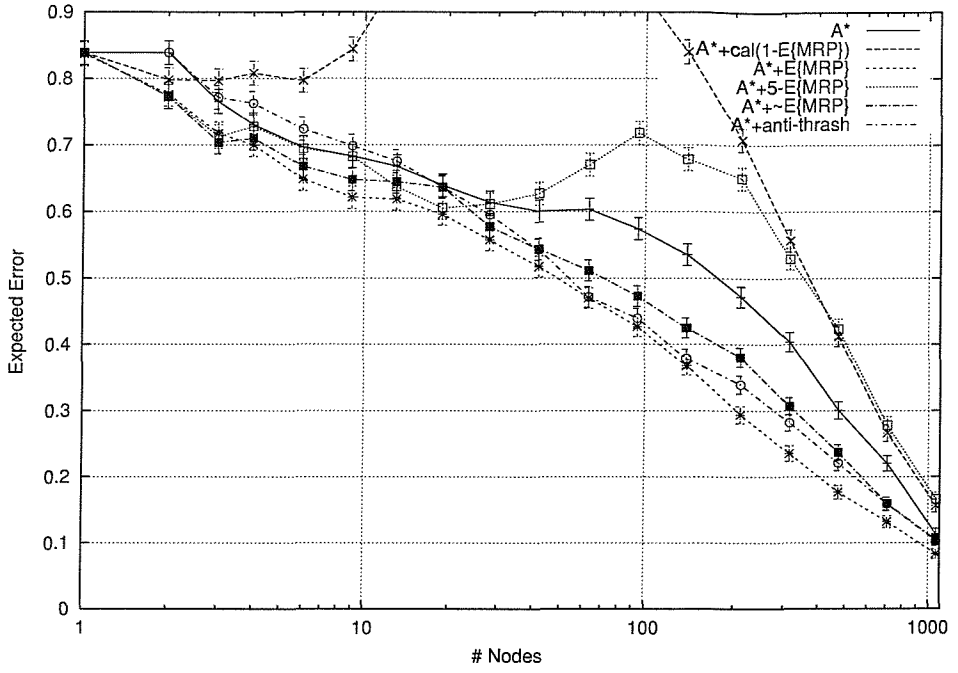
- The significant improvement in decision making performance compared to full-width search space construction for *all* the algorithms on the 8-puzzle ($\approx 1000\times$ fewer states), and more modest improvements on the 15-puzzle ($\approx 3\times$ fewer states).
- The reduction in performance compared to full-width search space construction for *all* the algorithms on the grid domains.
- The non-monotonic performance of MM-CAL and 5-E{MRP} on the puzzle problems and MM-CAL on the grid problems.
- The poor performance of A* in all domains compared to the very high performance of ANTI-THRASH on the 15-Puzzle and 100x100 grid for deep searches.

The most important result of this set of experiments is that E{MRP} and ~E{MRP} still perform consistently better than MINIMIN in all the domains, with an increased relative advantage on the 15-puzzle and grid domains. Thus, the E{MRP} algorithms are still making more effective use of the available information.

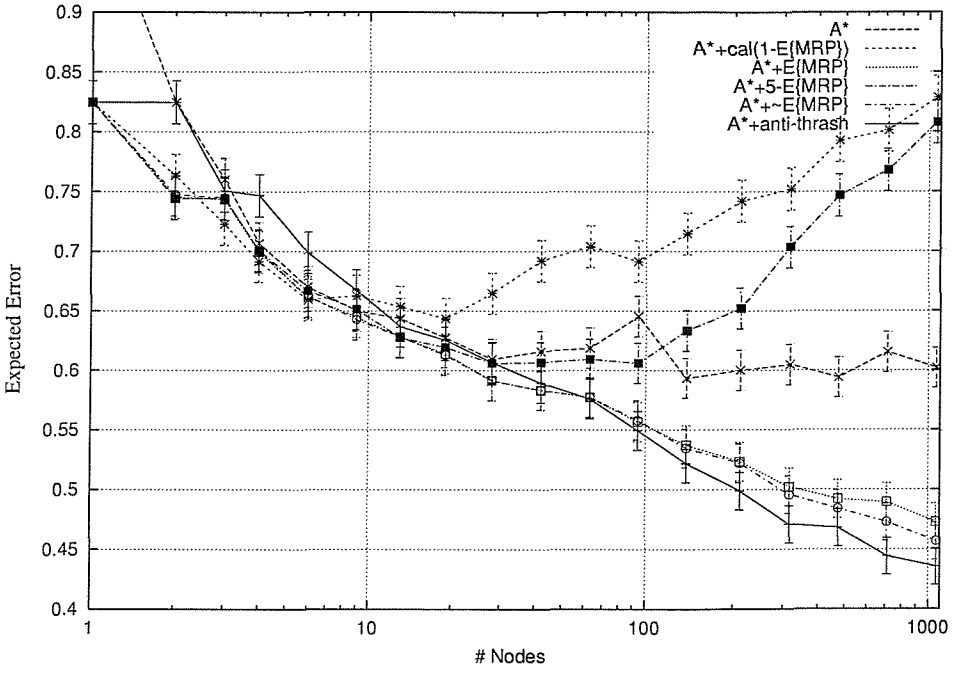
A*'s contradictory effects on puzzle and grid problems. On comparing these results with the fixed width results (Figure 4.15 and Figure 4.16) it is apparent that using a truncated A* constructed local search space has the contradictory effect of improving performance on the puzzle problems but reducing performance on the grid problems. This effect is attributed to the relative quality of the heuristics for guiding search space construction.

As noted earlier (Section 4.9.2.1) the heuristics are much better predictors of true value in the puzzle problems than the grid problems. Thus, the heuristics provide better guidance to A* for search space construction in the puzzle problems leading to improved search performance. The remarkable improvements in the 8-puzzle are likely due to the more accurate heuristics improving search space construction to such an extent that it significantly increases the chances of encountering a beacon state in the small state-space.

In the grid domains the opposite effect appears to hold, where the heuristics so massively underestimate the true value as to increase the chance of A* being mislead down blind alleys. For example, A* may prefer to expand a state with a low path cost, i.e. $r^*(x, x_f)$ the cost of getting from x to x_f, which leads away from the goal than one with a higher path cost which leads to the goal because the former's heuristic value underestimates the true cost of getting to the goal from its end-point.

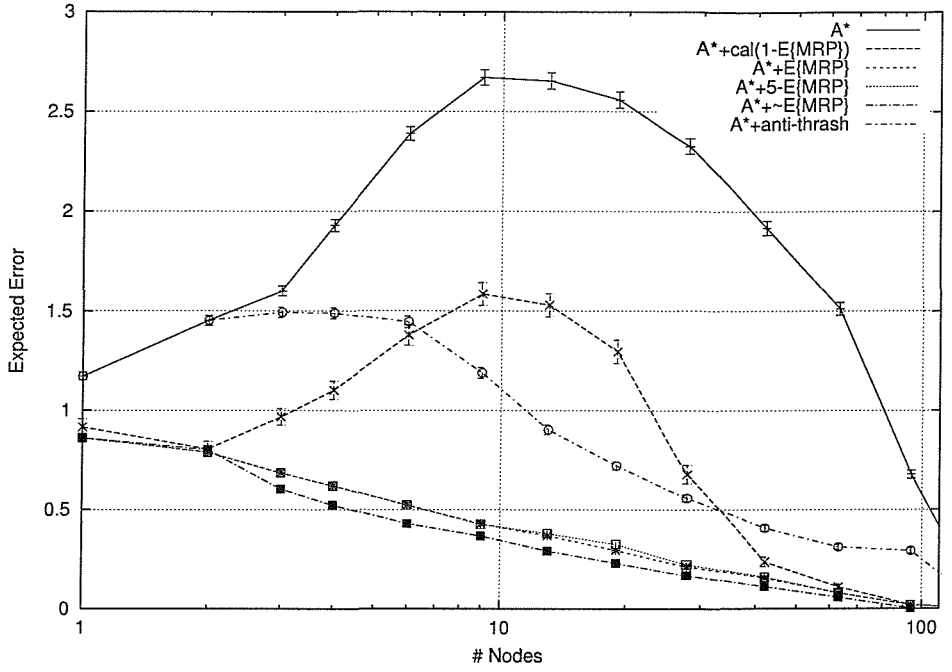


(a)

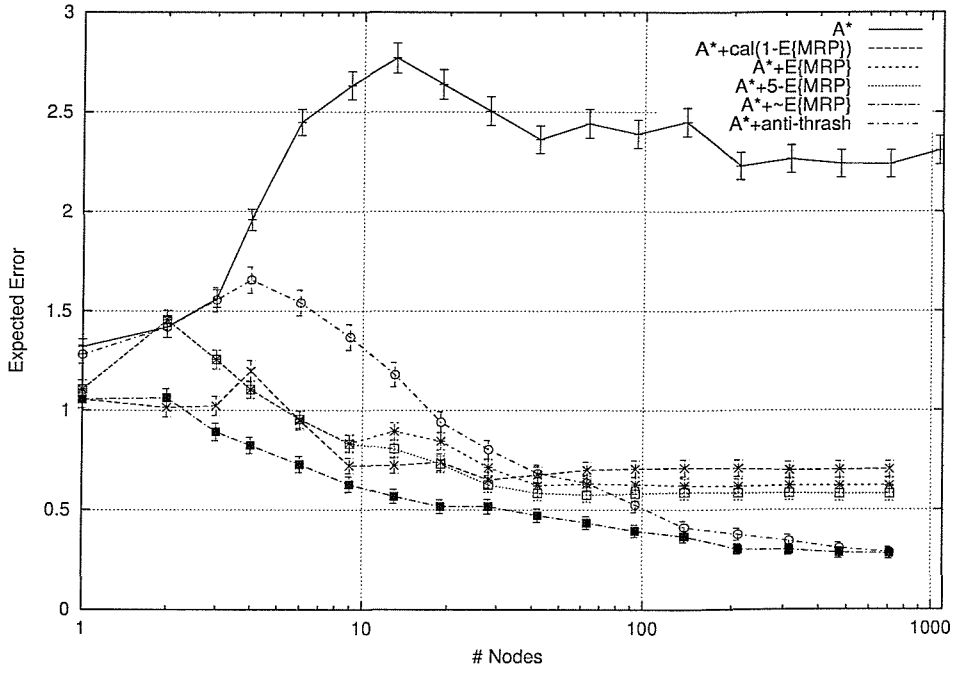


(b)

FIGURE 4.17: Decision making methods expected performance profiles with A* local search space construction on the (a) 8-Puzzle and (b) 15-Puzzle. The results are averages of 3,000 test problems error bars indicate the mean's sample variance. Note the log-scale, and non-zero y-axis in (b).



(a)



(b)

FIGURE 4.18: Decision making methods expected performance profiles with A* local search space construction on the (a) 10x10 grid and (b) 100x100 grid. The results are averages of 3,000 test problems and the error bars indicate the mean's sample variance.

Non-monotonic k-E{MRP} performance profiles. The most striking result on examination of Figure 4.17 and Figure 4.18 are the non-monotonic performance profiles for the k-E{MRP} algorithms, particularly 1-E{MRP}(MM-CAL), in all domains. This is believed to be caused by an adverse interaction between the decision making system and A*'s assessment of what the *best* frontier node is.

Consider, the MM-CAL case. MM-CAL always moves toward the frontier node which has maximal expected value, given its heuristic value. Now, it turns out that, for large heuristic values, the estimated expected value does not improve as rapidly as the heuristic value. Thus, A* can continue extending one part of the local search space (for which $r^*(x, x_f) + h(x_f)$ is still decreasing) long after MM-CAL has decided that another direction is more desirable (because $r^*(x, x_f) + \hat{V}(x_f|h(x_f))$ is lower). As MM-CAL won't change its mind until A* disproves its favoured alternative (by expanding it) A*'s additional effort is wasted. In fact this additional effort can be positively detrimental because it may increase the delay between A* causing MM-CAL to switch to a less informed alternative and this alternative being proved bad. Hence, increasing the chance that the search will be stopped while MM-CAL believes a less informed estimate is better. Essentially, in this case calibration exaggerates the thrashing effect which causes MINIMIN's problems.

A similar argument holds in the 5-E{MRP} case, and to a much lesser extent for E{MRP} and $\sim$-E{MRP}, except in these cases the combination of good value estimates which tend to occur in the best sub-tree smooth out the thrashing effect by requiring A* disprove a number of states before the decision maker switches to a less well-informed alternative.

This analysis, and the dramatic reductions in decision making performance possible, serve to emphasise that the meta-control and decision making systems must be carefully designed to work together to obtain good performance.

Very good performance of ANTI-THRASH vs. MINIMIN. The significantly superior performance of ANTI-THRASH compared to MINIMIN on all domains indicates that thrashing is indeed a significant problem with truncated A* search and that ANTI-THRASH is effective in combating it.

That ANTI-THRASH performs better than all the other algorithms for deep searches on the harder 15-puzzle and 100x100 grid problems is quite surprising, especially as the E{MRP} algorithms use more of the available information to make their decisions. The fact that the ANTI-THRASH's relative performance improves in a similar range to when 5-E{MRP} performance reduces indicates that this effect is likely also due to calibration induced thrashing of the E{MRP} decisions.

4.9.5 Summary of experiments

There are two main points to take away from these experiments;

- firstly that it is possible to make better decisions than naive mechanisms such as MINIMIN by making more effective use of the available information, and
- secondly that the integration of decision making and meta-control must be very carefully managed to maximise overall performance and avoid adverse interactions. If the interaction is managed well, as with ANTI-THRASH on the 15-puzzle, it can produce very significant benefits, however when managed poorly, as with A* and MM-CAL on the grid problems, it can result in massive reductions in overall performance.

4.10 Summary

This chapter has examined the decision making component of a dynamic deliberative agent. Incremental search was chosen as a framework for developing a flexible decision making component as it provides a relatively simple mechanism to trade-off decision quality against computational effort. As they simplify meta-control integration, it was decided to focus on explicitly rational decision making systems, which make decisions by maximising an explicitly computed value estimate. The key function of an explicitly rational decision making system is computing value estimates and the core of this chapter investigated on this problem. In all five main approaches to computing value estimates were described. Three of these; MINIMIN, GSE and BPS are based upon previous work. The other two, E{MRP} and ~E{MRP}, represent new approaches to value estimation in single agent search and ~E{MRP} in particular is a totally novel, highly efficient value estimation method. The effectiveness of these value estimation methods was experimentally tested using fixed depth and A* meta-control routines on 4 problem types, the 8-puzzle, 15-puzzle, 10x10 grid and 100x100 grid. The results of these experiments were encouraging in demonstrating that the sophisticated value estimate systems can improve decision making performance. However, the results also made clear that the right combination of decision making and meta-control is critical to overall performance. With this in mind meta-control is studied next.

Chapter 5

Search control and estimating the value of computation

This chapter extends the incremental search agent design of the previous chapter to include meta-control. This chapter focuses on the search control aspect of the meta-control problem and will not be concerned directly with decision making nor deciding when to stop exploration. The general problem the search control system must solve is,

Definition 5.1 (General search control problem). Given a decision making system, a labelled local state graph, γ, and that only a limited number of computations will be executed before a decision is made, choose the computation to execute which, on average, maximises the final decisions value.

In terms of the incremental decision model of Section 1.2 the search-control system performs step (3) of Figure 1.2. In general, a computation could be any operation which modifies the local state graph, such as computing a new heuristic value for a state or expanding all frontier nodes 1 level deeper. However, the only computations considered in this dissertation are individual frontier state expansions.

Of course search-control must be integrated within the larger incremental-search agent. Thus, because they integrate well with explicitly rational decision makers, we restrict attention to search-control systems based on tractable implementations of the META-GREEDY algorithm of Section 3.3.10. We can think of such systems as *explicitly meta-rational*, in that they explicitly calculate a set of local marginal value of computation estimates, $\Delta\hat{\mathbf{V}}$, and select computations by greedily maximising this value. Thus, the search-control has a similar structure to decision making, that is (i) identify useful features, (ii) compute estimated computation values, (iii) execute the maximum value option and repeat.

As with explicitly rational decision making, the performance of explicitly rational search control depends critically on the accuracy of the value estimates. Thus, the main focus of this chapter is on algorithms for efficiently generating accurate marginal value of computation estimates.

5.1 Approximating the marginal value of computation

As defined in Section 3.3.10 the marginal value of a computation is the change in the value of the agent's trajectory when it is made to execute the computation, c, next compared to executing the current Bayes action action, α, next. That is,

$$\Delta V(c) = V_{c,\pi} - V_{\alpha,\pi}, \tag{5.1}$$

where π indicates that after the initial computation or action execution the agent continues in its normal operation of executing further computations or actions. The agent's best estimates for $V_{\alpha,\pi}$ are computed by the decision maker. Thus the main difficulty in computing $\Delta V(c)$ is estimating the future value of the computation $V_{c,\pi}$.

Note, for brevity in the remainder of this thesis the dependence of a value or value estimate on the current root state x and the agent's default policy π will be left implicit when clear from context. Thus, $V = V_\pi(x_0)$, and $V(u) = V_\pi(x_0, u) = V_{u,\pi}(x_0)$.

As mentioned in Section 3.3.10 we can think of $\Delta V(c)$ as a measure of the sensitivity of the expected value of the agent's final decision to the additional information provided by the computation. For explicitly rational decision makers this decision depends on the relative value of the root's child action value estimates, $\hat{V}(u)$, and in particular on the estimated value of the Bayes action. Thus, as the final decision depends on how the decision maker computes these estimates the value of computation also depends on how the actions' value estimates are computed. Further, as the results of Section 4.9.4.2 demonstrated, to avoid adverse interactions and wasted effort it is particularly important that the meta-level know how the decision maker chooses actions and what information it uses. Thus, the approximations developed in this chapter are specific to the decision making system used.

As with the value function itself $\Delta V(c)$ can be approximated using either inferred, learnt or hybrid estimation functions. For search control only hybrid meta-value estimation functions are considered as learning is computationally intractable.

Learning the marginal value of computation is intractable because, as shown in Figure 3.7, the marginal value of expanding one frontier state is highly dependent on the relative value of other frontier states and how these values are likely to change. For example, if one state is definitely inferior then confirming this has little value (Figure 3.7(a)). However, if it is potentially the best frontier node then expanding it to confirm its superiority may be very valuable (Figure 3.7(c)). In either case the value of expanding a state depends on the likelihood of it being better than all other states. Thus, the meta-value is inherently non-local and any learnt function will depend on the entire set of local state space's value estimates. In Section 4.5.1 it was shown that training and storing a learnt value estimator (the GSE) which depends on the entire local state space is effectively intractable. In fact, the meta-value learning problem is harder than the GSE case because a marginal value must be learnt for each possible computation, and there is at least one

computation per frontier state. Thus, the overall complexity of the meta-value learning problem is $\mathcal{O}(n_f b^{2n-2}|V|^n)$, where n_f is the number of frontier states, b^{2n-2} is an upper-bound on the number of possible local search spaces[1] with effective branching factor b containing less than n states, and $|V|$ is the number of distinct state future values.

Hybrid estimators (Section 4.6) avoid the complexity of the learning systems by learning a simple *local* future value *distribution* estimator and using knowledge of how the value of expanding one node depends on the values of other nodes to define an inference mechanism which works for any local search space and frontier values. This approach of constructing hybrid value of computation estimators based upon identifying a factored model is essentially the one outlined in Section 3.3.10.1 and similar to that used in (Russell and Wefald 1989) and (Baum and Smith 1997).

The key step in developing such estimators is to use knowledge of the decision making systems detailed operation to identify the functional form of the dependency between the states' local value estimates and the inferred global value estimates for the agent's policy, $\hat{V}(x_0) = f(\hat{V}(x_1), \hat{V}(x_2), \ldots)$. The function f is called the *propagation function* as it transmits information from the states' local value estimates to the global estimates upon which decision making is based. The propagation function can then be used to calculate how this value will change with the computation's different outcomes,

$$\Pr(\hat{V}'(x_0){=}v|\hat{\mathbf{V}}, c) = \Pr(f(\hat{V}(x_1), \hat{V}(x_2), \ldots){=}v|\hat{\mathbf{V}}, c). \tag{5.2}$$

Thus, given a learnt distribution over the set of new frontier states local value estimates, $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, c)$, we can use the propagation function to infer a distribution over new global value estimates for the roots child actions, $\hat{V}'(u)$, and hence for the Bayes action. The distribution over future global value estimates can then be used in a number of different ways to estimate the value of the computation.

As computations are restricted to frontier state expansions, to develop a factored model of the effects of computation, we make the following simplifying assumptions,

(i) computations affect only the expanded states value, and

(ii) frontier value independence (Section 4.8.1.3), i.e. the outcome of a computation only depends on the state's local information.

Thus, the effect of a computation, c_x, which expands the state x only depends on and modifies that states local information, and is independent of everything else. This significantly simplifies the problem of learning a model of the computation's effects as $\Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, c_x) = \Pr(V'(x)|V(x), \mathbf{h}(x))$. This distribution over posterior local state value estimates may be the

[1]This, loose, upper bound follows by counting the number of possible edge choices in a depth first traversal of a n node tree. This upper bounds the number of trees as the edge order is unique to the traversal which is unique to the tree.

same distribution used by the decision maker, or specially learnt. Note, the factored model developed in this way is almost identical to the one used by the E{MRP} estimator.

One significant problem with using the value of computation, $\Delta V(c)$, for search control is that potentially it needs to be re-computed for each computation after every expansion. Thus, even if $\Delta V(c)$ can be computed in linear time this gives a search control overhead per node expansion of $\mathcal{O}(n_f)$. Russell and Wefald (1989) suggest only updating the $\Delta V(c)$ estimates after every "gulp" of G node expansions to reduce this overhead to $\mathcal{O}(n_s n_f/G)$. The cost of this reduction is potentially increasing the total number of expansions required due to the reduced accuracy of the $\Delta V(c)$ estimates used for search control. Alternatively, Baum and Smith (1997) suggest increasing the gulp size exponentially as more node expansions are performed. This has the advantage of re-computing the estimates more frequently early on when the overhead is lower and the estimates are likely to change rapidly, without incurring an overhead for larger searches. When the gulp size increases at the rate g this reduces the overhead per node expansion to $\mathcal{O}(n_s \log_g(n_f))$.

Computing a hybrid $\Delta V(c)$ approximation consists of two related parts,

- identifying the propagation function based on the decision makers operation, and
- using the distribution over future global value estimates w.r.t. a computation's possible outcomes to compute an estimate for the marginal value of computation.

The next sections present the development of such meta-value approximation schemes for the two inference based value estimate functions of Chapter 4; MINIMIN, E{MRP}.

5.2 Meta-value estimates for MINIMIN

MINIMIN's value estimates are computed using equation (4.5). Now, by assumption the $r(x,u)$ in the local search space are correct so $r^*_{\gamma_f}(x, u_{x_f})$ is fixed for each x_f. Thus, the propagation equation for the computation, c, which expands the set of frontier states $x_j \in \mathcal{X}_c$ to give updated local value estimates $\hat{V}'(x_f)$ is,

$$\hat{V}'(x) = \max_{x_f \in \gamma_f} \left[r^*_{\gamma_f}(x, u_{x_f}) + \hat{V}'(x_f) \right] = \max \left[\omega, \max_{x_j \in \mathcal{X}_c} [r^*_{\gamma_f}(x, u_{x_j}) + \hat{V}'(x_j)] \right], \quad (5.3)$$

where $\omega = \max_{x_f \in \gamma_f \setminus \mathcal{X}_c} \left[r^*_{\gamma_f}(x, u_{x_f}) + \hat{V}(x_f) \right]$ is the original MINIMIN value of all the frontier states not expanded by c. This is constant for all computation outcomes as computations are assumed to only affect the expanded states. ω acts as a filter on the effect of changing $\hat{V}'(x_j)$ preventing any change which would lower it below ω from having any influence. Thus the value of expanding a node depends on how the expansion changes the states estimated value relative to ω.

5.2.1 Optimistic frontier value estimates

If, as is assumed in A*, $\hat{V}(x)$ is optimistic then expanding a state can only make its value worse and ω has the effect of preventing any frontier state(s) but those with the best MINIMIN value from having any effect on the agent's current value estimate. Note, there may be more than one frontier state with the best MINIMIN value. This is the case examined by Russell and Wefald (1989) with their DTA* algorithm. Russell and Wefald point out that only if one of the current best MINIMIN value frontier states is expanded before the decision is made could the agent change its decision and hence the computation have any value. As all the MINIMIN frontier states must be expanded for any sequence of expansions to have any value, we might as well expand them first. This reasoning leads to A*'s search ordering where best value frontier states are expanded first.

In fact, the current decision is unchanged unless the MINIMIN value of the current preferred action, $\hat{V}(\alpha)$, is made worse than the MINIMIN value of the next best action $\hat{V}(\beta)$. Thus, only if all frontier states in the sub-tree below α with MINIMIN value better than $\hat{V}(\beta)$ are expanded to make all their values worse than $\hat{V}(\beta)$ can *any* sequence of computations have positive value. So once again we might as well expand these states first. To decide on the order for expanding states within this set Russell and Wefald introduce the *hardest first* criterion. Basically, this says that we should expand next the node which is *least likely* to be worse than $\hat{V}(\beta)$ because this is the node which is *most likely* to actually prove that α is actually the best action choice. Russell and Wefald note that this is a slight generalisation of the commonly used A* tie breaking strategy of favouring the node with the *lowest* heuristic value. Lower heuristic values have lower errors and hence are less likely to prove worse than ω if, as is generally true, heuristic error is an increasing function of heuristic value.

Thus, in the special case of MINIMIN decision making with optimistic frontier estimates near optimal search control can be implemented without explicitly computing the value of the possible computations using the hardest first search-ordering.

5.2.2 General frontier estimates

In the general case $\hat{V}(x_j)$ may be optimistic or pessimistic so potentially any node could be expanded to become better than ω. Therefore, there is no clear criterion for identifying which states *must* be expanded for any sequence of computations to have value, and we must actually compute an estimate for the expected value of a (set of) state expansion(s) for search control.

The propagation equation (5.3) allows us to easily compute the new estimated value of executing the action, u, immediately after the (sequence of) computation(s), c, denoted $\hat{V}_{c,u}$. Hence we can find the *myopic expected value of computation* (MEVC), $\Delta\hat{V}^i_{c,c_\alpha} = \hat{V}^i_{c,c_\alpha} - \hat{V}_{c,\alpha}$, using equation

(3.27) of Section 3.3.10. That is,

$$\Delta\hat{V}^i_{c,c_\alpha} = \sum_{\hat{\mathbf{V}}'} \Pr(\hat{\mathbf{V}}'|\hat{\mathbf{V}}, c)\left[\max_u \hat{V}'(u) - \hat{V}'(\alpha)\right], \tag{5.4}$$

where $\hat{V}'(u) = \hat{V}_{c,u}$ is computed using the propagation equations to propagate the new value for the expanded state to the roots action value estimates.

The MEVC computation (5.4) simplifies considerably if we assume "sub-tree independence" (Russell and Wefald 1989) so a computation only affects a single root action value, $\hat{V}(u)$. Under this assumption all the root's action values except the one affected by the computation are constant. Depending on which action value is affected (5.4) simplifies to one of two cases. Firstly, if the computation *does not* effect $\hat{V}(\alpha)$ then the Bayes action will remain unchanged unless the new MINIMIN value is better than $\hat{V}(\alpha) = \omega$. Hence, only the cases where $\hat{V}'(c) > \omega$ need be considered. Let, $\mathrm{p}_c(v) = \Pr(\hat{V}'(c)|\hat{\mathbf{V}}, c)$ denote the probability of the new MINIMIN value of the states expanded by computation c having value v. Then when c does not effect $\hat{V}(\alpha)$ the MEVC is given by,

$$\Delta\hat{V}^i_{c,c_\alpha} = \sum_{v>\omega} \mathrm{p}_c(v)[v - \omega]. \tag{5.5}$$

However, if the computation *only* affects $\hat{V}(\alpha)$ then the Bayes action will be unchanged unless α's new value is worse than the value of the next best action, $\hat{V}(\beta)$. Thus, only the cases where $\hat{V}'(\alpha) < \hat{V}(\beta)$ need be considered. Further, α's new MINIMIN value cannot be made worse than the MINIMIN value of its non-expanded states. Denote this value by ω as it also represents the value of the best non-expanded α leaf. As α's new value cannot be less than ω the Bayes action can only change and the MEVC be non-zero if $\omega < \hat{V}(\beta)$. Then when c only effects $\hat{V}(\alpha)$ the MEVC is given by,

$$\Delta\hat{V}^i_{c,c_\alpha} = \sum_{\hat{V}'(\alpha)<\hat{V}(\beta)} \Pr(\hat{V}'(\alpha)|\hat{\mathbf{V}}, c)[\hat{V}(\beta) - \hat{V}'(\alpha)], \tag{5.6}$$

$$= \sum_{v<\omega} \mathrm{p}_c(v)[\hat{V}(\beta) - \omega] + \sum_{\omega<v<\hat{V}(\beta)} \mathrm{p}_c(v)[\hat{V}(\beta) - v]. \tag{5.7}$$

5.2.3 Implementation issues

The MEVC computation can be simplified by moving to continuous space and transforming from PDFs to CDFs in the following way. Consider first, the case when c does not effect $\hat{V}(\alpha)$, (5.5) becomes,

$$\Delta\hat{V}^i_{c,c_\alpha} = \int_\omega^\infty \mathrm{p}_c(v)[v - \omega]dv = \int_\omega^\infty \mathrm{p}_c(v)vdv - \int_\omega^\infty \mathrm{p}_c(v)\omega dv. \tag{5.8}$$

Using the fact that $\int_x^\infty \mathrm{p}(y)dy = \mathrm{P}^>(x)$ so $-\mathrm{p}(x) = \frac{d\,\mathrm{P}^>(x)}{dx}$ and integrating by parts,

$$\Delta\hat{V}^i_{c,c_\alpha} = [-v\,\mathrm{P}^>_c(v)]^\infty_\omega - \int_\omega^\infty -\mathrm{P}^>_c(v)dv - \omega\,\mathrm{P}^>_c(\omega), \tag{5.9}$$

$$= \int_\omega^\infty \mathrm{P}^>_c(v)dv. \tag{5.10}$$

where we have used the fact that, by l'Hôpital's rule, $\lim_{v\to\infty} v\,\mathrm{P}^>(v) = 0$ if $\mathrm{p}(v)$ tends to zero in the positive limit faster than $1/v^2$. This is true of all distributions with finite variance, which are the only ones considered here. Similarly, for the case when c only effects $\hat{V}(\alpha)$, (5.7) becomes,

$$\Delta\hat{V}^i_{c,c_\alpha} = \int_{-\infty}^\omega \mathrm{p}_c(v)[\beta - \omega]dv + \int_\omega^\beta \mathrm{p}_c(v)[\beta - v]dv, \tag{5.11}$$

where $\beta = \hat{V}(\beta)$ for brevity. Multiplying out the brackets, collecting terms and then integrating by parts gives,

$$\Delta\hat{V}^i_{c,c_\alpha} = \int_{-\infty}^\beta \mathrm{p}_c(v)\beta dv - \int_{-\infty}^\omega \mathrm{p}_c(v)\omega dv - \int_\omega^\beta \mathrm{p}_c(v)v dv, \tag{5.12}$$

$$= \beta\,\mathrm{P}_c(\beta) - \omega\,\mathrm{P}_c(\omega) - [v\,\mathrm{P}_c(v)]^\beta_\omega + \int_\omega^\beta \mathrm{P}_c(v)dv, \tag{5.13}$$

$$= \int_\omega^\beta \mathrm{P}_c(v)dv. \tag{5.14}$$

Combining these results we have the following equation for the myopic value of a computation, c, for a MINIMIN decision maker,

$$\Delta\hat{V}^i_{c,c_\alpha} = \begin{cases} \int_\omega^\infty 1 - \mathrm{P}_c(v)dv & \text{if } c \textbf{ does not} \text{ effect } \hat{V}(\alpha) \\ \int_\omega^\beta \mathrm{P}_c(v)dv & \text{if } c \textbf{ only} \text{ effects } \hat{V}(\alpha) \end{cases}. \tag{5.15}$$

where ω is the MINIMIN value of the states in the α sub-tree *not* expanded by c, β is the value of the second best action, and $\mathrm{P}_c(v)$ is the probability that the new MINIMIN value of the states expanded by c less than v. Equation (5.15) can be used to rapidly calculate the MEVC by a simple integration of CDFs. In the discrete case this integration becomes a summation, and the complexity of computing the MEVC is $\mathcal{O}(n_s)$ per state, where n_s is the number of steps in the discrete distribution. Equation (5.15) is a generalisation of a similar result stated without proof for discrete integer valued distributions in (Russell and Wefald 1989, p 146).

One problem with using the MEVC for search control is that its myopic nature can cause it to significantly underestimate the value of a computation. This can lead to the situations, such as that illustrated in Figure 3.7(d), where computation is clearly still worthwhile but no single computation has non-zero MEVC. For example, if more than 1 frontier state below α has MINIMIN value better than $\hat{V}(\beta)$ then expanding only one of these nodes cannot change the agent's decision and hence has zero value. This short-sightedness can be overcome in a number of ways such as using Russell and Wefald (1989)'s hardest-first tie-breaking rule of the

previous section to decide between zero MEVC computations or one of the alternative value of computation approximations presented in this chapter.

A more direct approach is simply to look further head in the MEVC calculation by considering sets of computations. Fortunately, equation (5.15) is still valid for sets of state expansions, $\mathbf{c}$, provided its conditions are met. For computational tractability, the number of sets of computations considered must be limited in some way as the number of possibilities is exponential in the set size. For example, Russell and Wefald only consider sets which are constructed incrementally by adding states according to the "hardest first" heuristic. The frontier value independence assumption means the new MINIMIN value distribution of the set is simply the product of their individual distributions, i.e. $\mathrm{P}'_{\mathbf{c}}(v) = \prod_i \mathrm{P}'_{\mathbf{c}(i)}(v)$.

5.3 Meta-value estimates for E{MRP} and ~E{MRP} estimators

The E{MRP} estimator (Section 4.8.1) computes updated global value estimates for the state x based on the assumption that the Bayes action choice at x is made after the current frontier value estimates have been updated with the additional information, i. Under the E{MRP}'s assumptions and the additional frontier value independence assumption the state's updated value distribution is computed using (4.25). This value distribution computed in this way is called the E{MRP} CDF. Notice, that for the E{MRP} estimator the frontier states value estimates are distributions and the effect of expanding a state is to give it a new distribution. Let $\mathrm{P}_x(v) = \Pr(\hat{V}'(x)<v|\hat{V}(x), i)$ denote the old future value distribution for the frontier state x. Then, the propagation function for the computation, c, which expands the set of frontier states $x_j \in \mathcal{X}_c$ to give updated state future value distributions $\mathrm{P}'_{x_j}(v)$ is,

$$\Pr(\hat{V}'(x) < v) = \prod_{x_f \in \gamma_f} \Pr\left(r^*_\gamma(x, u_{x_f}) + \hat{V}'(x_f) < v | \mathrm{P}'_{x_f}\right), \tag{5.16}$$

$$= \Pr(\omega<v) \prod_{x_j \in \mathcal{X}_c} \Pr\left(r^*_\gamma(x, u_{x_f}) + \hat{V}'(x_j) < v | \mathrm{P}'_{x_j}\right), \tag{5.17}$$

$$= \mathrm{P}_\omega(v)\, \mathrm{P}_c(v|\mathrm{P}'), \tag{5.18}$$

where, $\mathrm{P}_\omega(v) = \Pr(\omega<v) = \prod_{x_f \in \gamma_f \setminus \mathcal{X}_c} \Pr\left(r^*_\gamma(x, u_{x_f}) + \hat{V}(x_f) < v|\hat{V}(x), i\right)$ is the original E{MRP} CDF for all the frontier states excluding those in $\mathcal{X}_c$, and $\mathrm{P}_c(v|\mathrm{P}') = \prod_{x_j \in \mathcal{X}_c} \Pr(r^*_\gamma(x, u_{x_j}) + \hat{V}'(x_j) < v|\mathrm{P}'_{x_j})$ is the new E{MRP} CDF for all the frontier states updated by c.

By the frontier value independence assumption $\Pr(\omega<v)$ is independent of the computations outcome. Thus, as $\Pr(\omega<v)$ is constant at each v, x's new E{MRP} CDF is a *linear function* of the new E{MRP} CDF of the states expanded by c, i.e. $\Pr(\hat{V}'(x)<v) = k\,\mathrm{P}_c(v|\mathrm{P}')$.

Decision making is done with respect to the expected value of the root's actions, $\langle\hat{V}'(u)\rangle$. Let, $\mathrm{P}'_c(v)$ denote the new E{MRP} CDF for the frontier states below u affected by the computation

c, and $\mathrm{P}_\omega(v)$ the original E{MRP} CDF for those not affected by c. Then, using (5.18), u's new global future value estimate is given by,

$$\langle \hat{V}'(u) | \, \mathrm{P}'_c \rangle = \int_{-\infty}^{\infty} v \frac{d(\mathrm{P}_\omega(v) \, \mathrm{P}'_c(v))}{dv} dv. \tag{5.19}$$

5.3.1 The MEVC approximation

If we assume sub-tree independence, so a computation cannot effect both the current Bayes action's value and some other action's value, then equations (5.18) and (5.19) can be used to compute a MEVC approximation for the value of computation. As for the MINIMIN MEVC estimate there are two cases to consider. Firstly, if c only effects $\hat{V}(\alpha)$ then,

$$\Delta \hat{V}^i_{c,c_\alpha} = \int \Pr(\mathrm{p}'_c | \hat{\mathbf{V}}, c) \left(\max[\hat{V}(\beta), \langle \hat{V}'(\alpha) | \, \mathrm{p}'_c \rangle] - \langle \hat{V}'(\alpha) | \, \mathrm{p}'_c \rangle \right) d\mathrm{p}'_c, \tag{5.20}$$

where the integral is over the entire range of new p'_c value distributions and $\langle \hat{V}'(\alpha) | \, \mathrm{p}'_c \rangle$ is computed using (5.19). Secondly, if c only effects $\hat{V}(u)$ then MEVC is given by,

$$\Delta \hat{V}^i_{c,c_\alpha} = \int \Pr(\mathrm{p}'_c | \hat{\mathbf{V}}, c) \left(\max[\langle \hat{V}'(u) | \, \mathrm{p}'_c \rangle, \hat{V}(\alpha)] - \hat{V}(\alpha) \right) d\mathrm{p}'_c. \tag{5.21}$$

To actually compute these values we need to define an integrable set of new distributions for p'_c and a second order distribution over these new distributions. In general, these quantities must be learnt. However, in certain circumstances they can be derived from the existing distribution information, $\Pr(\hat{V}(x_j) = v | \hat{\mathbf{V}}, i)$.

As defined in the derivation of the E{MRP} estimator (Section 4.8.1) x's original future value distribution, $\Pr(\hat{V}(x){=}v | \hat{\mathbf{V}}, i)$, represents the probability that the effect of the additional information, i, is to resolve the uncertainty about the value of x to the correct *exact value*, v. Thus, if we assume that the computation c realises exactly the same information about each state expanded as contained in i, then c must have the same effect with the same probability. Now, by definition for a set of frontier states $\mathcal{X}_c$ the E{MRP} propagated CDF, $\mathrm{P}_{\mathcal{X}_c}(v|i) = \prod_{x \in \mathcal{X}_c} \mathrm{P}_x(v|i)$, gives the probability that the expanded states new E{MRP} estimate is less than v, when the states take on exact values according to their combined probability distribution. Thus, $\mathrm{p}_{\mathcal{X}_c}(v) = d\mathrm{P}_{\mathcal{X}_c}(v)/dv$, gives the probability that the new E{MRP} estimate for these states is exactly v. Hence, if c expands the set of frontier states $\mathcal{X}_c$ then, with probability $\mathrm{p}_{\mathcal{X}_c}(v) = \mathrm{p}_c(v)$, the new E{MRP} value estimate for all the states updated by c is exactly v and their new value distribution is an impulse, i.e. $\mathrm{p}'_c(y) = \delta_v(y) = \delta(y{-}v)$.

Let $\mathrm{P}_c(v) = \prod_{x \in \mathcal{X}_c} \Pr(\hat{V}(x){<}v | \hat{\mathbf{V}}, i)$ denote the original E{MRP} CDF for the states updated by c, and $\mathrm{p}_c(v)$ their PDF. Then, assuming c has the same effect on the states it expands as i, the

MEVC can be computed when c only effects $\hat{V}(\alpha)$ using,

$$\Delta\hat{V}^i_{c,c_\alpha} = \int_{-\infty}^{\hat{V}(\beta)} \mathrm{p}_c(v)\left(\max[\hat{V}(\beta), \langle\hat{V}'(\alpha)|\,\mathrm{p}'_c{=}\delta_v\rangle] - \langle\hat{V}'(\alpha)|\,\mathrm{p}'_c{=}\delta_v\rangle\right) dv, \tag{5.22}$$

where the upper limit follows because v is exact and hence a lower bound on $\langle\hat{V}'(\alpha)|\,\mathrm{p}'_c{=}\delta_v\rangle$ and the Bayes action can only change if $\langle\hat{V}'(\alpha)|\,\mathrm{p}'_c{=}\delta_v\rangle{<}\hat{V}(\beta)$. If c only effects $\hat{V}(u)$ then the MEVC is given by,

$$\Delta\hat{V}^i_{c,c_\alpha} = \int_{-\infty}^{\infty} \mathrm{p}_c(v)\left(\max[\langle\hat{V}'(u)|\,\mathrm{p}'_c{=}\delta_v\rangle, \hat{V}(\alpha)] - \hat{V}(\alpha)\right) dv. \tag{5.23}$$

This approximation suffers from the same short-sightedness problems as in the MINIMIN case. Fortunately, the E{MRP} estimator already includes the effects of the additional information, i, which can be used to derive alternative marginal value of computation estimates which do have this problem.

5.3.2 The root value sensitivity approximation, $d\hat{V}(x_0)$

As mentioned earlier (Section 4.8.1.3) the E{MRP} estimate is equivalent to estimating the intrinsic value of the policy, $[c_i, c_\alpha]$, consisting of executing the computation, c_i, which generates the additional information i and then executing the Bayes action w.r.t. the updated action value estimates. Hence an actions E{MRP} estimate $\hat{V}(u)$ is actually $\hat{V}^i_{u,c_i,c_\alpha}$, and the root states E{MRP} estimate $\hat{V}(x_0)$ is $\hat{V}^i_{c_i,c_\alpha}(x_0)$. Therefore the root state's E{MRP} estimate is a direct measure of the future value of the information, i.

Now, as above, assume that the computation c is part of c_i so it has the same effect on the states it expands. Consider the case when c has already been executed and the frontier values updated to reflect the information it provides. Executing c again will not realise any new information and hence has no value. The value of $\hat{V}^i_{c_i,c_\alpha}(x_0)$ computed in this new situation is a measure of the value of the remaining computations, $c_i - c$. Hence, the *change* in $\hat{V}^i_{c_i,c_\alpha}(x_0)$ before and after the computation gives a measure of the *contribution* of c to the total value of c_i, i.e. $\Delta V(c) \approx V(c_i) - V(c_i - c)$. Essentially, this gives us a non-myopic estimate for the value of c which includes its effects on the future value of the computation c_i which is carried after it finishes.

Unfortunately, whilst for any particular computation outcome the change in $\hat{V}^i_{c_i,c_\alpha}$ may have positive or negative value, its expected change is zero. This is because $\hat{V}^i_{c_i,c_\alpha}$ is defined as the average value of the Bayes action over final frontier value assignments after c_i is executed. As the probability of any particular set of final frontier value assignments is unaffected by executing c first, the average must also be unaffected. Hence, $\hat{V}^i_{c_i,c_\alpha} = \hat{V}^i_{c,c_i,c_\alpha}$.

To overcome this problem we instead compute the *expected absolute change* in $\hat{V}^i_{c_i,c_\alpha}$. This gives a measure of how much of the uncertainty in the true value of $\hat{V}^i_{c_i,c_\alpha}$ is due to c, and hence how

much the uncertainty would be reduced by executing c. The variance of $\hat{V}^i_{c_i,c_\alpha}$ gives a similar measure. The absolute value is preferred as it weights all changes equally which is desirable as the changes represent true marginal values. We denote this measure by $d\hat{V}(x_0)$ because it computes the sensitivity of the root's E{MRP} value to changes in the values of a subset of the frontier states. It is given by,

$$d\hat{V}(x_0) = \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left| \langle \hat{V}'(x_0) | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(x_0) \rangle \right| dv. \tag{5.24}$$

5.3.3 The expected step size approximation, ESS

One problem with $d\hat{V}(x_0)$ is that it treats all changes in the value of the root as equally important, irrespective of whether the Bayes action choice has changed. Thus, it tends to over-value expansions which reduce the uncertainty in the Bayes actions value but are unlikely to change it. To overcome this problem Baum and Smith (1997) suggest an alternative non-myopic approximation called the expected step size, ESS.

The ESS is based upon the observation that the E{MRP} estimate of the current Bayes action, $\langle \hat{V}(\alpha) \rangle$, gives a direct measure of the value of the agent's current information, whereas the E{MRP} estimate for the value of the root state, $\langle \hat{V}(x_0) \rangle$, gives a measure of the value of the additional information i. The difference between these two values gives a measure of the *marginal* benefit of the additional information i assuming the agent executes the Bayes action after i. Hence, assuming i is generated by the computation c_i, its MEVC is given by,

$$\Delta V(i) = \Delta V(c_i) = \langle \hat{V}(x_0) \rangle - \langle \hat{V}(\alpha) \rangle = \hat{V}^i_{c_i,c_\alpha} - \hat{V}^i_{\alpha,c_i}. \tag{5.25}$$

Baum and Smith call this the expected value of all future expansions, denoted $U_{\text{all leaves}}$, as c_i represents the expansion of all the current frontier states to exact values. Now, if we again assume that the computation c is a subset of c_i then the change in $\Delta V(i)$ before and after c is a measure of its contribution to i's total marginal value, i.e. $\Delta V(c) \approx \Delta V(c_i) - \Delta V(c_i - c)$. The advantage of this measure over $d\hat{V}(x_0)$ is that changes in $\hat{V}(\alpha)$ which do not change the Bayes action have no net marginal value because they affect $\hat{V}(x_0)$ and $\hat{V}(\alpha)$ equally.

Given this reasoning the expected change in $\Delta V(i)$, denoted $d\Delta V(i)$, seems a natural non-myopic measure of the marginal value of the computation c,

$$d\Delta V(i) = \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left[\Delta V(i | \, \mathrm{p}'_c = \delta_v) - \Delta V(i) \right] dv, \tag{5.26}$$

$$= \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left[\langle \hat{V}(x_0) | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(\alpha') | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(x_0) \rangle + \langle \hat{V}(\alpha) \rangle \right] dv, \tag{5.27}$$

where $\alpha' = \max_u \hat{V}'(u)$ is the new Bayes action after the computation. Unfortunately, as Baum and Smith point out, this is actually equal to the negative of the MEVC. The problem is that, as noted above, by definition the average change in $\langle \hat{V}(x_0) \rangle$ is zero, so $\langle \hat{V}(x_0) \rangle$ and

$\langle \hat{V}(x_0) | \, \mathrm{p}'_c = \delta_v \rangle$ cancel and (5.27) becomes,

$$d\Delta V(i) = \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left[\langle \hat{V}(\alpha) \rangle - \langle \hat{V}(\alpha') | \, \mathrm{p}'_c = \delta_v \rangle \right] dv = -\mathrm{MEVC}(c), \tag{5.28}$$

which, as can be seen by comparison with (5.4) is in fact just the negative of c's MEVC.

To overcome this problem Baum and Smith suggest we instead compute the expected absolute change in $\Delta V(c_i)$,

$$d|\Delta V(i)| = \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left| \Delta V(i | \, \mathrm{p}'_c = \delta_v) - \Delta V(i) \right| dv, \tag{5.29}$$

$$= \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left| \langle \hat{V}(x_0) | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(\alpha') | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(x_0) \rangle + \langle \hat{V}(\alpha) \rangle \right| dv, \tag{5.30}$$

Baum and Smith justify this measure by noting that, as for the $d\hat{V}(x_0)$ case, all changes in $\Delta V(i)$ are equally important. Increases, indicate that our current information was is serious error and much less reliable than we thought and therefore help avoid potentially costly blunders. Decreases indicate that we are becoming more sure of our current information and hence getting closer to committing to action.

However, we do not in fact use $d|\Delta V(i)|$ but a closely related approximation that differs from (5.30) only in that we require that $\alpha' = \alpha$. This is the ESS,

$$\mathrm{ESS}(c) = \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left| \Delta V(i | \, \mathrm{p}'_c = \delta_v, \alpha' = \alpha) - \Delta V(i) \right| dv, \tag{5.31}$$

$$= \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left| \langle \hat{V}(x_0) | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(\alpha) | \, \mathrm{p}'_c = \delta_v \rangle - \langle \hat{V}(x_0) \rangle + \langle \hat{V}(\alpha) \rangle \right| dv, \tag{5.32}$$

$$= \int_{-\infty}^{\infty} \mathrm{p}_c(v) \left| d\hat{V}(x_0) - d\hat{V}(\alpha) \right| dv. \tag{5.33}$$

The ESS is identical to $d|\Delta V(i)|$ when the computation does *not* change the Bayes action. However, when the Bayes action does change the ESS underestimates $\langle \hat{V}(\alpha') | \, \mathrm{p}'_c = \delta_v \rangle$, which can have either a positive or negative effect on the total absolute change. Thus, the net effect of this approximation is to introduce an additional source of error into the value of information estimates. Notice, that (5.33) degenerates to $d\hat{V}(x_0)$ if c does not affect $\hat{V}(\alpha)$, as in this case $\langle \hat{V}(\alpha) \rangle$ is a constant and cancels leaving only the expected absolute change in $d\hat{V}(x_0)$. This fact can be used to further speed up the calculation of the ESS.

5.3.4 Implementation issues

Examining the equations for the value of computation approximations developed in this section, that is; (5.22) and (5.23) for the MEVC, (5.24) for the $d\hat{V}(x_0)$, and (5.33) for the ESS, we see that they all rely critically on efficiently computing the new average E{MRP} estimate for the root node or one of its children with respect to the computations possible outcomes,

$\langle \hat{V}'(\cdot) | \, \mathrm{p}'_c = \delta_y \rangle$. Fortunately, due to the linearity of the propagation equations and the special form of the computations outcome distribution, these averages can be computed in constant time.

The required new average E{MRP} value for a state x can be found using the propagation function (5.19). In the specific case considered here, where $\mathrm{P}'_c(v) = \mathcal{H}(v-y)$, this simplifies to,

$$\begin{aligned}
\langle \hat{V}'(x) | \, \mathrm{p}'_c = \delta_y \rangle &= \int_{-\infty}^{\infty} v \frac{d(\mathrm{P}_\omega(v)\,\mathrm{P}'_c(v))}{dv} dv, && (5.34)\\
&= \int_{-\infty}^{y^-} v \frac{d(\mathrm{P}_\omega(v)\mathcal{H}(v-y))}{dv} dv - \int_{y+}^{\infty} v \frac{d(1-\mathrm{P}_\omega(v)\mathcal{H}(v-y))}{dv} dv, && (5.35)\\
&= 0 - [v(1-\mathrm{P}_\omega(v))]_{y^+}^{\infty} + \int_{y+}^{\infty} (1-\mathrm{P}_\omega(v)) dv, && (5.36)
\end{aligned}$$

where we have used the fact that, $df(x)/dx = -d(1-f(x))/dx$, and that $\mathcal{H}(v-y) = 0$ below y to eliminate the first integration in (5.35) and integration by parts for the second. The notation y^- and y^+ denote the limits approaching y from the negative and positive directions respectively. Using the fact that $\lim_{v\to\infty} v(1-\mathrm{P}_\omega(v)) = 0$ when $\mathrm{p}_\omega(v)$ tends to zero faster than v^2, this simplifies to,

$$\langle \hat{V}'(u) | \, \mathrm{p}'_c = \delta_y \rangle = y(1-\mathrm{P}_\omega(y^+)) + \int_{y^+}^{\infty} (1-\mathrm{P}_\omega(v)) dv. \quad (5.37)$$

To complete the computation of $\langle \hat{V}'(x) | \, \mathrm{p}'_c = \delta_y \rangle$ we need $\mathrm{P}_\omega(v)$. This can be found by inverting (5.18) with the original distributions to give, $\mathrm{P}_\omega(v) = \Pr(\hat{V}(x)<v)/\mathrm{P}_c(v)$. Substituting this into (5.37) gives,

$$\langle \hat{V}'(x) | \, \mathrm{p}'_c = \delta_y \rangle = y\left(1 - \frac{\mathrm{P}_x(y^+)}{\mathrm{P}_c(y^+)}\right) + \int_{y^+}^{\infty} \left(1 - \frac{\mathrm{P}_x(v)}{\mathrm{P}_c(v)}\right) dv, \quad (5.38)$$

where $\mathrm{P}_x(v) = \Pr(\hat{V}(x)<v)$ and $\mathrm{P}_c(v)$ are the original E{MRP} CDFs for the state x and the computation c respectively. Computing the integral in (5.38) still requires significant computational effort. However, if $\mathrm{P}_c(v)$ is a piecewise constant staircase distribution (Section 4.8.2.1) this can be reduced to constant time. Let v_i denote lower bound of the ith step of $\mathrm{P}_c(v)$, then as $\mathrm{P}_c(v)$ is constant between its steps the integral in (5.38) can be simplified to,

$$\begin{aligned}
I(v_i) &= \int_{v_i}^{\infty} \left(1 - \frac{\mathrm{P}_x(z)}{\mathrm{P}_c(z)}\right) dz = \int_{v_i}^{v_{i+1}} \left(1 - \frac{\mathrm{P}_x(z)}{\mathrm{P}_c(z)}\right) dz + \int_{v_{i+1}}^{\infty} \left(1 - \frac{\mathrm{P}_x(z)}{\mathrm{P}_c(z)}\right) dz, && (5.39)\\
&= [z]_{v_i}^{v_{i+1}} - \frac{1}{\mathrm{P}_c(v_i)} \int_{v_i}^{v_{i+1}} \mathrm{P}_x(z) dv + I(v_{i+1}), && (5.40)\\
&= v_{i+1} - v_i - \frac{1}{\mathrm{P}_c(v_i)} \left(\int_{-\infty}^{v_{i+1}} \mathrm{P}_x(z) dz - \int_{-\infty}^{v_i} \mathrm{P}_x(z) dz \right) + I(v_{i+1}), && (5.41)
\end{aligned}$$

where $\mathrm{P}_c(z)$ can be taken outside the first integral because it is constant over the range of integration. Hence, if $\int_{-\infty}^{v_i} \mathrm{P}_x(z) dz$ can be computed in constant time, by for example

pre-computing it and caching the results, then by working back-wards from ∞ (so $I(v_{i+1})$ is computed before $I(v_i)$) the integral I can be computed iteratively in constant time for consecutive steps in c's CDF. Notice, this is a significant saving as the naive approach of computing the integral directly for each of $\mathrm{P}_x(z)$'s steps could take up to $n_f \times n_s$ iterations because $\mathrm{P}_x(z)$ is a combination of all n_f frontier states.

Thus, if the outer integrals over c's outcomes in equations (5.22), (5.23), (5.24) and (5.33), are also computed backwards from ∞ the updated averages can be computed in constant time per-outcome. Hence the overall complexity of computing the any of the MEVC, $d\hat{V}(x_0)$ or ESS approximations is $\mathcal{O}(n_c)$, where n_c is the number of steps in c's outcome distribution.

5.3.4.1 Implementation for the ~E{MRP} approximations

The ~E{MRP} estimator has exactly the same propagation function, (5.18), as E{MRP} so the same value of computation approximations can be used, namely MEVC, $d\hat{V}(x_0)$ and ESS. Indeed, as $\mathrm{P}_x(z)$ and $\mathrm{P}_\omega(z)$ are both normally distributed it should technically be possible to compute (5.38) exactly. However, this requires evaluating the following integration,

$$\int_{y^+}^{\infty} \left(1 - \frac{\Phi\left(\frac{v-\mu_x}{\sigma_x}\right)}{\Phi\left(\frac{v-\mu_c}{\sigma_c}\right)}\right) dv, \tag{5.42}$$

where (μ_x, σ_x) and (μ_c, σ_c) are the mean and standard deviations for respectively x and c's distributions. Unfortunately, I was unable to find a analytic solution to this integral.

Instead the value of computation measures are calculated using (5.41) by approximating $\mathrm{P}_c(v)$ with a discrete staircase distribution. $\mathrm{P}_x(z)$ does not have to be discretised as it can be integrated exactly. If $\mathrm{p}_x(z)$ is the normal distribution $\mathcal{N}_{\mu,\sigma}$ then,

$$\int_{-\infty}^{v_i} \mathrm{P}_x(z)dz = \int_{-\infty}^{v_i}\int_{-\infty}^{z} \frac{1}{\sigma}\mathcal{N}\left(\frac{x-\mu}{\sigma}\right) dxdz, \tag{5.43}$$

$$= \left[z\Phi\left(\frac{x-\mu}{\sigma}\right)\right]_{-\infty}^{v_i} - \int_{-\infty}^{v_i} \frac{z}{\sigma}\mathcal{N}\left(\frac{z-\mu}{\sigma}\right) dz. \tag{5.44}$$

Note that $d(\mathcal{N}(\frac{z-\mu}{\sigma}))/dz = -\frac{z-\mu}{\sigma}\mathcal{N}(\frac{z-\mu}{\sigma})$, so $z\mathcal{N}(\frac{z-\mu}{\sigma}) = \mu\mathcal{N}(\frac{z-\mu}{\sigma}) - \sigma d(\mathcal{N}(\frac{z-\mu}{\sigma}))/dz$. Using this to substitute for $z\mathcal{N}(\frac{z-\mu}{\sigma})$ in the integral and integrating by parts we obtain,

$$\int_{-\infty}^{v_i} \mathrm{P}_x(z)dz = (v_i - \mu)\Phi\left(\frac{v_i-\mu}{\sigma}\right) + \sigma\mathcal{N}\left(\frac{v_i-\mu}{\sigma}\right). \tag{5.45}$$

Thus, as $\int_{-\infty}^{v_i} \mathrm{P}_x(z)dz$ can be computed in $\mathcal{O}(1)$ time the value of computation measures can be computed in $\mathcal{O}(n_c)$ time in the continuous case using the same techniques as for the discrete case.

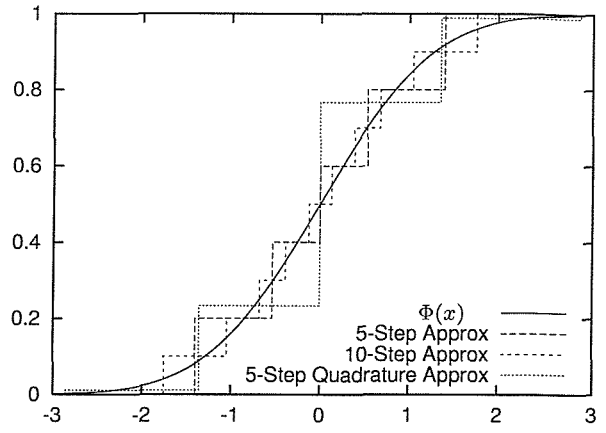


FIGURE 5.1: This diagram shows three staircase approximations for the standard normal distribution's CDF, two found by allocating equal probability to each step, and another based upon Gaussian quadrature. Surprisingly, the 5-step quadrature based approximation gives equivalent equivalently accurate $\langle \hat{V}'(x) | \, \mathrm{p}'_c \rangle$ approximations as the 10-step equal allocation.

The quality of this approximation is clearly dependent on the accuracy of $\mathrm{P}_c(v)$'s discretisation. However, as the cost of the computation increases linearly with the number of steps in the approximation, we must trade-off approximation accuracy against computational cost. Thus, placing the steps in the staircase approximation is very important. Initially, the staircase was constructed to minimise the total absolute error, by allocating equal probability mass to each step. However, in use it was found that the tails of the distribution were most important to accurately computing $\langle \hat{V}'(x) | \, \mathrm{p}'_c = \delta_y \rangle$ and hence the value of computation. A discretisation based upon Gaussian quadrature was used to approximate the tails more accurately and improve the accuracy of the value of computation calculation. This was found to give a significant improvement in approximation accuracy, with a 5-step quadrature approximation having equivalent accuracy to a 10-step equal allocation. Thus, 5-step quadrature based discretisation has been used for all the experiments in this dissertation. These discrete approximations are illustrated in Figure 5.1.

5.3.5 Discussion

Baum and Smith present a novel way of computing the ESS efficiently for discrete staircase distributions in NEGAMAX game trees by propagating *influence functions* up and down the tree. In a NEGAMAX (Pearl 1984, p 228) tree each state attempts to maximise the negative of its children's values. The influence functions are derived from the linearity of the E{MRP} propagation equation (5.18) and give the marginal change in some linear function of a parent state's PDF, $g(x) = \sum_{v=-\infty}^{\infty} g(v)\, \mathrm{p}_x(v)$, as a linear function of the change at a single leaf, so $\Delta g(x) = \sum_{v=-\infty}^{\infty} I_c^x(v) \delta\, \mathrm{p}_c(v)$, where $I_c^x(v)$ is the influence of c on x. Because the value distributions and influence functions must be computed for *every* state in the local search space this method requires $\mathcal{O}(n_f n_s d)$ time to propagate the influence functions and $\mathcal{O}(n_f n_s d)$ space to hold them, where d is the average depth of the local search space.

The method developed in this section represents a new alternative approach (inspired by Baum and Smith's work) to computing the ESS (and other related marginal value of computation measures). As this method does not require propagation through all the interior nodes of the local search space it requires a factor proportional to the local search space's depth *less* time and space than Baum and Smith's influence function approach. This represents a significant saving for resource limited agents. Further, the derivation presented in this section is more general than Baum and Smith's in that it applies equally well to continuous and discrete distributions. This flexibility allowed the easy development of new and efficient value of computation measures for continuous parameterised distributions.

5.4 Experimental analysis

This chapter has presented five main methods for implementing the search control component of an incremental search agent, two for MINIMIN decision makers; A* and MEVC, and three for E{MRP} and ~E{MRP} decision makers; MEVC, $d\hat{V}(x_0)$ and ESS. This section presents an empirical comparison of these methods to examine if the assumptions they are based upon are justified and assess their relative merits for improving decision making performance.

5.4.1 General methodology

The general methodology used is identical to that of Section 4.9.4.2 except that one of the value of computation measures developed in this chapter is used to select the states to be added to the local search space. The search-control procedures tested are;

A*+AT — Conventional A* search control with minimum heuristic value tie-breaking, and ANTI-THRASH decision making. This gives a base-line for comparison with the results of Section 4.9.4.2.

MEVC+HF — MINIMIN decision making with MEVC search control and hardest first tie-breaking. Hardness is the amount by which a frontier states expected value must change to make it worse than $\hat{V}(\beta)$ for α sub-tree states or better than $\hat{V}(\alpha)$ for non-alpha sub-tree states.

$d\hat{V}(x_0)$ — E{MRP} decision making with discrete distributions and $d\hat{V}(x_0)$ search control.

ESS — E{MRP} decision making with discrete distribution and ESS search control.

$\sim d\hat{V}(x_0)$ — ~E{MRP} decision making with $d\hat{V}(x_0)$ search control.

~ESS — ~E{MRP} decision making with ESS search control.

As before all the algorithms were implemented and tested within a single generic C++ incremental search framework using the same graph construction, heuristic evaluation, and performance assessment code to minimise implementation specific artifacts. Further, all experiments are conducted on the same set of problems as used in Section 4.9 so the results are directly comparable.

5.4.2 Results

The results of these experiments are presented in Figure 5.2 for the puzzle domains and Figure 5.3 for the grid domains. Figure 5.4 also presents examples of local search spaces constructed by the different search control algorithms for a 50x50 grid problem for comparison. Comparing these results with the equivalent results for the purely A* based search control results presented in Figure 4.17 and Figure 4.18 the following features can be identified,

- Significant improvement in decision making performance on the grid problems for the E{MRP} based algorithms, much better than even with full-width search. As shown in Figure 5.4 this is probably due to the improved accuracy of the calibrated future value estimates leading to much better directed local search space construction, which provides more accurate information to the decision making system.

- The indistinguishable but high performance of the E{MRP} algorithms on the "small" 10x10 grid and 8-puzzle problems. In a small problem it quite likely the search will encounter a number of "beacon" states in the same sub-tree, on the edge of the basin of attraction of the goal. The E{MRP} propagation function combines and magnifies the importance of these states to focus decision making and search in this direction (this effect is also shown in Figure 5.4 where the E{MRP} algorithms are much more goal directed than A*). On the small problems this magnification outweighs the differences in search control leading to effectively identical performance.

- Excellent performance of ANTI-THRASH on the puzzle domains, particularly the 15-puzzle where for large numbers of states it is the best of all the algorithms studied, compared to its poor performance on the grid domains. As in Section 4.9 this is attributed to the good heuristic quality of the puzzle domains meaning A* performs particularly well.

- Superior performance of the approximate E{MRP} algorithms on the 15-puzzle. As in Section 4.9.4.2, this is attributed to the ability of the normal approximation to smooth out the noise in the sampled heuristic estimates.

- Relatively poor performance of MEVC+HF, particularly for deeper searches. MEVC clearly performs significantly better than pure A*, particularly on the grid problems, however it is much worse than the other search control algorithms. This is hypothesised to be caused by the myopic assumption of the MEVC becoming less valid for larger searches

causing it to fall back on the hardest-first tie-breaking rule. As hardest-first is effectively A* search control for large search MEVC degenerates to A*.

Overall these results are encouraging but not spectacular. They do show that it is possible to improve significantly on A*'s performance for the strongly biased low quality heuristics of the grid problems. However, when the heuristics are more accurate, such as on the puzzle problems, the benefits of more complex search control are marginal. The superior performance of A* ANTI-THRASH for deep searches on the 15-puzzle is particularly discouraging, as it implies that in this case sophisticated search control is actually detrimental to performance.

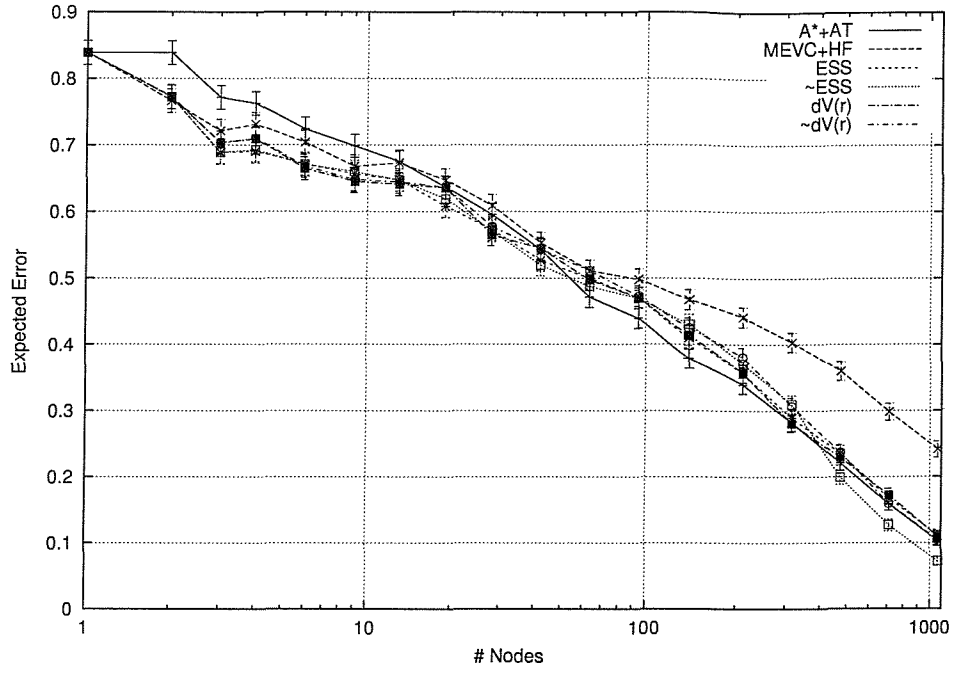
5.5 Summary

This chapter has examined the search-control component of an incremental search agent. The search control component's role is to identify which is the best frontier state to expand next such that it maximises the quality of the agent's decisions and hence its overall performance. It was decided to focus on explicitly meta-rational search control systems, which select states for expansion by maximising an explicitly computed marginal value of computation estimate, because they integrate well with explicitly rational decision makers.

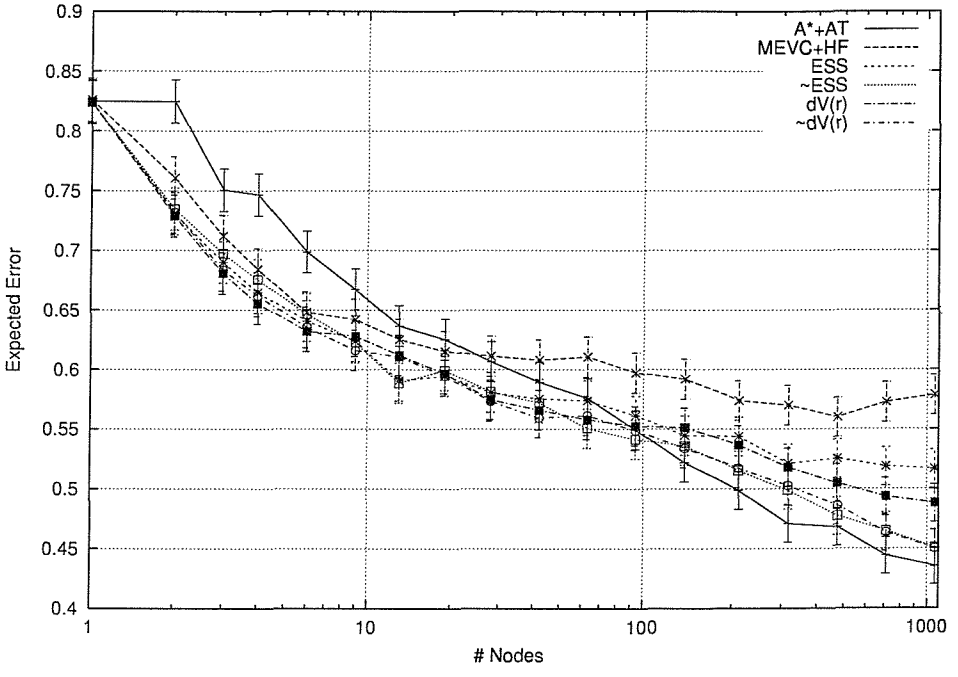
Clearly, the key function in explicitly rational meta-control is computing the value of computation estimates. As computing the true value of computation is intractable, due to the size of the agent's belief space, the main focus of this chapter was on developing tractable hybrid approximations. The value of computation approximations developed are specific to the decision making system because the value of expanding a state depends on its effect on the decision makers decision. The key step in developing a tractable value of information approximation is identifying the propagation function which determines how a particular computation's outcome affects the decision makers action value estimates and hence its decision making. By identifying the propagation functions for the MINIMIN and E{MRP} decision making systems, 5 value of information approximations were developed.

The A* and MEVC approximations for MINIMIN decision makers were developed from previous work, particularly (Russell and Wefald 1989), and a new derivation of the equations for the MEVC's efficient implementation presented. For the E{MRP} decision makers three value of information approximations, MEVC, $d\hat{V}(x_0)$, and ESS, were developed. It was then shown how these approximations could be efficiently computed using a new method for the efficient computation of $\langle \hat{V}'(x) | \, \mathrm{p}'_c = \delta_y \rangle$. It was also shown how this derivation could be readily extended to work with continuous distributions, and hence ~E{MRP} decision makers.

The performance of the search control systems developed in this chapter was assessed experimentally on the same 4 problem types as used in Chapter 4. The results of these experiments were encouraging in that they demonstrated that E{MRP} based decision making and search control has clear advantages for low quality highly biased heuristics. However, the

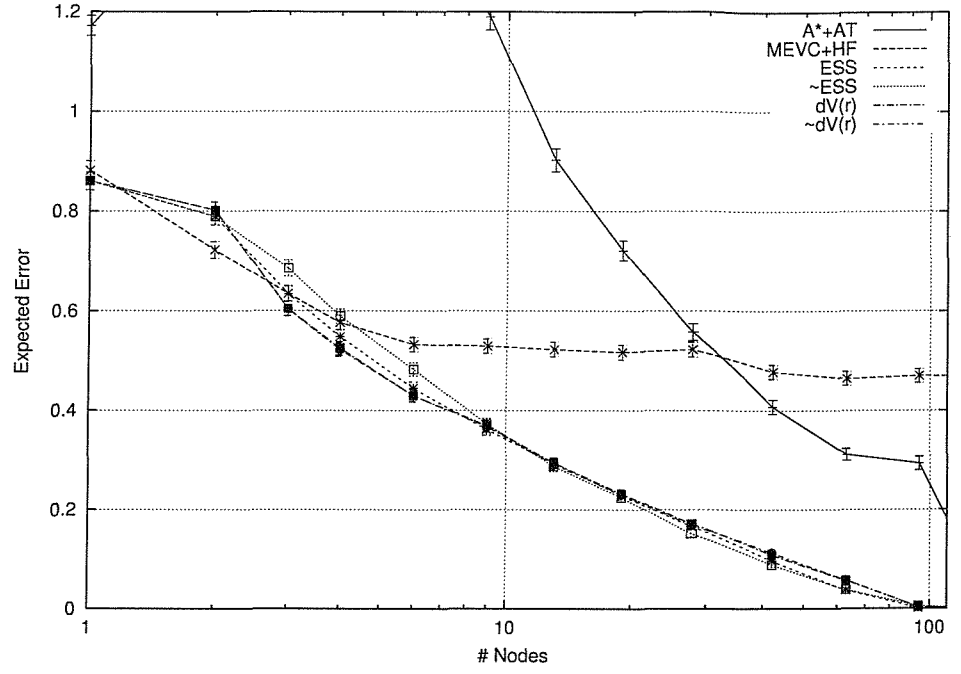


(a)

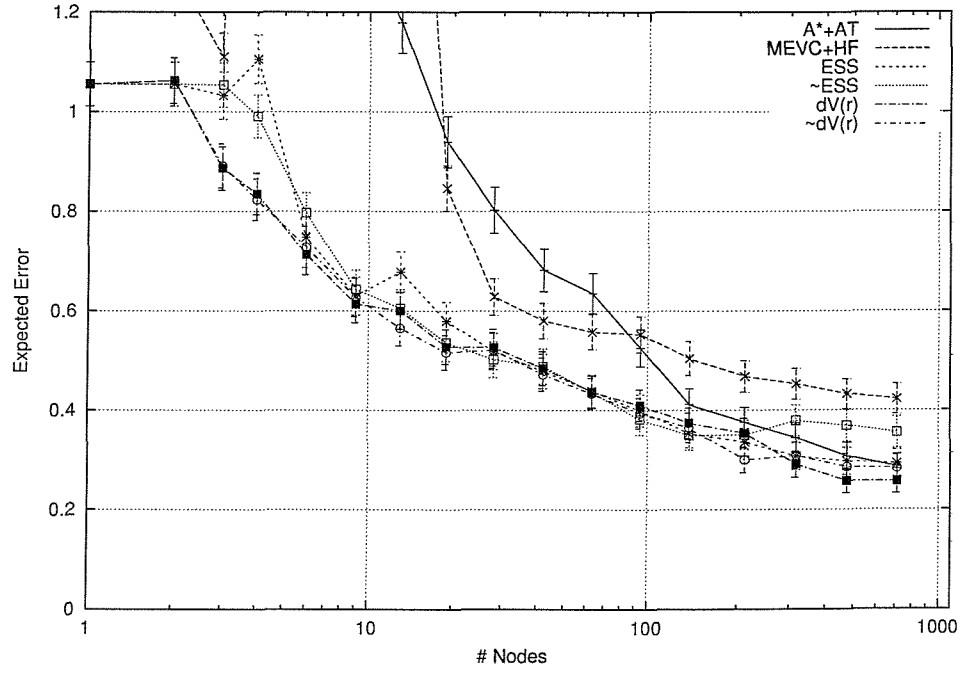


(b)

FIGURE 5.2: Search-control methods expected performance profiles on the (a) 8-Puzzle and (b) 15-Puzzle. All results are averaged over 3,000 problems. The error bars indicate the mean's sample variance. Note the log-scale on the x-axis.

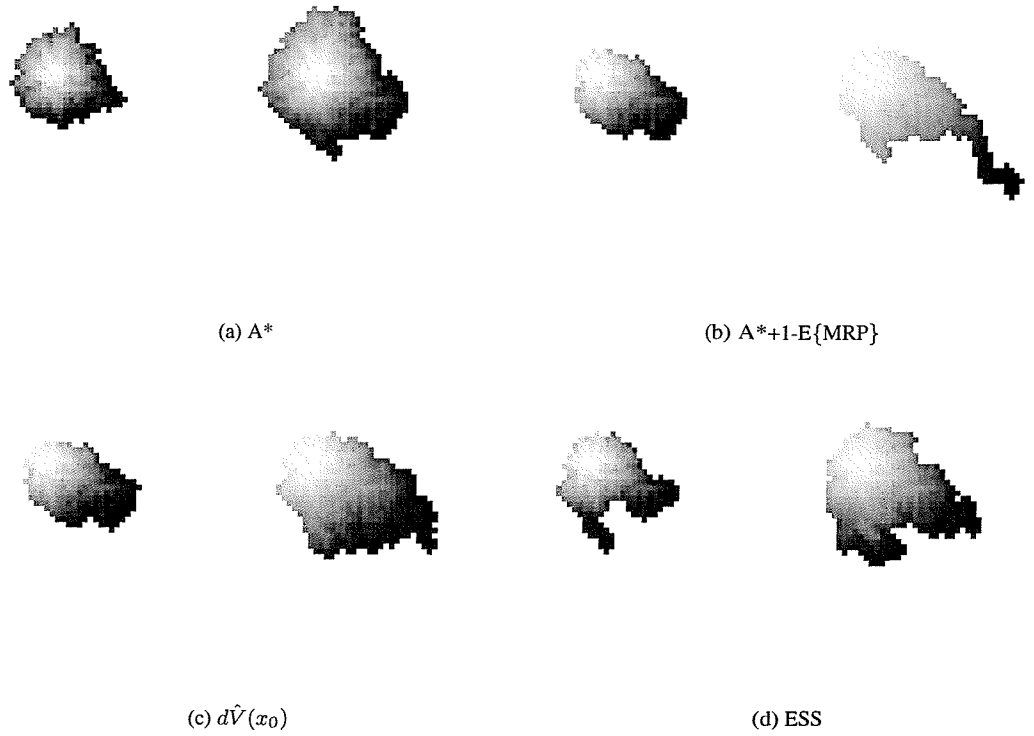


(a)



(b)

FIGURE 5.3: Search-control methods expected performance profiles on the (a) 10x10 grid and (b) 100x100 grid. The results are averaged over 10,000 problems for the 10x10 grid and 2,000 problems for the 100x100 grid. The error bars indicate the mean's sample variance. Note the log-scale on the x-axis.



(a) A* (b) A*+1-E{MRP}

(c) $d\hat{V}(x_0)$ (d) ESS

FIGURE 5.4: This diagram shows the local search spaces constructed by the different search control strategies after 200 and 400 state expansions for a single 50x50 grid problem with the goal in the bottom right. Shading indicates distance from the start state. Part (a) shows A* which is clearly the least goal directed algorithm, part (b) shows A* with calibrated heuristics which increase goal direction but may cause the search to get lost down a "blind-alley" as shown in the second picture. Parts (c) and (d) show E{MRP} based search control which is more goal directed that A* but doesn't get lost like calibrated A*. Part (d) shows that ESS is more goal directed that $d\hat{V}(x_0)$.

superior performance of ANTI-THRASH decision making on the 15-puzzle demonstrates there is still some way to go in developing efficient search control systems.

Chapter 6

Putting it all together: Stopping and Convergence

The previous chapters have developed the decision making (Chapter 4) and search control (Chapter 5) components of an incremental search agent. This chapter ties these two components together to make a complete incremental search agent. This requires the addition of a meta-control system to terminate the search at the correct point and learning capabilities to prevent cycles and ensure convergence towards the goal.

The performance of ANTI-THRASH in the results of Section 4.9.4.2 and Section 5.4.2 clearly demonstrate that stopping the search at just the right point can have a significant effect on decision making performance. ANTI-THRASH simulates a normal A* search with MINIMIN decision making which is stopped immediately before the search depth increases. Therefore it is hoped that effective stopping will similarly improve the performance of the decision making and search control algorithms developed previously.

Learning and adaptation is required to provide the incremental search agent with some memory so it can avoid repeating its previous mistakes and cycling forever. Learning can also improve an incremental search agents real-time performance by allowing it to avoid repeating work by caching previous results.

6.1 The stopping problem

The general stopping problem is;

Definition 6.1 (General stopping problem). Given a decision making system and a search control system, choose when to stop working on the current decision so that, on average, the agents total task reward is maximised.

The stopping problem is concerned with scheduling the agents computational resources over the entire problem solving episode. In fact, the stopping problem can be modelled as a macro-control problem (Section 3.3.11) where the agent must decide when to interrupt the current extended duration decision-making macro (consisting of search control and decision making) and switch to the next one. This macro control problem is complicated by the fact that the quality of the macros output and resources consumed executing it influence the performance of subsequent decision-making macros. Thus, given an appropriate model of the decision-making macros operation and the resource cost, any of the macro-control algorithms discussed in Section 3.3.11.2 could be used to solve the stopping problem.

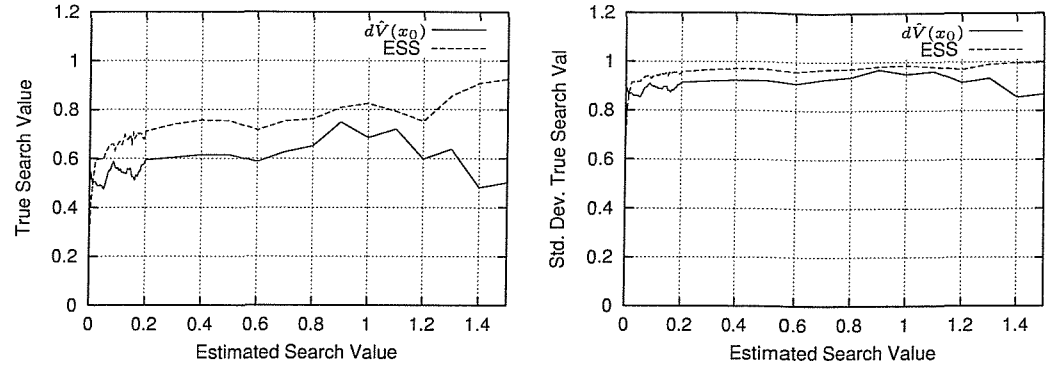
As discussed in Section 3.3.11 existing solutions to the stopping problem fall into two main camps; (1) META-GREEDY solutions based upon approximations for the marginal value of further computation (Russell and Wefald 1989; Horvitz 1990; Baum and Smith 1997), and (2) solutions based upon solving the macro-control MDP using dynamic programming (Hansen and Zilberstein 2001a) or reinforcement learning (Harada and Russell 1999; Bulitko, Levner, and Greiner 2002; Markovitch and Sella 1996). In this dissertation, because the explicitly meta-rational search control systems developed in the previous chapter inherently compute marginal value of computation estimates, only META-GREEDY solutions are considered. Studying how dynamic programming or reinforcement learning could be used to improve stopping problem solution quality is an area of ongoing research.

6.1.1 Meta-greedy stopping policies

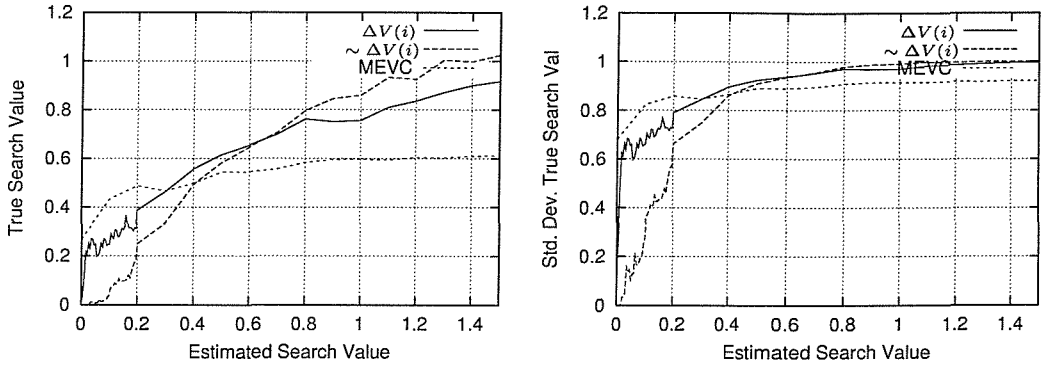
As defined in Section 3.3.11.2 a META-GREEDY stopping policy stops execution of the current decision-making macro once the estimated marginal value for further computation, $\Delta V(c)$, is less than the marginal cost of computation, $\Delta C(|c|)$. Thus, implementing a META-GREEDY stopping policy requires estimates for both the marginal value and cost of computation.

The most obvious approach to estimating the marginal value of computation is to use one of the approximations developed in the previous chapter—that is either the MEVC, $d\hat{V}(x_0)$ or ESS approximations. The problem with this approach is that these estimates can be very unreliable and (particularly the MEVC) significantly underestimate the value of further computation, which can cause META-GREEDY to stop the search prematurely. Therefore, a more reliable and cautious marginal value of computation estimate is used for the stopping rule than for search control.

Russell and Wefald's (1989) solution to this problem is to compute estimates for the value and costs of a set for state expansions and stop when the cost of this set is greater than its benefit. The sets value is estimated using the MEVC and the cost estimated as $C(|nb^d|)$ where, C is the cost function, n is the number of states expanded, b domains estimated branching factor, and d the depth to which the node is to be expanded. They construct the set of computations considered incrementally using the hardest-first criterion of Section 5.2.1. Baum and Smith's solution is



(a)



(b)

FIGURE 6.1: Example results of the various methods for estimating the value of computation as predictors of true decision quality, and hence the value of further search, found by averaging over 3,000 8-puzzle problems. Part (a) shows the results for the single step estimators, $d\hat{V}(x_0)$, and ESS (the continuous versions performed similarly). Note, the single step MEVC results are not shown as they were all 0. Part (b) the results for the all further computation estimators MEVC and $\Delta V(i)$. The results show that whilst both types of estimate are very poor estimators of the true value (as shown by their high variances) the all further computation estimators are much better in that they are at least smoothly increasing with increasing error.

to use $\Delta V(i) = \langle \hat{V}(x_0) \rangle - \langle \hat{V}(\alpha) \rangle$ (defined in Section 5.3.3), which they call $U_{\text{all leaves}}$, to estimate the expected value of all further computations. They estimate the cost of performing all the computations necessary to realise this value using a learnt function which takes account of the "opportunity cost" of continuing to work on the current problem as opposed to starting the next one.

Figure 6.1 shows that these non-myopic MEVC and $\Delta V(i)$ estimates are considerably better predictors of the true search value than the single computation step estimators, $d\hat{V}(x_0)$, and ESS. In these results the MEVC estimates are computed for MINIMIN decision making based upon expanding *all* the frontier states in the α sub-tree. The combined future value distribution for the expanded states is found assuming frontier value independence using (4.25). Notice, that all the estimators tend to significantly *overestimate* the value of further computation, as the

curves are consistently above the $x = y$ line. Further, the high variance of the non-myopic estimates indicates they are still very noisy predictors of the true search value.

Now, the non-myopic MEVC or $\Delta V(i)$ estimates compute the benefit of all future computations. However, the META-GREEDY stopping rule is based upon the relative cost/benefit of computation. Therefore to use these estimates we require either; (1) an estimate for the cost of the computation required to realise this value, or (2) an estimate for the rate of change of this value. In this work both approaches have been tried.

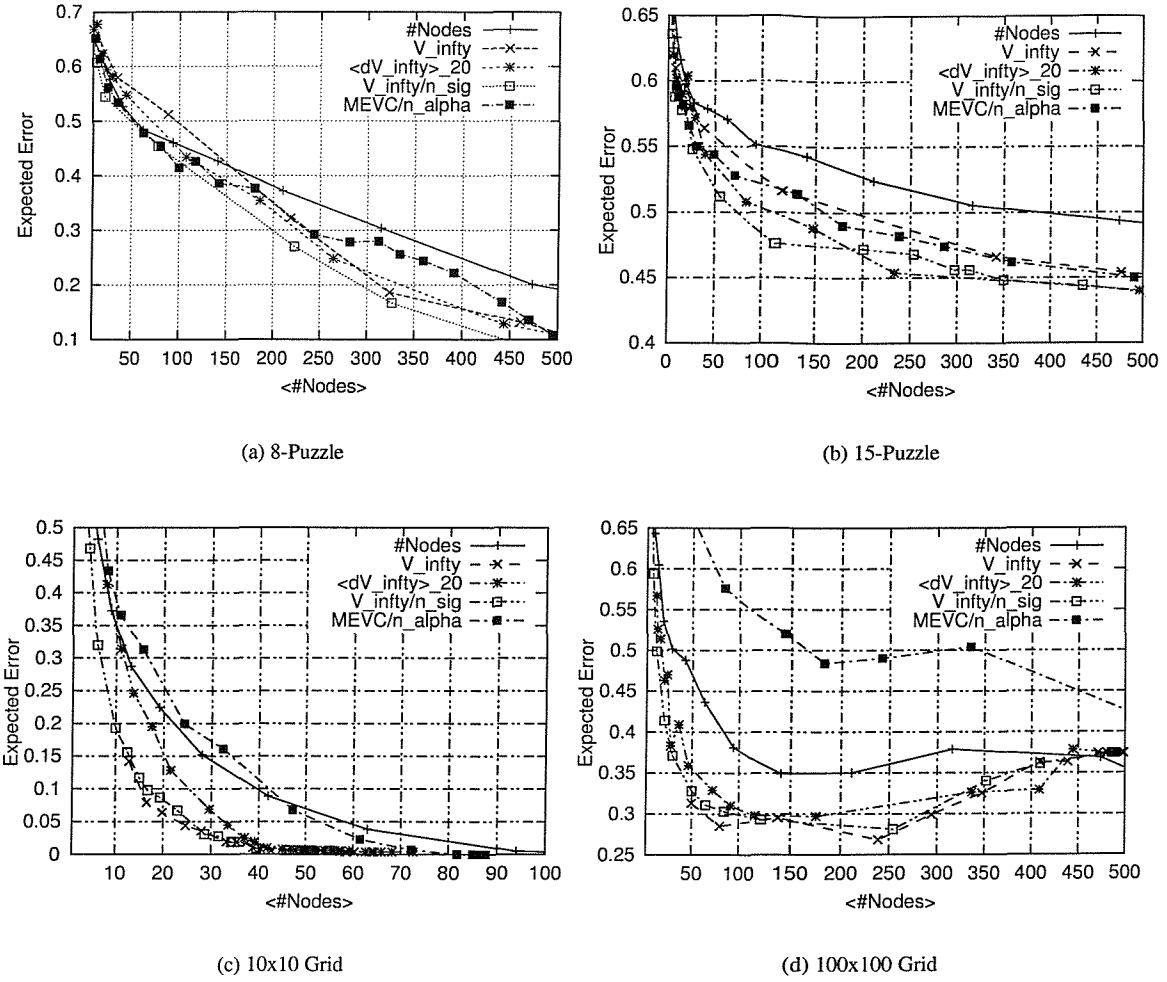
The first approach taken is to estimate the number of nodes which must be expanded to realise the value of all further computations, and hence the cost of this computation. In many cases most of the local search spaces frontier nodes have little or no influence on decision making. Thus, assuming all frontier nodes must be expanded would tend to overestimate the computation required. Instead, the amount of computation required is estimated by simply counting the number of nodes with greater than zero expansion value, as computed by the ESS or $d\hat{V}(x_0)$ estimates, denoted n_{sig}. If search control provides good estimates for the value of individual computations this should be a good estimate of the number of significant nodes.

The second approach directly estimates the local rate of change of future value, $dV(i)/dt$, using a k step moving average of the *absolute* changes in the total future value estimate. That is,

$$\frac{dV(i)}{dt} \approx \frac{1}{k} \sum_{t=n-k}^{n} \left| \hat{V}_{c_\infty}(t) - \hat{V}_{c_\infty}(t-1) \right|, \tag{6.1}$$

where n is the current expansion number, k is the length of the averaging window, and $\hat{V}_{c_\infty}$ is the non-myopic value of future computation estimate used (either MEVC or $\Delta V(i)$). The moving average filter is required to smooth out the noise in the underlying signal, which the gradient estimate tends to magnify. The absolute change in $\hat{V}_{c_\infty}$ is used because both increases and decreases in the value of further computation are deemed valuable. Increases indicate that our previous estimate was wrong so we shouldn't stop whereas decreases indicate that we are moving towards being sure of our decision. In practice using the absolute value tends to stop the search terminating early on when the value estimates are still very uncertain.

Throughout this dissertation the cost of a single computation is assumed to be constant, $C(1)$, for the duration of the current decision. This constant indicates the relative importance of minimising computational effort verses maximising the intrinsic value of the solution computed. For example, $C(1) = 0.1$ indicates that a reduction in the solution cost of 1 is worth 10 node expansions and $C(1) = 0.001$ that a unit cost reduction is worth 1,000 node expansions. Thus when the cost of computation is estimated by counting the number of significant nodes, n_{sig}, the META-GREEDY stopping rule is simply to stop when $\hat{V}_{c_\infty} < n_{\text{sig}} C(1)$ which is equivalent to stopping when $\frac{\hat{V}_{c_\infty}}{n_{\text{sig}}} < C(1)$. Similarly, when the value of one more node expansion is estimated directly the META-GREEDY stopping rule is to stop when $\frac{d\hat{V}_{c_\infty}}{dt} < C(1)$.



(a) 8-Puzzle

(b) 15-Puzzle

(c) 10x10 Grid

(d) 100x100 Grid

FIGURE 6.2: Representative comparative performance results for stopping rules based upon $\sim\Delta V(i)$ for; (a) the 8-puzzle, (b) 15-puzzle, (c) the 10x10 grid and (d) the 100x100 grid domains. The results show the expected decision quality verses the expected number of node expansions required, with each point representing a particular threshold tested. The MEVC/n_alpha results were found using hardest first (HF) search control and MINIMIN decision making. The results using A* search control were almost identical. All other results were found using the $\sim$ESS search control procedure and $\sim$E{MRP} decision making. The results for the other E{MRP} based search control functions were similar.

6.1.2 Experimental analysis

The relative performance of the different stopping rules was tested by running the appropriate search-control and decision making algorithm on a random selection of test problems for a wide range of computational costs and recording the error in the final decision at the point the stopping rule would have terminated the search. The stopping rules tested were,

#Nodes	stop when a certain number of nodes have been expanded.
$\hat{V}_{c_\infty}$	stop when $\hat{V}_{c_\infty} = \Delta V(i)$ becomes less than $C(1)$ (labelled V_infty).

$\hat{V}_{c_\infty}/n_{\text{sig}}$ stop when $\hat{V}_{c_\infty}/n_{\text{sig}}$ drops below $C(1)$, where n_{sig} is the number of frontier states with non-zero estimated expansion value (labeled V_infty/n_sig).

$\langle|d\hat{V}_{c_\infty}/dt|\rangle_{20}$ stop when the 20 step moving average of the absolute change in $\hat{V}_{c_\infty}$ drops below $C(1)$ (labelled <|dV_infty|>_20). A 20 step average was chosen after initial experiments indicated this gave the best performance.

MEVC(α)/n_α stop when MEVC(α)/n_α drops below $C(1)$, where MEVC(α) is the myopic EVC estimate for expanding all frontier nodes in the α sub-tree, and n_α is the number of α's frontier states. (labelled MEVC/n_alpha).

For the #Nodes stopping rule thresholds of; 2, 5, 10, 20, 50, 140, 200, 400, 600, 800 and 1000 nodes were used. All other stopping rules were tested with the computational costs, $C(1)$'s, ranging from 0.005 to 2. Thus, the results measure stopping performance across a wide range of computational resource availabilities. Representative results from these tests are presented in Figure 6.2. In these tests MEVC+HF (Section 5.2.1) search control was used with MINIMIN decision making for the MEVC/n_alpha results. The results for A* search control were almost identical. All other results used the ~ESS search control method of Section 5.3.3 with ~E{MRP} decision making. The results for the other E{MRP} based search control techniques, i.e. ESS, $d\hat{V}(x_0)$ and $\sim d\hat{V}(x_0)$, were similar.

These results clearly show that an intelligent stopping test can significantly improve the performance of the search control and decision making system. Reducing the average effort required by E{MRP} decision makers to reach a certain performance level compared to node based stopping by up to 20% on the 8-puzzle, a factor of 5 or more on the 15-puzzle, and about a factor of 2 on the grid problems. For MINIMIN decision makers with MEVC+HF search control MEVC(α)/n_α stopping reduces search effort below that required by ~ESS with node based stopping for all but the 100x100 grid. Compared to node based stopping the results of Section 5.4.2 show this is a significant reduction in effort. That all the $\Delta V(i)$ based stopping rules gave similar performance improvements, indicates once again that $\Delta V(i)$ is a reasonably good indicator of the true value of further search. However it is unclear which stopping rule is best over all. On the smaller 8-Puzzle and 10x10 grid problems the $\hat{V}_{c_\infty}/n_{\text{sig}}$ estimate gives generally best performance, particularly for low numbers of nodes. The reduced performance of $\langle|d\hat{V}_{c_\infty}/dt|\rangle_{20}$ for low numbers of nodes on these small problems is attributed to the averager introducing a lag which prevents the system stopping as early as it should in some cases. On the larger 15-Puzzle and 100x100 grid problems identifying the best stopping rule is more difficult. $\hat{V}_{c_\infty}/n_{\text{sig}}$ still appears to be best for less than about 50 nodes. However, above this point $\langle|d\hat{V}_{c_\infty}/dt|\rangle_{20}$ dominates in the 15-puzzle and $\hat{V}_{c_\infty}$ for the 100x100 grid. The reducing performance of the E{MRP} algorithms for greater than about 250 nodes on the 100x100 grid demonstrates that improved stopping cannot always make up for the decision making algorithms problems, as shown in Figure 5.3(b) where ~ESS decision making also has worsening performance at this point.

Due to its better performance with shallower searches, and reasonable performance in all other cases the $\hat{V}_{c_\infty}/n_{\text{sig}}$ stopping rule is used in the remainder of this dissertation.

6.2 Learning and cycle avoidance

Up to this point the analysis has focused on ensuring that the agent's individual decisions are as near to optimal as possible given the agents current beliefs. The assumption so far has been that by making the best locally rational decisions the agent will make the best sequence of decisions. This assumption is fatally flawed. Locally rational decision making is effectively just greedy hill-climbing and hence prone to getting stuck in local optima. For an incremental search agent which must execute some action such local optima manifest themselves as the agent cycling repeatedly through the same sequence of states. Thus, a complete incremental search agent requires some mechanism to "kick" it out of local minima, and send it on its way towards the goal. This section discusses how learning and memory can be used in this way to guarantee the agent eventually convergences on a solution.

The importance of cycle avoidance can be demonstrated by considering results from using an incremental search agent based upon a depth 10 A* search on the 15-puzzle. Without cycle avoidance this agent solved less than 10% of 1,000 test problems in 1,000 steps or less. With a simple cycle avoidance mechanism (using the α update method described later) the same agent can solve all 1,000 problems in less than 1,000 steps.

There is a vast literature on techniques for escaping local optima in local search, including:

1. Randomised decision making to break cycles, such as used in simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983).
2. Trajectory traces, such as Tabu lists (Glover and Laguna 1997), used to directly penalise re-visiting places the agent has been before.
3. Value function refinements used to adapt the approximate value function to remove local minima. Examples include the Bellman backups used in real-time dynamic programming (RTDP) (Barto, Baradtke, and Singh 1995) algorithms such as Korf's LRTA* (Korf 1990), or the adaptive penalty terms used in reactive search (Battiti 1996).

Current incremental search algorithms have tended to use a combination of randomised tie-breaking and RTDP based value function refinements for cycle avoidance and convergence guarantees. This is the approach followed in this thesis.

Value function refinements can improve the agent's performance in two ways,

1. Improve short term performance within a problem by helping avoid cycles.

2. Improve long term performance across problems by improving the quality of the approximate value function.

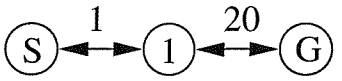
As Korf (1990) points out in his original RTA* paper the same value function updates cannot fulfil both these objectives. The purpose of updating a states value estimate is to revise it to better reflect the true future value of any trajectory starting from the state. Now, if we re-visit a state during a single problem solving episode then this means that the agents old estimate for the future value was wrong, or it would have solved the problem. However, if the agent re-visits a state in a different problem solving episode then the value of the path it took previously may be correct. Thus the two objectives require different types of update; cycle avoidance updates should be based upon the pessimistic assumption that the path followed from the state was wrong, and long term performance optimising updates should be based upon optimistically assuming that the path followed from the state was right. To achieve both objectives simultaneously requires two updated values for each visited state; a pessimistic update which is only used for the current problem and an optimistic one which is retained across problems.

Using only one type of update can result in reduced agent performance. In the RTA* algorithm cycle avoidance is effected by updating the root states value estimate with the revised MINIMIN value of the *second best* action, $\hat{V}(x_0) \leftarrow \hat{V}(\beta)$. As explained below this is a correct cycle avoidance update if the state space is a tree. However, retaining this updated value across problems may lead to reduced performance as the updated values will be pessimistic for states for which the *best* action, α, was the right action to take. To improve performance across problems LRTA* updates the root states value estimate with the revised MINIMIN value of the *best* action, $\hat{V}(x_0) \leftarrow \hat{V}(\alpha)$. Providing the initial value estimates are *optimistic* and the optimal values are finite, this update is guaranteed to converge the value estimates to the optimal values. This was initially proven by Korf for LRTA* and later generalised to any trial based RTDP algorithm using a Bellman backup procedure (Barto, Baradtke, and Singh 1995). However, as Korf points out and illustrated in Figure 6.3, this update can reduce performance within a single problem.

In this dissertation only single problem solving episodes are considered, so we focus on the use of value updates to escape from local minima. Using a second update procedure to improve performance across problems is a possible area for future research.

6.2.1 Cycle avoidance value updates

Even when only considering how to optimise single episode problem solving performance determining the best value function update is more difficult than would at first appear. In tree structured state spaces $\hat{V}(\beta)$ update is correct, as noted above. To see why consider the simple problem shown in Figure 6.4. Because the problem is tree structured (ignoring for now the dotted line from c to S), after leaving S via α in order to re-visit S the agent must again pass



	$\hat{V}(\alpha)$ updates			$\hat{V}(\beta)$ updates		
step	$\hat{V}(S)$	$\hat{V}(1)$	$\hat{V}(G)$	$\hat{V}(S)$	$\hat{V}(1)$	$\hat{V}(G)$
0	0	0	0	0	0	0
1	0←1			0← ∞		
2		0 ←2			0←20	
3	1←3					solved
⋮						
21		19←20				
22			solved			

FIGURE 6.3: This diagram shows how using Bellman value function updates can actually result in *worse* problem solving performance the first time a problem is encountered. The agent starts in state S and has to get to state G for minimal cost. The table shows the updates performed to the current state and trajectory followed for the two update types. Clearly $\hat{V}(\beta)$ update solves the problem in 2 steps for an optimal cost of 21, whereas $\hat{V}(\alpha)$ takes 22 steps for a total cost of 42.

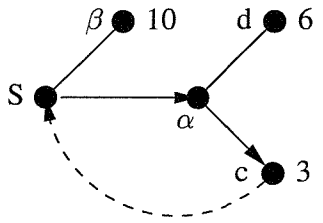


FIGURE 6.4: This diagram demonstrates the problems with using $\hat{V}(\beta)$ for cycle avoidance on a graph. The agent is in state S and the local search space is shown by the solid lines, the numbers indicate the states future value estimate, so the best frontier nodes is c. Consider the case when the agent moves to c and finds that it can travel back to S along the dotted line. As d's frontier value of 6 has not yet been proven incorrect it should consider this as the future value of S. Thus, updating S with the β sub-tree's value of 10 is incorrect in this case.

through α. Now it is clearly sub-optimal to transition from α to S and then back to α, hence it is only worthwhile re-visiting S if it is better to leave it again via some other action. Thus, the value of the best trajectory after re-visiting S is the value of the best action from it *excluding* α, which is by definition $\hat{V}(\beta)$. However, if the state space is a graph, the agent can potentially re-visit S without first passing through α, as shown by the dotted line in Figure 6.4. This (literally) short-circuits the above reasoning as now it may be worthwhile re-visiting S to leave it again via α, for example if this is a better path to d. Thus, $\hat{V}(\beta)$ is no longer the best trajectory to follow after re-visiting S, and the $\hat{V}(\beta)$ update may be overly pessimistic. On a graph the only thing one can be sure of on revisiting a state is that the *best* frontier value from that state was wrong.

Thus, the cycle avoidance updates used on graphs tend to be based upon "rules of thumb" which were found to give good performance in the past. The following lemma shows that there is great flexibility in choosing the update procedure.

Lemma 6.1 (Convergence on ergodic domains). *For a finite ergodic domain, where every state is reachable from every other state, providing the value of very node is initially finite and*

made strictly worse every time it is visited by a finite amount bounded below by some constant then a greedy incremental search agent will eventually reach the goal.

Proof. Since the value of nodes in a cycle becomes progressively worse each time round the cycle eventually one of them will become worse than the fixed value of a node not in the cycle, causing the agent to leave the cycle. Further, as the updated state values are always finite, in a finite state space with finite initial state values the agent must eventually visit every state, including the goal state. To do otherwise implies the agent would have to traverse forever some finite cycle of states, which is impossible if the values of states not in the cycle are finite. □

This lemma is based on Russell and Wefald's Lemma 3 (1989, p 126) which in turn is a generalisation of Korf's original (1990) proof. Thus, an incremental agent will eventually solve any finite problem so long as a value update always makes node values worse by a finite amount.

This is not to say that all value updates are equal, the optimal update should make a node just bad enough to stop cycles but not so much that it prevents revisiting a state if it turns out the α move was a wrong turn. In the absence of any other guidance the value update rules used in this dissertation for the different incremental search systems have been selected based upon an empirical analysis of a range of possible rules.

6.2.1.1 Cycle avoidance for MINIMIM decision makers.

Russell and Wefald (1989) performed an extensive empirical analysis of a number of value update rules for MINIMIN decision making. Update rules they tested included, (1) α update with the MINIMIN value of the best action. (2) β update with the MINIMIN value of the second best action. (3) $\alpha \times 1.1$ update with the MINIMIN value of the best action plus 10%, (4) $\alpha + 1$ update with the MINIMIN value of the best action plus 1, (5) $\beta + 1$ update the MINIMIN value of the second best action + 1. Notice, that all these rules obey the requirements of Lemma 6.1 and so guarantee eventual convergence. In their experiments on the 15-puzzle they found that the $\alpha \times 1.1$ update gave the best results closely followed by $\alpha + 1$ update.

I have repeated this empirical study but based upon the above analysis of the problems with the $\hat{V}(\alpha)$ and $\hat{V}(\beta)$ updates on graphs have included an additional update, denoted β-leaf, which updates the root's estimate with the second best frontier state's value. β-leaf is a valid update for graph structured domains. Further, a depth limited A* search has been used to construct the local search space rather than RTA*'s α-pruned search as used by Russell and Wefald (1989). As a final efficiency measure the A* search is terminated early if it tries to re-expand a previously visited state. This additional termination criteria is justified by noting that that because a fixed

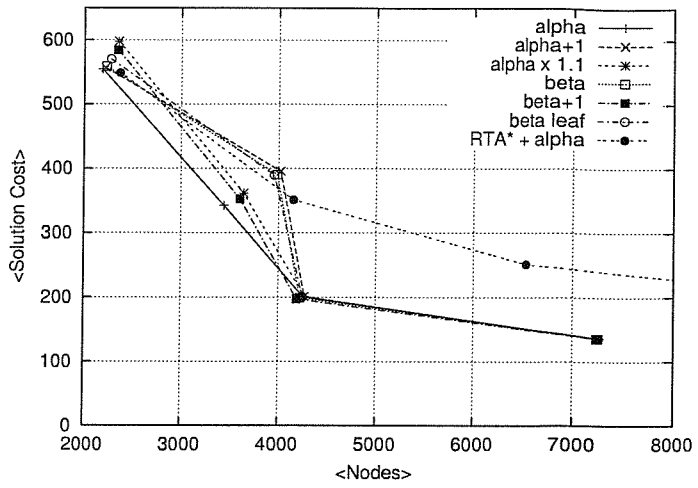


FIGURE 6.5: 15-Puzzle experimental results for various update rules. The incremental search agent used a depth limited A* search control procedure to construct the local search space and MINIMIN decision making. Each point represents the average number of nodes expanded and deviation from optimality over the same 1,000 randomly generated problems for depth bounds of 3, 5, 7 and 10. Searches were terminated at 1,000 steps and deviations of 1,000 steps recorded.

depth search expands interior states less deeply than root states the re-expansion of a previously visited interior state is unlikely to make it look worse than its previously updated value.[1]

The results of these experiments for the 15-puzzle are presented in Figure 6.5. These results clearly show the superiority of the depth limited A* search over RTA*'s α-pruned search, particularly for deeper searches. The reduction in node expansions is to be expected as A* search will generally construct much smaller local search spaces than RTA*'s α pruning, with the degree of reduction increasing with increasing search depth.[2] That depth limited A*'s solution quality is better than RTA*'s at the same depth bound with the same update rule (for example at depth 7 A*'s average solution length is $\approx$200 whereas RTA*'s is $\approx$250) is slightly more surprising as in the vast majority of cases both algorithms make the same action selections. The only differences in this case come down how the algorithms break ties between states with the same value. RTA* breaks ties randomly, whereas A* picks the state with the lowest heuristic value. As noted previously, when heuristic error is proportional to heuristic value, A*'s strategy is equivalent to breaking ties towards the state with best expected future value. Thus, A*'s superior performance indicates using some information about the errors in the agents current estimates is useful for improving performance.

[1]Russell and Wefald (1989) erroneously state that a reduced depth re-expansion *cannot* improve a states value estimate. This is incorrect because the interior states neighbours may have had their values updated after it last had its value updated. These revised values could then be used to improve the interior states estimate. To see how this can occur, consider again Figure 6.4. If S is updated with the value of 6 from the second best frontier state and in the process of reaching c the value of α is updated to be greater than 10 then the correct future value for S is now 10, the value of the β sub-tree. Thus, in this case, re-expanding S can increase its value, potentially preventing an additional cycle round the $S \rightarrow \alpha \rightarrow c$ loop.

[2]Note, these results cast some doubt on the results reported in (Russell and Wefald 1989) where they claim that using DTA* on the 15-puzzle results in "a total solution cost about 25% less on average", than SRTA*. However, as SRTA* is a variant of RTA* which uses α pruning and DTA* uses A*, these results indicate that the majority of this improvement is likely due to the change in search control rule rather than DTA*'s decision theoretic stopping rule.

These results also show that for depth limited A* search whilst α update gives the best overall performance, the update function is not particularly important for search depths greater than 7. This is in stark contrast to Russell and Wefald's results which showed a strong update function dependence at all depths with $\alpha+1$ and $\alpha \times 1.1$ giving best performance. This is again attributed to A*'s tie-breaking strategy which, because it avoids states with high heuristic values, will tend to avoid previously visited states more strongly than RTA*. Thus, a state's value does not have to be made as much worse to prevent cycling, reducing the cycle avoidance advantages of increasing a states value more than strictly necessary. However, increasing a state's value more than necessary can still introduce additional cycles, causing the overall effect of the non-α update rules to be a reduction in solution quality for shallow depths. This effect is reduced at deeper depths as the agent is less likely to make a wrong turn and need to re-visit a previously visited state. The fact that α update performs better than β-leaf update implies that the situation described in Figure 6.3 does not occur frequently in the 15-puzzle. The remaining variation in solution performance for different update rules for low search depths can be attributed to the different updates influencing A*'s tie-breaking rules in different ways.

As it gave the overall best performance α update will be used for cycle avoidance with depth limited A* search and MINIMIN decision making in the remainder of this dissertation.

6.2.2 Cycle avoidance for E{MRP} decision makers.

Determining a good cycle avoidance function for the E{MRP} estimators poses two particular problems. Firstly because decision making is based upon distributions we must identify an appropriate distribution with which to update the root state. Secondly, because a states value distribution may be optimistic or pessimistic with respect to its true value, it is easily possible that a local search indicates the root states true value is actually *better* than its current one. Thus, using this value to update a node is not guaranteed to prevent cycles as it violates the requirement of Lemma 6.1 that a states value get worse each time it is visited. In fact, updating a nodes value in this way may induce *more* cycles than no update, as the updated roots value may be better than its immediate children causing greedy action selection to return to the root state immediately after leaving it.

A number of possible distribution update rules were developed and tested, including; updating with the E{MRP} estimate for the best (α) sub-tree, updating with the E{MRP} estimate for the second best (β) sub-tree, updating with the future value distribution from the frontier state with best expected future value, or using any of these updates plus an addition constant to ensure the updated value gets worse every cycle. Initial experiments showed that because the sub-tree based updates included the effects of all frontier states in the sub-tree they tended to produce optimistic updated value distributions, which induced additional cycling. Using the best frontier states value distribution suffered from the same problem to a lesser extent but still induced some cycling when its value distribution was better than the root state's children. To totally remove

cycling required including an additional constant of about the maximum single action cost to ensure the roots updated value got worse after each visit.

Whilst not a totally satisfactory solution, the approach of using the future value distribution of the best frontier state plus an additional constant was found to give the best performance, and is used with E{MRP} decision making in the remainder of this dissertation. Investigating more principled cycle avoidance update rules for use with distribution based estimators is clearly an area for future work.

6.3 Experimental Analysis

This section presents comparative performance results for complete on-line planning agents consisting of; decision making, search control, stopping and cycle avoidance components. As such it represents the culmination of the incremental search agent development presented in this thesis and a final test of whether the effort expended on developing sophisticated decision theoretic decision making and meta-control methods is worthwhile.

The experiments were conducted using the same general methodology and C++ implementation framework as used in Sections 4.9.4.2 and 5.4, where the system was presented with a set of randomly generated problems from each domain. The problems were actually the same ones uses throughout this thesis as the same generation procedure was used. The main difference from the preceding experiments is to chain the individual decisions together by executing the action recommended by the decision making system after the stopping rule terminates the search. The root state is then updated with the result of the action and a new incremental search procedure started. Before the new incremental search is begun all previously acquired state space information, except for the root's updated value, is deleted. Thus decision making is essentially memory-less. This process of decide and execute is repeated until either the agent transitions into a goal state, or a domain dependent maximum number of action executions is reached. The limits uses were; 800 for the 8 puzzle, 1,000 for the 15 Puzzle, and 800 for the grid problems.

The algorithms tested were;

A*+MD Depth limited A* search for local search space construction. Decision making using the MINIMIN child value estimate (Section 6.2.2) and minimum heuristic value tie-breaking. $\alpha + 1$ update (Section 6.2.1.1) is used for cycle avoidance.

HF+MEVC Hardest first (Section 5.2.1) search control with MEVC$(\alpha)/n_\alpha$ (Section 6.1.1) stopping used for local search space construction. Decision making as for A*+MD. Cycle avoidance using the best frontier states MINIMIN value worsened by a

problem specific constant (1 for the puzzle domains and 10 for the grid domains).[3] This is essentially Russell and Wefald (1989) DTA* algorithm.

ESS — ESS (Section 5.3.3) search control with a $\hat{V}_{c_\infty}/n_{\text{sig}}$ (Section 6.1.1) stopping rule used for local search space construction. Decision making based upon the E{MRP} (Section 4.8.1) child value estimates. The best expected value frontier states value distribution worsened by an problem specific constant (5 for the puzzle domains and 10 for the grid domains) root update is used for cycle avoidance (Section 6.2.2).

$\sim$ESS — As for the ESS but using Gaussian distributions to approximate state future value distributions and the Gaussian approximation based algorithms; $\sim$ESS for value of computation based search control (Section 4.8.3).

$d\hat{V}(x_0)$ — As for the ESS but using the $d\hat{V}(x_0)$ value of computation estimate based search control (Section 5.3.2).

$\sim d\hat{V}(x_0)$ — As for $\sim$ESS but using the $\sim d\hat{V}(x_0)$ value of computation estimate based search control.

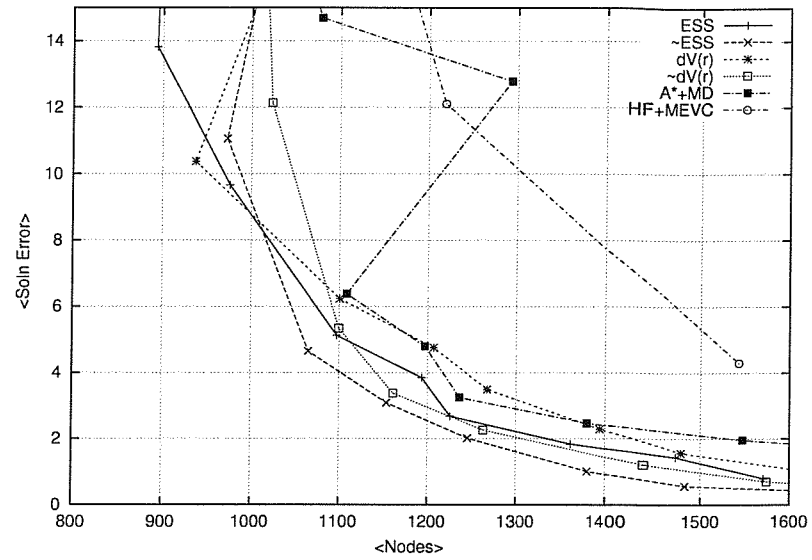
6.3.1 Results

The results of these experiments are presented in Figure 6.6 for the puzzle domains and Figure 6.7 for the grid domains.

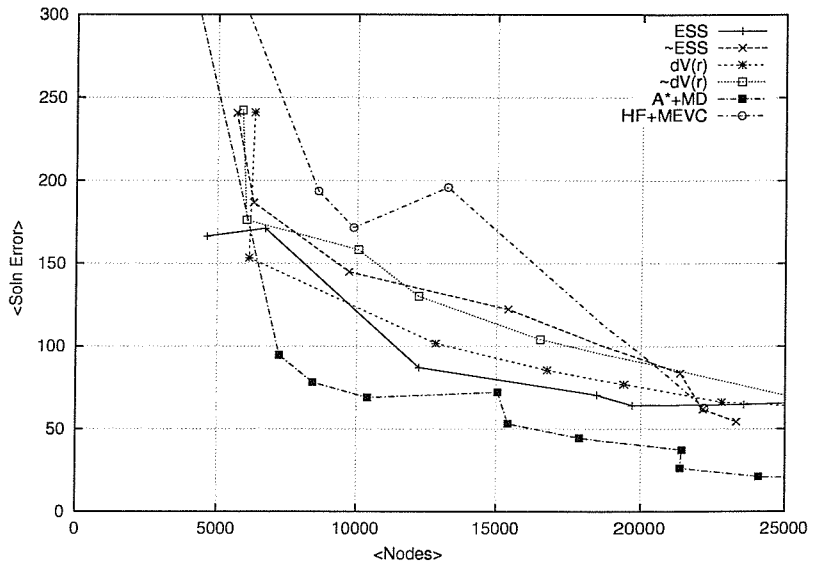
The contrast presented by these results could not be more striking. On the grid problems the E{MRP} based incremental search algorithms provide *at least* a factor of 2 reduction in total node expansions required compared to A*+MD at an equivalent solution quality. This reduction improves to a factor of 10 or more for increasing solution quality and search effort. However, on puzzle domains the advantages are much reduced, with at best a 50% search effort reduction for the 8-Puzzle and, more importantly, at least a factor of 2 *increase* in search effort on the 15-Puzzle. The results also show the generally poor performance of HF+MEVC which performs worse than any other algorithm on the Puzzle domains, and only beats A*+MD for shallow searches on the grid problems.

These results clearly demonstrate the non-linear effect of improving individual decision making on an incremental search agent's overall problem performance. The non-linearity arises because when the agent makes better decisions, its solutions are shorter and thus it has less chance to make a wrong decision. However, if it makes bad decisions, it gets more chance to compound its mistake by making more bad decisions. On the grid problems the results in Figure 5.3 showed that the E{MRP} algorithms had a clear decision making advantage over depth limited A* (as simulated by A*+ANTI-THRASH). Thus using the E{MRP} decision making procedure in an

[3] the additional constant was required to prevent the agent returning to states which were given low updated values because MEVC$(\alpha)/n_\alpha$ stopped them being searched deeply.

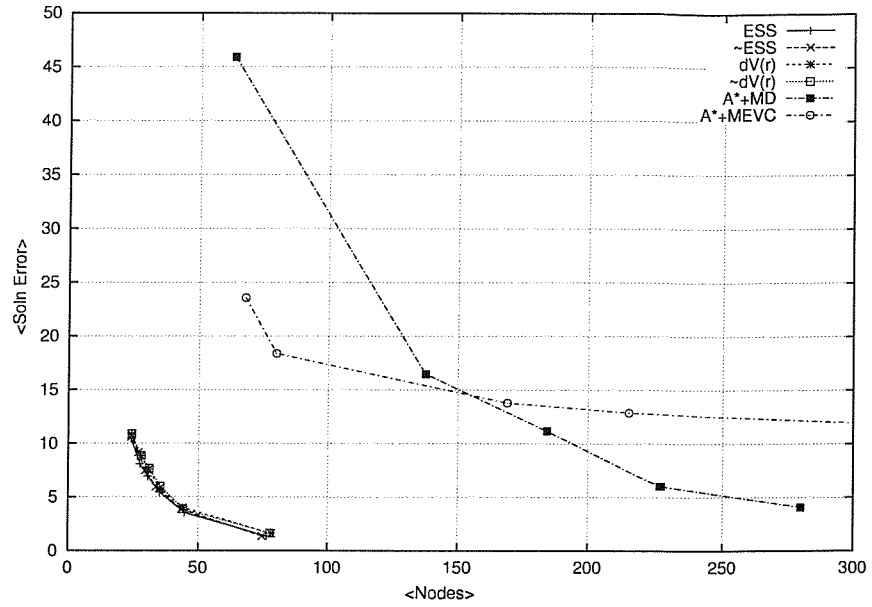


(a)

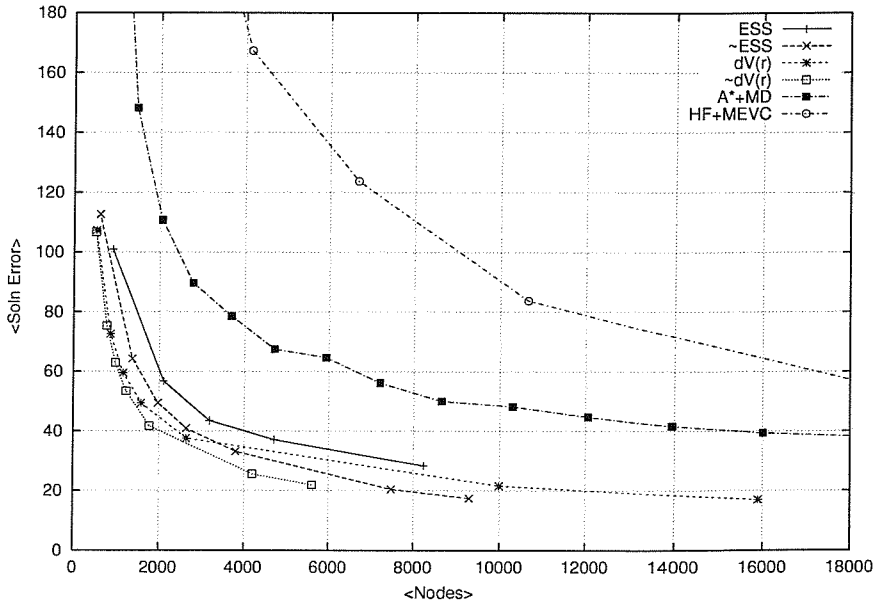


(b)

FIGURE 6.6: These graphs present the performance of the incremental search algorithms developed in this dissertation on (a) the 8-Puzzle and (b) the 15-Puzzle. Each point represents a particular stopping test threshold and gives the average over over 1,000 test problems of the total number of nodes expanded and the deviation from optimality of the solution found by this algorithm at this threshold. If the algorithm did not solve the problem it's deviation is recorded as 1,000. Error bars are not presented to reduce graph clutter.



(a)



(b)

FIGURE 6.7: These graphs present the performance of the incremental search algorithms developed in this dissertation on (a) the 10x10-Grid and (b) the 100x100-Grid. Each point represents a particular stopping test threshold and gives the average over over 1,000 test problems of the total number of nodes expanded and the deviation from optimality of the solution found by this algorithm at this threshold. If the algorithm did not solve the problem it's deviation is recorded as 1,000. Error bars are not presented to reduce graph clutter.

incremental search agent results in a significantly higher performance. However, on the 15-Puzzle Figure 5.2(b) indicated that for more than about 100 states expanded A*+ANTI-THRASH was better. As the average solution length on the 15-puzzle is around 80 this corresponds anywhere to the right of 80,000 total nodes expanded in Figure 6.6(b). Thus, as A*+ANTI-THRASH simulates depth limited A*, the superiority of this algorithm is unsurprising.

The case of the 8-Puzzle is slightly more complex, as the results of Figure 5.2(a) showed that the E{MRP} algorithms had little or no advantage over A*+ANTI-THRASH. Thus, we would expect the algorithms incremental search results to be similar. That the E{MRP} algorithms exhibit a modest advantage in this case can be attributed to their more intelligent $\hat{V}_{c_\infty}/n_{\text{sig}}$ based adaptive stopping rule. By preventing the incremental search agent wasting effort on simple decisions this allows the agent to obtain equivalent solution quality for less effort, or equivalently improved solution quality for the same effort by re-allocating the effort to the harder decisions. However, the advantages provided by intelligent stopping are more modest because the agent is unlikely to encounter many "easy" decisions in solving a problem, and because an effective search control procedure already prevents the agent wasting too much effort by solving the easy problem more rapidly. Indeed, the ability of depth limited A*'s search control to prevent it wasting time on easy problems is one of the reasons why it performs so much better than α-pruned RTA*.

The generally poor performance of HF+MEVC also bears out this hypothesis, as the results of Section 5.4.2 showed that hardest first based search control performed much worse than the E{MRP} based search control procedures. Further, the only point where HF+MEVC shows a significant advantage over A*+MD, i.e. on the 10x10 grid at shallow depths, is also the only point where hardest first showed a significant advantage over A* ANTI-THRASH.

These results demonstrate two main points,

1. The primary factor influencing the performance of an incremental search agent is the efficiency of its decision making, in terms of the number of states which must be expanded to reach a certain decision quality. Increasing the efficiency of an agent's decisions can have a non-linear effect in improving the agent's overall performance.

2. Of secondary importance is effective stopping-control which allows the agent to focus its effort on making the hard decisions, where it is most useful.

This non-linearity is also the cause of some of the fluctuations in the agent's performance profile for low numbers of nodes where it is easily possible for reducing the effort expended on each individual decision to increase the number of decisions to be made and hence increase the total effort expended. This effect is particularly apparent in Figure 5.2 on the 15-Puzzle where for low total node expansions many of the algorithms require more node expansions for reduced decision quality. The remainder of the fluctuations are presumably due to the noise in the sampling process used. In particular the reduction in A*+MD's performance at about 15,000 nodes in Figure 5.2(b) is due to the fact that at these particular depths it is unable to solve one of the problems tested in less than 1,000 steps.

In addition to these main points note that, when the E{MRP} algorithms performed well, the approximate algorithms performed better than the exact ones, with ~ESS performing best on the 8-Puzzle and $\sim d\hat{V}(x_0)$ best on the 100x100 grid. Again this is attributed to the approximate algorithms being less sensitive to the noise in the sampled future value distributions.

Given that A*+MD and MEVC+HF use the same decision procedures and in many situations grow the local search space in the same way, the generally poor performance of HF+MEVC must be attributed to its stopping rule. This indicates that, except for the 10x10 grid at shallow depths, the MEVC$(\alpha)/n_\alpha$ stopping rule is actually *less* effective than simply stopping at a fixed depth bound. The atrocious performance of MEVC+HF on the 100x100 grid makes this point particularly clear. Clearly, there remains a lot of work to be done in identifying an improvement on fixed depth stopping for MINIMIN decision makers.

6.4 Summary

This chapter completes the development of a incremental search agent for on-line decision problems presented in this dissertation by adding stopping and cycle avoidance components to the search control and decision making components developed in the previous two chapters. The stopping component is responsible for determining when it is no-longer worthwhile continuing to work on choosing the current action as the resources could be better spent elsewhere, either in executing the chosen action or working on the next problem. The cycle avoidance component provides the agent with some adaptive memory which allows it to escape from local minima and avoid endlessly repeating the same mistakes.

The stopping problem is essentially a macro-control problem of deciding when the value of spending additional computational resources solving the next problem is worth more than continuing to work on the current problem. As the E{MRP} based decision making systems already provide estimates for the value of information which can be used in a META-GREEDY stopping rule, this is the approach studied in this dissertation. Developing a META-GREEDY stopping rule requires estimates for the marginal value and cost of continuing to work on the current problem. It was found that the best estimates were provided by the non-myopic $\Delta V(i)$ estimate for the value of all further computation rather than the single computation estimates used for search control. Two stopping rules were developed from the $\Delta V(i)$ estimate. The first estimated the cost of the computations required to realise this value by counting the number of states with non-zero individual expansion value. The second used $\Delta V(i)$ to estimate the value of a single computation by computing a moving average of its rate of change. Tests of these stopping rules revealed that, compared to naively stopping after a certain number of nodes have been expanded, they significantly improved decision making performance in all domains. As estimating the rate of change of $\Delta V(i)$ using a moving average introduced a slight lag in the stopping decision, it was decided to use the stopping rule based upon computing the cost of further search in the experiments in the rest of this thesis.

Cycle avoidance is critical to the performance of an incremental search agent, allowing it to avoid past mistakes and guarantee that it will, eventually, solve its problem. Of the many possible cycle avoidance mechanisms this dissertation only considered methods based upon updating a state's future value estimate each time it is visited. It was shown, using a lemma due originally to Korf (1990), that, in an ergodic state space, if such an update makes a state look worse by a finite amount each time the state is visited it is guaranteed to eventually escape from any cycle and reach the goal. Thus there is great flexibility in choosing a cycle avoidance value update rule. For maximum performance the update rule should not make a state so much worse that it stops the agent re-visiting it if it turns out it has made a wrong turn. A number of possible update rules were tested for depth limited A* and E{MRP} based incremental search agents. As a result of these tests it was found that for depth limited A* based agents updating with the estimated value of the path through the best frontier state gave best performance. For the E{MRP} agents updating with the estimated future value *distribution* of the path through the best frontier state worsened by a problem specific constant gives best performance.

The performance of the incremental search agents developed in this chapter was assessed experimentally on the same 4 problem types used throughout this dissertation. The results of these experiments were somewhat mixed. The E{MRP} based incremental search agents were found to give significantly, between 2 and 10 times, better performance than depth limited A* on the grid problems. The E{MRP} based agents also performed better on the 8-puzzle domain, in this case reducing total problem solving effort by up 50%. However, on the 15 puzzle E{MRP} based agents failed miserably, performing at least a factor of 2 *worse* than depth limited A*. Analysis of these results indicated that the primary determiner of incremental search agent performance is the efficiency of the search control and decision making sub-systems. On the grid problems the E{MRP} systems performed significantly better than A* due to their use of calibrated future value estimates. This resulted in significantly better incremental search agent performance. On the 15-puzzle the E{MRP} system performed slightly worse than depth limited A*. The iterative operation of the incremental search agent magnified this performance disadvantage to give the overall performance reduction. The 8-puzzle results demonstrated the advantages of an intelligent stopping rule, where, despite have similar decision making efficiencies, the E{MRP} based agents gave superior performance as they avoided wasting search effort on easy decisions.

The main conclusion to draw from the results of this chapter is that (assuming the agent has effective cycle avoidance routines and does not suffer adverse interactions between its sub-components) the biggest improvements in incremental search agent performance are achieved by improving the efficiency of the agents decision making by effective search control and value inference techniques. Improving how the agent allocates search effort between incremental problems by improving its stopping rule offers smaller but still significant performance improvements. Overall, these results demonstrate that value of computation based meta-control (in the form of search control and stopping routines) *may* provide significant benefits to incremental agent performance.

Chapter 7

Conclusions and further work

7.1 Conclusions

The stated long-term goal of my research is the construction of "autonomous intelligent agents able to solve real-world problems", such as control of autonomous vehicles or chemical plants. Fundamentally, such agents must solve an on-line planning (or optimal control) problem which consists of taking a sequence of observations and deciding *in real-time* upon a sequence of actions to execute which optimise some objective criterion dependent on the sequence of states visited and actions executed. Using dynamic programming arguments it was shown that on-line planning problems can be solved optimally by greedy local hill-climbing w.r.t. the total objective value the agent can get from each state (the state values). Thus, computing state values is of key importance to solving on-line planning problems and was the focus of much of the work in this dissertation. A formal state space model of on-line planning problems was presented in Chapter 2.

The problem we face as agent designers is to design an agent which solves the on-line planning problem as well as possible given the agent's physical and computational limitations. Such an agent design must be *bounded rational* (Russell and Subramaniam 1995; Parkes 1996) in that the agent's decision making procedures must "pay for themselves" by improving action choice enough to offset their computational cost. This notion was formalised in decision theoretic terms in Chapter 2, where it was also shown that proving an agent design is bounded optimal (i.e. the best possible) is generally intractable. Hence, agent designs are usually based on empirical experience. This dissertation contributes to the body of empirical experience by evaluating agent designs based on decision theoretic principles.

Three main types of agent design have been used in the literature to solve the on-line planning problems; 1) the programming design where the designer hand codes the action choices, 2) the planning design where the agent chooses actions based upon some internal world model, and 3) the learning design where the action choice procedure is adaptively modified based upon

the agent's experiences. This dissertation focused specifically on planning (or deliberative) designs. Whilst each design has its individual strengths and weakness, Chapter 2 concluded that planning designs are most useful when "easily grounded world models with sufficient predictive power are available *and* locally observable environmental queues alone are insufficient to simply identify actions which provide the required performance level".

In real-world open environments domain uncertainty and real-time issues prevent a deliberative agent from generating a complete solution before acting. Instead the agent must interleave decision making and execution to maximise real-time performance. In Chapter 3 two designs for deliberative AI components suitable for on-line contexts were presented; bounded deliberation, where the action choice procedure has a guaranteed maximum resource usage, and dynamic deliberation, where a meta-controller can adaptively trade-off the action choice procedure's resource usage against its decision quality. Dynamic deliberation is potentially more efficient than bounded deliberation as it allows the agent to exploit any variance in time pressure and problem difficulty to make best use of the resources available. However, as demonstrated in this work, designing a meta-controller to realise this benefit is far from trivial.

In the dynamic deliberative agent designs studied in this dissertation the basic thinking/execution interleaving is provided by incremental search, where decisions are made incrementally based on a partial search through the local space of possible solutions. Incremental search was chosen because it provides a relatively simple mechanism to trade-off decision quality against computational effort by varying the amount of search performed per decision. An effective incremental search based agent design must address four main sub-problems;

1. **Decision Making** – how does the agent make "good" action selections based upon the often incomplete (and and therefore uncertain) information it has available? These decisions must take account of the complexity induced uncertainty caused by the agent's computational limitations.

2. **Search Control** – given the agent can only perform a limited amount of search to gather information before a decision must be made, how does it order its local search space expansions so it makes the "best" decisions possible?

3. **Stopping** – given that further computation has both variable costs due to delaying decision making and benefits due to improvements in the information action choice is based upon, how does the agent decide when it is time to stop thinking and commit to action?

4. **Cycle Avoidance** – given the limited *local* information the agent has available to make each decision, how does it modify its operation to avoid repeating past mistakes and hence ensure progress towards a solution?

The search control and stopping problems are meta-control problems concerned with effective management of the agent's computational resources. Decomposing the agent design problem in this way helps by allowing each of the sub-problems to be considered independently (whilst,

of course, acknowledging that their interactions may cause us problems later). This was the approach taken in this dissertation where the agent design is developed and analysed incrementally by adding additional abilities to previously developed designs.

Clearly, the four sub-problems are highly inter-dependent requiring that the agent be carefully designed so the sub-problems solutions do not interact adversely. Indeed, one of the main results of this dissertation is to show that improving one component in isolation can actually result in reduced overall agent performance. For example, Section 4.9.4.2 showed that improving search control by using truncated A* search instead of fixed depth search on grid problems reduces agent performance. This dissertation investigated the use of decision theoretic techniques to cope with the uncertainty in these problems caused by the agent's computational limitations.

Chapter 4 examined the decision making component of a dynamic deliberative agent. Only explicitly rational decision making systems, which make decisions by greedily maximising explicitly computed local state future value estimates, were studied because they simplify meta-control integration. Storing the true future value for every state is intractable so the local values required must be estimated on-line from the information contained in the local search space constructed by the incremental search. To improve the quality of the value estimates specially chosen local feature detectors, such as heuristic estimates or lower bounds, were used to label the local search space with additional useful information.

The information contained in the labelled local search space can be used to compute local state value estimates by either using prior information to define a way of inferring value estimates, or using machine learning techniques to learn a value estimation function, or using a hybrid technique which combines the inference and learning techniques. The MINIMIN estimate, which is based on the invariants implied by the Bellman backup equations, was presented as an inference technique. This is the estimation technique underlying A* and Korf's RTA*. The gold standard estimate, GSE, which learns a function mapping directly from labelled local search spaces to local value estimates, was presented as an optimal learnt estimate. Unfortunately, the GSE requires exponential memory making it impractical for general use. Hansson and Mayer's BPS algorithm was presented as an alternative learnt estimation function. This uses additional assumptions to offset GSE's memory requirements. The E{MRP} estimate was developed as a hybrid technique based upon probabilistic inference. This uses strong assumptions about frontier states future value independence to define a probabilistic equivalent of the Bellman backup equation. The E{MRP} estimates, by treating the agent's current frontier value estimates as uncertain, allow the agent to take some account of any additional information it may obtain in the immediate future. This is a hybrid technique because the likely effects of the additional information are learnt. The ~E{MRP} was developed as a computationally efficient approximation of the E{MRP} based upon approximating probability distributions with parameterised Gaussian distributions. This approximation is one of the key contributions of this dissertation.

The five possible value estimate methods were tested experimentally in Section 4.9 on 4 problem types, the 8-puzzle, 15-puzzle, 10x10 grid and 100x100 grid. These experiments demonstrated that the labelled local search space contains much information which can be exploited by sophisticated value estimate systems to improve decision making performance well over that of MINIMIN. The BPS, E{MRP} and ~E{MRP} algorithms were all reasonably effective at exploiting this information. However, the significantly better performance of GSE demonstrates there is much potential for further improvement. Performing the same tests using A* based search control demonstrated the importance of coordinating the search control and decision making components. Poor coordination results in thrashing where the agent performs additional search which has no beneficial (and may even have a detrimental) effect on value estimation, wasting resources and reducing overall performance.

The search-control component of an incremental search agent is examined in Chapter 5. In Section 3.3.1 it was shown that the meta-control problem could be modelled as a Markov decision problem (MDP) which, like the original planning problem, can be solved using greedy local hill-climbing w.r.t. the computation's expected future value. Unfortunately, as the meta-control MDP is actually more complex than the original on-line planning problem, computing the true value of computation is intractable. The value estimates computed by the decision making system provide useful information for estimating the true value of information. Thus, it was decided to focus on explicitly meta-rational search control systems, which choose the local search space expansions to execute by greedily maximising explicitly computed approximations to the marginal value of computation.

The key step in developing a tractable marginal value of computation approximations is identifying a propagation function which shows how a particular computation outcome affects the agent's decision making. By identifying the propagation functions for the MINIMIN and E{MRP} decision making systems, 3 types of value of computation approximation can be developed. The myopic expected value of computation (MEVC) approximation estimates the value of computation by assuming the decision will be made immediately after the next computation so all other frontier values are fixed at their current values. The root state sensitivity ($d\hat{V}(x_0)$) approximation estimates the value of computation in terms of the expected absolute change in the estimated value of the root state. The expected step size (ESS) approximation estimates the value of computation in terms of the expected absolute change in the marginal value of all further computations. The efficient computation of the $d\hat{V}(x_0)$ and ESS approximations was made possible for E{MRP} decision makers using a new technique for the efficient computation of a state's updated expected value after the computation, $\langle \hat{V}'(x) | \, \mathrm{p}'_c = \delta_y \rangle$. It was also shown how this derivation could be readily extended to work with continuous distributions, and hence for ~E{MRP} decision makers.

The performance of the search control systems was evaluated in Section 5.4 compared against A*. These experiments showed that value of computation based search control could be effective in improving the quality of decision making by avoiding the thrashing problems encountered earlier. These benefits are particularly apparent for low quality feature detectors. However, the

superior performance of ANTI-THRASH decision making on the 15-puzzle demonstrates there is still some way to go in developing search control systems. As expected the short-sighted nature of the MEVC approximation was found to give significantly poorer performance than the non-myopic $d\hat{V}(x_0)$ and ESS approximations.

The stopping and cycle avoidance components of an incremental search agent were examined in Chapter 6. The stopping problem is also a meta-control problem which can be solved by greedily maximising an estimate of the marginal value of further computation. The estimates for the value of a single computation developed for search control were shown to be poor predictors of the true value of continuing to search, so two new value of computation estimates were developed. For the MINIMIN decision makers this was computed by making the MEVC estimate less short-sighted by considering sets of computations. For the E{MRP} decision makers the value estimates computed as part of the decision making process can be used to directly estimate the marginal value of all further computation. Given these estimates the stopping rule is simply to stop when the estimated marginal value of computation exceeds its estimated cost. Tests of these stopping rules revealed that, compared to naively stopping after a certain number of nodes have been expanded, they significantly improved decision making performance in all domains, though again the MEVC estimator generally performed worse than the E{MRP} one.

Cycle avoidance is critical to the performance of an incremental search agent, allowing it to avoid past mistakes and guarantee that it will, eventually, solve its problem. Of the many possible cycle avoidance mechanisms this dissertation only considered methods based upon updating a state's future value estimate each time it is visited. It was shown, using a lemma due originally to Korf (1990), that in an ergodic state space if the value update makes each state look worse by a finite amount each time it is visited then a greedy agent is guaranteed to eventually escape from any cycle and reach the goal. Thus there is great flexibility in choosing a cycle avoidance value update rule. For maximum performance the update rule should not make a state so much worse that it stops the agent re-visiting it if it has made a wrong turn. A number of possible update rules were tested for depth limited A* based incremental search agents. Surprisingly these results showed that, so long as the convergence requirements were met, agent performance was relatively insensitive to the update rule used, particularly for deeper searches.

The performance of complete incremental search agents, consisting of decision making, search control, stopping and cycle avoidance was evaluated in Section 6.3. The results of these experiments were somewhat mixed, with the E{MRP} based incremental search agents performing significantly better on the grid problems, and slightly better on the 8-puzzle. On the 15-puzzle however E{MRP} performed much worse than depth limited A*. Analysis of these results indicated that (assuming no adverse interactions) the primary determiner of incremental search agent performance is the efficiency of the search control and decision making sub-systems, with the iterative operation of the incremental search agent magnifying slight efficiency improvements into significant performance advantages. The stopping rule was found to have

a less dramatic but still significant effect on agent performance by reducing the search effort wasted on easy decisions.

The main conclusions to draw from this dissertation can be summarised in four main points,

1. Effectively managing the interaction between the decision making, search control and stopping components of an incremental search agent is critical to its performance. Poor interactions can lead to thrashing and significantly reduced performance. Well managed interactions, such as the combination of MINIMIN decision making with A* search control and fixed depth stopping in A*+MD, can compensate for the poor performance of one component (MINIMIN decision making in A*+MD's case) to give high overall performance.

2. The most significant improvements to an incremental search agent's performance (assuming no adverse interactions) can be obtained by improving the way in which it exploits the information already contained in the local search space to improve the efficiency of its decision making. Probabilistic inference techniques, such as E{MRP} and BPS, are promising ways of doing this.

3. The efficiency of the agent's decision making can also be improved by using sophisticated search control procedures to ensure the local search space only contains the information most useful for decision making. However, in order to avoid thrashing problems the search control and decision making systems must be carefully coordinated. Value of computation based greedy search control techniques, such as $d\hat{V}(x_0)$ and ESS, are promising ways of doing this.

4. An incremental search agent's performance can also be improved using sophisticated stopping control to manage how it allocates effort to individual sub-problems. However, effective search control limits the benefits available in this way so it gives the smallest improvements over-all. Again, to avoid thrashing problems stopping control must be carefully coordinated with decision making. Value of computation based greedy search control techniques are promising ways of doing this.

7.2 Further work

There are considerable possible avenues for future work suggested by the results contained in this dissertation.

Once it is realised that value estimation can be cast as a probabilistic inference problem there is considerable scope to develop alternative value estimation techniques. In particular, it would be interesting to develop approximate E{MRP} estimation algorithms using alternative parameterised distributions. Mixtures of Gaussians and truncated exponential distributions appear to be particularly appropriate to this task. Also, the success of parameterised probabilistic

propagation in reducing the computational cost of E{MRP} estimation indicates a similar approach may be useful for reducing the computational cost of the GSE and BPS estimators.

One significant potential advantage of probabilistic inference for value estimation which has not been tested is its ability to exploit the information contained in multiple feature detectors (heuristics) to improve decision making performance. Thus, an alternative to developing a single high quality heuristic is to use multiple lower quality heuristics. Similar techniques utilising many of low quality feature detectors (such as mixture of experts, bagging, and boosting) have recently been used with some success in machine learning contexts and it would be interesting to see if this success could be replicated in on-line planning contexts. The multiple heuristic estimates produced by pattern databases represent one obvious way of producing the required multiple feature detectors.

Alternatively, a significant problem with the search control and decision making techniques developed in this dissertation is that they require the entire local search space be stored in memory to operate. For severely resource limited agents or very difficult decision problems this is clearly infeasible. Hence developing a low-memory version of these procedures would be advantageous. Using an adaptive iterative deepening technique such as IDA*_CR (Sarkar, Chakrabarti, Ghose, and de Sarkar 1991) appears to be one possible way of doing this. However, one significant problem with this approach is efficiently re-generating an interior nodes expansion value to allow sub-tree pruning in the depth first search.

Another problem with the incremental search agents developed in this dissertation is that they forget the local search space between problems. This is very inefficient, and modifying the algorithms to only forget the information no-longer valid in the new problem is one way of improving the agent's performance. This would also have the side-benefit of improving the agent's cycle avoidance behaviour.

The generally high performance of ANTI-THRASH and fixed depth A* search (when used with reasonably accurate heuristics) indicates that to get best performance A* searches should only be stopped when the depth increases. However, using a fixed depth for an entire task seems wasteful. Thus, it would be interesting to investigate whether using meta-control techniques to control the depth of the A* search is beneficial. Note, this differs from the HF+MEVC technique tested in Section 6.3 in that a stopping decision could only be made when the depth increases. This could be done using the value of computation estimation techniques developed in this dissertation. Alternatively, as the number of decisions is quite small the meta-controller could be learnt directly. Indeed, given the poor quality and significant bias of the future value of computation estimates used for stopping in this dissertation, a learnt stopping rule may be beneficial even for E{MRP} based decision makers.

Bibliography

Agre, P. and D. Chapman (1990). What are plans for? In P. Maes (Ed.), *New Architectures for Autonomous Agents: Task-level Decomposition and Emergent Functionality*. Cambridge, Mass: MIT Press.

Allen, J. F., J. Hendler, and A. Tate (1990). *Readings in Planning*. Morgan Kaufmann.

Arkin, R. C. (1998). *Behaviour-Based Robotics*. Cambridge, MA: MIT Press.

Astrom, K. and B. Wittenmark (1994, December). *Adaptive Control* (2nd ed.). Addison-Wesley.

Baird III, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pp. 30–37.

Barnett, S. and R. Cameron (1985). *Introduction to Mathematical Control Theory* (2nd ed.). New York, NY: Clarendon Press.

Barto, A. G., S. J. Baradtke, and S. P. Singh (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence 72*(1-2), 81–138. Also published as UMAss Amherst Technical Report 91-57 in 1991.

Battiti, R. (1996). Reactive search: Toward self–tuning heuristics. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith (Eds.), *Modern Heuristic Search Methods*, pp. 61–83. Chichester: John Wiley & Sons Ltd.

Baum, E. B. (1993). How a Bayesian approaches games like chess. In *Games: Planning and Learning, Papers from the 1993 Fall Symposium*, Menlo Park CA, pp. 48–50. AAAI Press. Technical Report FS-93-02.

Baum, E. B. and W. D. Smith (1995, September). Best play for imperfect players and game tree search; Part I - Theory. Technical report, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540.

Baum, E. B. and W. D. Smith (1997). A Bayesian approach to relevance in game playing. *Artificial Intelligence 97*, 195–242.

Bellman, R. and S. E. Dreyfus (1962). *Applied Dynamic Programming*. Princeton, N.J.: Princeton Univ. Press.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P. and S. E. Shreve (1978). *Stochastic Optimal Control: The discrete time case*, Volume 139 of *Mathematics in Science and Engineering*. Academic Press.

Binmore, K. (1992). *Fun and Games: A text on Game Theory*. D.C. Heath and Company, Massachusetts.

Boddy, M. and T. L. Dean (1994). Deliberation scheduling for problem solving in time-constrained enviroments. *Artificial Intelligence 67*(2), 245–285.

Bonet, B. and H. Geffner (1999, September). Planning as heuristic search: New Results. In *Proceedings of the Fifth European Conference on Planning (ECP-99)*, Durham, United Kingdom. Springer.

Bonet, B., G. Loerincs, and H. Geffner (1997). A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, Providence, Rhode Island, pp. 714–719. AAAI Press / MIT Press.

Boutilier, C., T. Dean, and S. Hanks (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence 11*, 1–94.

Brooks, R. (1982). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation RA-2(1)*, 14–23.

Brooks, R. A. (1991). Intelligence without reason. In J. Myopoulos and R. Reiter (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, pp. 569–595. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Bulitko, V., I. Levner, and R. Greiner (2002, July). Real-time lookahead control policies. In *Proceedings of the AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems.*, Edmonton, Alberta, Canada.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence 69*, 161–204.

Camacho, E. F., C. Bordons, and M. Johnson (1999). *Model Predictive Control*. Springer Verlag.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence 32*, 333–379.

Chen, J. and A. M. K. Cheng (1994). A fast, partially parallelizalbe algorithm for predicting execution time of EQL rule-based programs. In *International Conference on Parallel Processing*.

Chung, J. Y., W. S. Liu, and K. J. Lin (1990). Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers 39*, 1156–1173.

Cox, R. (1946). Probability, frequency, and reasonable expectation. *Americian Journal of Physics 14*(1), 1–13.

Dean, T. and M. Boddy (1988). An analysis of time-dependent planning. In *Proc. Seventh National Conference on Artificial Intelligence*, pp. 49–54.

Dean, T. and S.-H. Lin (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*.

Dean, T. L. and M. P. Wellman (1991). *Planning and Control*. San Mateo: Morgan Kaufmann.

Doran, J. and D. Michie (1966). Experiments with the graph traverser program. In *Proc. Royal Society of Lundon*, Volume 294 of *A*, pp. 235–259.

Doyle, J. (1992). Rationality and its roles in reasoning. *Computational Intelligence 8*(2), 376–409.

Drummond, M. and A. Tate (1989). AI planning: A tutorial and review. Technical report, Artificial Intelligence Applications Institute, Division of Informatics, The Unversity of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN.

Fikes, R. E. and N. J. Nilsson (1993). STRIPS: a retrospective. *Artificial Intelligence 59*(1-2), 227–232.

Firby, R. J. (1989, January). *Adaptive Execution in Complex Dynamic Domains*. PhD. thesis, Yale University.

Garcia, C., D. Prett, and M. Morari (1989). Model predictive control: Theory and practice - A survey. *Automatica 25*(3), 335–348.

Garvey, A., M. Humphrey, and V. Lesser (1994). Task interdependencies in design-to-time real-time scheduling. Technical report, University of Massachusetts.

Garvey, A. and V. Lesser (1993). Design-to-time Real-Time Scheduling. *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Planning, Scheduling and Control 23*(6), 1491–1502.

Garvey, A. and V. Lesser (1994, May). A survey of Research in Deliberative Real-Time Artificial Intelligence. *Journal of Real-Time Systems 6*(3), 317–347.

Gasching, J. C. (1977). Exactly how good are heursitics: Toward a realistic predictive theory of best-first search. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Boston, MA, pp. 434–441.

Georgeff, M. P. and F. F. Ingrand (1989, August). Decision-making in an embedded reasoning system. In *Proc. IEEE International Conference on Robotics and Automation*, pp. 972–978.

Gillespie, D. T. (1992). *Markov processes : an introduction for physical scientists*. Boston: Academic Press.

Glover, F. and M. Laguna (1997). *Tabu Search*. Kluwer Academic Publishers.

Good, I. J. (1968). A five year plan for automatic chess. *Machine Intelligence 2*, 89–118.

Good, I. J. (1971). Twenty-seven principles of rationality. In V. P. Godambe and D. A. Sprott (Eds.), *Foundations of Statistical Inference*, pp. 108–141. Toronto: Hold, Rinehart, Winston.

Good, I. J. (1983). *Good thinking: The foundations of probability and its applications.* Mineapolis: University of Minnesota Press.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In A. Prieditis and S. Russell (Eds.), *Proceedings of the Twelfth International Conference on Machine Learning*, San Francisco, CA, pp. 261–268. Morgan Kaufmann.

Hacking, I. (1967, December). Slightly more realistic personal probability. *Philosophy of Science 34*, 311–325.

Hamilton, S. and L. Garber (1997). Deep Blue's hardware-software synergy. *Computer 30*(10), 29–35.

Hansen, E. and S. Zilberstein (2001a). Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence 126*, 139–157.

Hansen, E. A. and S. Zilberstein (2001b). LAO* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence 129*(1-2), 35–62.

Hansen, E. A., S. Zilberstein, and V. A. Danilchenko (1997). Anytime heuristic search: First results. CMPSCI Technical Reports 97-50, University of Massachusetts Amherst.

Hansson, O. (1998). *Bayesian Problem Solving Applied to Scheduling*. PhD. thesis, University of California, Berkeley.

Hansson, O. and A. Mayer (1989). Heuristic search as evidential reasoning. In *Proceedings of the fifth workshop on uncertainty in Artificial Intelligence*, Windsor, Ontario, pp. 152–161.

Hansson, O. and A. Mayer (1994). DTS: A decision-theoretic scheduler for space telescope applications. In M. Zweben and M. S. Fox (Eds.), *Intelligent Schedulling*, Chapter 13, pp. 371–388. Morgan Kauffman.

Harada, D. and S. Russell (1999). Extended abstract: Learning search strategies. In *Proc. AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, Stanford, CA. AAAI.

Hauskrecht, M., N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Uncertainty in Artificial Intelligence*, pp. 220–229.

Holete, R., M. Preez, R. M. Zimmer, and A. J. MacDonald (1996). Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proceedings of the Thirteenth National Conference on Artificiail Intelligence (AAAI-96)*, pp. 530–535.

Horvitz, E. (1990). *Computation and action under bounded resources*. PhD. thesis, Stanford University.

Horvitz, E. (2001). Principles and application of continual computation. *Artificial Intelligence 126*(1-2), 159–196.

Horvitz, E., H. Suermondt, and G. Cooper (1989). Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proc. of the Fifth Workshop on Uncertainty in Artificial Intelligence*, North-Holland, pp. 182–189.

Horvitz, E. and S. Zilbertstein (Eds.) (1996). *Fall Symposium on Flexible computation in Intellligent Systems*, Menlo Park, CA.

Horvitz, E. J. (1988, August). Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN, pp. 111–116. Morgan Kaufmann, San Mateo.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. New York: Technology Press and John Wiley and Sons.

Howard, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics SSC-2*(1), 22–26.

Hyvärinen, A., J. Karhunen, and E. Oja (2001). *Independent Component Analysis*. John Wiley and Sons.

Ingrand, F. F. and M. P. Georgeff (1990, November). Managing deliberation and reasoning in real-time ai systems. In *Proc. Workshop in Innovative Approaches to Planning , Scheduling, and Control*, pp. 284–291.

Ingrand, F. F., M. P. Georgeff, and A. S. Rao (1992, December). An architecture of real-time reasoning and system control. *IEEE Expert 7*(6), 34–44.

Jacobs, O. (1974). *Introduction to Control Theory*. New York, NY: Oxford University Press.

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1997). Planning and acting in partially observable stochastic domains. Unpublished Technical Report.

Kaelbling, L. P. and S. J. Rosenschein (1990). Action and planning in embedded agents. In P. Maes (Ed.), *New Architectures for Autonomous Agents: Task-level Decomposition and Emergent Functionality*, pp. 35–48. Cambridge, Mass: MIT Press.

Kim, K.-E. and T. Dean (2003). Solving factored MDPs using non-homogeneous partitions. *Artificial Intelligence 147*(1-2), 225–251.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *Science 220*, 671–680.

Kirsh, D. (1991). Today the earwig, tommorow man. *Artificial Intelligence 47*(3), 161–184.

Koenig, S. (2001). Agent-centered search. *Artificial Intelligence Magazine 22*(2), 109–131.

Koenig, S., A. Blum, T. Ishada, and R. Korf (Eds.) (1997, July). *Proceedings of the AAAI-97 Workshop on On-Line Search*, Providence, Rhode Island. AAAI Press. Available as AAAI Technical Report WS-97-10.

Koenig, S. and R. G. Simmons (1995). Real-Time Search in Non-Deterministic Domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1660–1669.

Korf, R. (1985a). Depth-first iterative-deepening: An optimal admissible tree search. *Artificiall Intelligence 27*, 97–109.

Korf, R. E. (1985b). Macro operators: A weak method for learning. *Artificial Intelligence 26*, 35–77.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence 42*, 1189–211.

Littman, M. L., A. R. Cassandra, and L. P. Kaebling (1995). Efficient dynamic programming updates in POMDPs. Technical Report CS-95-19, Brown University.

Liu, J., K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao (1991). Algorithms for scheduling imprecise computations. *IEEE Transactions on Computers 24*(5), 58–68.

Markovitch, S. and Y. Sella (1996). Learning of resource allocation strategies for game playing. *Computational Intelligence 12*(1), 88–105.

Mayer, A. E. (1994). *Rational Search*. PhD. thesis, University of California at Berkeley.

Mayne, D. Q., J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert (2000, June). Constrained model predictive control: Stability and optimality. *Automatica 36*(6), 789–814.

Mok, A. (1990). Formal analysis of real-time equational rule-based systems. In *Proc. of the Real-Time Systems Symposium*, pp. 308–318. IEEE.

Mouaddib, A.-I. and S. Zilberstein (1997). Handling duration uncertainty in meta-level control of progressive processing. In M. Pollack (Ed.), *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1201–1206. Morgan Kaufmann.

Musliner, D. (1993). *CIRCA: The cooperative intelligent real-time control architecture*. PhD. thesis, Computer Science Deptarment, University of Michigan.

Musliner, D., E. Durfee, and K. Shin (1993). CIRCA: A cooperative intelligent real-time control architecture. *IEEE Trans. Sys. Man, and Cybernetics 23*(6), 1561–1574.

Myers, K. L. (1996). A procedural knowledge approach to task-level control. In *Proceedings of the Third International Conference on AI Planning Systems*.

Neal, M. (1995). *Bayesian Learning for Neural Networks*. Springer-Verlag.

Neumann, J. V. and O. Morgenstern (1944). *The Theory of Games and Economic Behaviour*. Pricenton University Press.

Parkes, D. and L. G. Greenwald (2001). Approximate and compensate: A method for risk-sensitive meta-deliberation and continual computation. In *AAAI Fall Symposium on Using Uncertainty within Computation*. AAAI.

Parkes, D. C. (1996). Bounded rationality. Technical report, Computer and Information Science Department, University of Pennsylvania.

Parr, R. E. (1998). *Hierarchical Control and Learning for Markov Decision Processes*. PhD. thesis, Princeton University, University of California at Berkeley.

Pearl, J. (1984). *Heuristics*. Addison-Wesley.

Pearl, J. (1988). *Probablistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Pemberton, J. C. (1995). *Incremental Search Methods for Real-Time Decision Making*. PhD. thesis, University of California, Los Angeles, University of California, Los Angeles.

Peng, J. and R. J. Williams (1995). Incremental multi-step Q-learning. *Machine Learning 22*, 283–290.

Poupart, P., C. Boutilier, R. Patrascu, and D. Schuurmans (2002, July 28–August 1). Piecewise linear value function approximation for factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02)*, Menlo Parc, CA, USA, pp. 292–299. AAAI Press.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley.

Raiffa, H. and R. Schlaifer (1961). *Applied Statistical Decision Theory*. Harvard University Press.

Ramsey, F. P. (1931). *Foundations of Mathematics and Other Logical Essays*. London: Routledge and Kegan.

Rosenblatt, J. (2000). Maximizing expected utility for optimal action selection under uncertainty. *Autonomous Robots 9*(1), 17–25.

Russell, S. and P. Norvig (1995). *Artificial Intelligence a Modern Approach*. Prentice Hall International Inc.

Russell, S. and D. Subramaniam (1995). Provably Bounded-Optimal Agents. *Journal of Artificial Intelligence 2*, 575–609.

Russell, S. and E. Wefald (1989). *Do the right thing: Studies in Limited Rationality*. The MIT Press.

Russell, S. J. and W. Wefald (1991). Principles of metareasoning. *Artificial Intelligence 49*, 362–395.

Russell, S. J. and S. Zilberstein (1991). Composing real-time systems. In *Twelfth International Joint Conference on Artificial Intelligence*, pp. 212–217.

Sarkar, U., P. Chakrabarti, S. Ghose, and S. de Sarkar (1991). Reducing re-expansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence 50*, 207–221.

Savage, L. J. (1954). *The Foundations of Statistics*. New York: Wiley.

Schoppers (1987). Universal plans for reactive robots in unpredicatable environments. In *Proceedings Internations Joing conference on Artificial Intelligence*, pp. 1039–1046.

Shafer, G. (1976). *A mathematical theory of evidence*. Princeton University Press.

Simon, H. A. (1982). *Models of Bounded Rationality*. M. I. T. Press.

Skyrms, B. (1990). *The dynamics of rational deliberation*. Cambridge, Massachusetts: Harvard University Press.

Sloman, A. (1999). What sort of architecture is required for a human-like agent? In M. Wooldridge and A. Rao (Eds.), *Foundations of Rational Agency*. Kluwer Academic Publishers.

Smith, W. D. (1992, December). Approximation of staircases by staircases. Technical Report 92-109-3-0058-8, NEC Research Institute Inc., 4 Independence Way, Princeton, New Jersey.

Sondik, E. J. (1978). The optimal control of partially observable markov processes over the infinite horizion:discounted costs. *Operations Research 26*(2), 282–304.

Stentz, A. and M. Hebert (1995). A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots 2*(2), 127–145.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning 3*, 9–44.

Sutton, R. S. (1990). Integerated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh Int. Conf. on Machine Learning*, pp. 216–224. Morgan Kaufmann.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tash, J. K. (1996). *Decision Theory Made Tractable: The value of Deliberation with Application to Markov Decision Process Planning*. PhD. thesis, University of California, Berkeley.

Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 888–893.

Thrun, S., A. Buecken, W. Burgard, D. Fox, T. Froehlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt (1998). Map Learning and High-speed Navigation in RHINO. In D. Kortenkamp, R. Bonassso, and R. Murphy (Eds.), *Artificial Intelligence-Based Mobile Robotics: Case Studies of Successful Robot Systems*, pp. 21–52. MIT Press, Cambridge Mass.

Tsai, H. and A. M. K. Cheng (1994). Termination analysis of OPS5 expert systems. In *National Conference on Artificial Intelligence*, pp. 193–198.

von Winterfeldt, D. and W. Edwards (1986). *Decision Analysis and Behavioral Research*. Cambridge University Press.

Vrbsky, S. and J. Liu (1995). *Producing monotonically improving approximate answers to database queries*, pp. 117–133. Kluwer Academic Publishers.

Watkins, C. J. (1989). *Models of Delayed Reinforcement Learning*. PhD. thesis, Cambridge University, Psychology Department, Cambridge, United Kingdom.

Watson, S. R. and D. M. Buede (1987). *Decision Synthesis: The Principles and Practice of Decision Analysis*. Cambridge Univesity Press.

Wellman, M. and C. Liu (1994). State-space abstraction for anytime evaluation of probabilistic networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, Seattle, WA.

Whittle, P. (1983). *Optimization over time*, Volume 2 of *Wiley Seris in probability and methematical statistics*. Wiley.

Williams, R. and L. Baird (1993, November). Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University.

Zadeh, I. A. (1965). Fuzzy sets. *Information and Control 8*, 338–353.

Zhang, N. and W. Zhang (2001). Speeding up the convergence of value iteration in partially observable markov decision processes. *JAIR 14*, 29–51.

Zilberstein, S. (1993). *Operational rationality thorugh compilation of anytime algorithms*. PhD. thesis, Computer Science Division, University of California.

Zilberstein, S., F. Charpillet, and P. Chassaing (2003). Optimal sequencing of contract algorithms. *Annals of Mathematics and Artificial Intelligence 39*(1-2), 1–18.

Zilberstein, S. and S. Russell (1996). Optimal composition of real-time systems. *Artificial Intelligence 82*(1-2), 181–213.