The Control of Reasoning in Resource-Bounded Agents

Schut, M.C.; Wooldridge, M.

published in
Knowledge Engineering Review
2002

DOI (link to publisher)
10.1017/S0269888901000157

document version
Publisher's PDF, also known as Version of record

citation for published version (APA)
Schut, M. C., & Wooldridge, M. (2002). The Control of Reasoning in Resource-Bounded Agents. *Knowledge Engineering Review*, *16*(3), 215-240. https://doi.org/10.1017/S0269888901000157

The Knowledge Engineering Review, Vol. 16:3, 215–240. © 2001, Cambridge University Press
DOI: 10.1017/S0269888901000157 Printed in the United Kingdom

The control of reasoning in resource-bounded agents

MARTIJN SCHUT and MICHAEL WOOLDRIDGE

Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK.
E-mail: m.c.schut@csc.liv.ac.uk; m.j.wooldridge@csc.liv.ac.uk

Abstract

Autonomous agents are systems capable of autonomous decision-making in real-time environments. Computation is a valuable resource for such decision-making, and yet the amount of computation that an autonomous agent may carry out will be limited. It follows that an agent must be equipped with a mechanism that enables it to make the best possible use of the computational resources at its disposal. In this paper we review three approaches to the control of computation in resource-bounded agents. In addition to a detailed description of each framework, this paper compares and contrasts the approaches, and lists the advantages and disadvantages of each.

1 Introduction

Until about the mid-1980s, research in the design of intelligent agents was dominated by STRIPS-style classical planning approaches (Allen *et al*., 1990). These approaches focussed on algorithms for automatic plan generation, where an agent generates a plan (cf. program) to achieve some state of affairs more or less from first principles. This style of planning, it was believed, was a central component in rational action. By the mid-1980s, however, a number of researchers, of whom Rodney Brooks is probably the best known (Brooks, 1999), began to claim that planning approaches, which directly depend upon explicit symbolic reasoning, were fundamentally flawed. First, it was argued, symbolic reasoning systems tend to be computationally intractable – rendering them of limited value to agents that must operate in anything like real-time environments (Bylander, 1994; Chapman, 1987). Second, much everyday behaviour does not appear to involve abstract deliberation, but seems to arise from the interaction between comparatively simple agent behaviours and the agent's environment.

The challenge posed by the behaviour-based artificial intelligence (AI) research of Brooks and colleagues has arguably led to some fundamental changes in the agenda of the AI community in general. First, it has become widely accepted that intelligent behaviour in an agent is more closely coupled to the environment occupied by the agent than was perhaps hitherto acknowledged. As a consequence, there has been renewed interest in the use of more realistic environmental settings for the evaluation of agent control architectures. Second, it has become accepted that while reasoning is an important resource for intelligent decision-making, it is not the only such resource. As a consequence, there has been much interest in *hybrid* approaches to agent design, which attempt to combine reasoning and behavioural decision-making (Mueller, 1997; Wooldridge & Jennings, 1995).

The basic problem faced by an autonomous agent is that of *decision-making* and, in particular, deciding what action to perform. But a normative notion of decision-making, (such as decision theory (Von Neumann & Morgenstern, 1944)), is not suitable for implementation in intelligent agents: decision theory has been highly successful as a tool with which to analyse formal mathematical models of decision situations, but it was never intended for *implementation*. In fact, classical decision theory seems to imply that an agent must have both unlimited time and computational resources in order to reach a decision. In addition, it assumes that an agent always has complete access to its environment

to gather all relevant information necessary to making a decision. For a resource-bounded agent situated in a real-time environment, these assumptions are not valid.

Reasoning is thus a valuable resource for decision-making in intelligent agents. It follows that the ability to effectively *control* reasoning – to apply it to best effect – is likely to be critical to the success of rational agents. In this paper we present a survey of the three best-known approaches to the control of reasoning in rational agents:

- *Continuous deliberation scheduling* (Boddy & Dean, 1989)
  This planning algorithm is based on the idea that an agent has a fixed set of decision procedures to react to events happening in the environment. The quality of the solution of a decision procedure depends on the time given to the procedure. Continuous deliberation scheduling is an algorithm that schedules decision procedures to achieve the highest overall satisfaction.
- *Discrete deliberation scheduling* (Russell & Wefald, 1991)
  This planning algorithm is based on the idea that at every moment in time an agent must deliberate or act. Discrete deliberation scheduling is an algorithm that decides, on the basis of the expected values of deliberation or action, whether to deliberate or to act.
- *Bounded optimality* (Russell *et al*., 1993)
  A perfectly rational agent bases its reasoning on decision theory, given what it knows of the environment. In practice, where agents reason in real time, this type of rationality is not feasible, thus we have to select a subset of these perfectly rational agents that are able to reason in real time. The agents in this subset are called bounded optimal agents, and behave as well as possible given their computational resources.

Research on resource-bounded agents originates in Simon's work (1982a) on bounded rationality, which investigates rationality assuming that the resources of the decision-maker are limited. All three of the algorithms discussed above are based on ideas from bounded rationality.

The remainder of this paper is structured as follows. In Section 2 we explain the theoretical background of the subject discussed in this paper: in Section 2.1 we set out the fundamentals of decision theory and introduce utility theory and probability theory; in Section 2.2 we discuss the origin of the research described in this paper, i.e. bounded rationality; and Section 2.3 explains how the concept of *meta-reasoning* – reasoning about reasoning – is used here. In Section 3 we discuss the three mentioned time-dependent planning algorithms and compare and contrast them. Finally, in Section 4 we tease out some common threads from the three approaches and present some conclusions.

2 Background

2.1 Decision theory

Decision theory makes a distinction between a *decision-maker* and an *environment*, and views the decision process as an interaction between these two. A parallel can be drawn with the concept of agency: the basic notion of an agent is a mapping from environment states to actions: depending on the current state, an agent's concern is to generate actions to respond to events in that state (Russell & Norvig, 1995). In decision theory, the decision process is essentially a mapping from environmental states to actions as well. But, additionally, decision theory imposes two important principles on the action space: the *principle of completeness* requires that all possible actions must be represented entirely, and the *principle of exclusion* requires that all actions must exclude the others, i.e. only a single action can be chosen at any point in time. We explain later in this section how decision theory accomplishes these principles. A mechanism that obeys these principles should be able to deal with complete decision situations in which the occurrence of future environmental states is certain. However, it cannot deal with *incomplete* decision situations. We consider two different kinds of incompleteness with respect to the available information about the action outcome space. First, *decisions in risk*: when there is risk involved in a decision situation, then for each future state the

available information includes the probability of occurrence of this state. Thus the outcome set is known and probabilities are assigned to every member of this set. Second, *decisions in uncertainty*: in this case, the set of possible future states is known, but no information is available about the individual probabilities of occurrence. Compared to the decisions in risk, this second kind of incompleteness contains even less information: the outcome space is known, but no probabilities can be assigned to its members.

In this section we formalise these thoughts on decision-making by discussing decision theory. We regard *decision theory* as the theory of rational decision-making (Wilson & Keil, 1999):

Definition 1. Rational decision-making *is choosing among alternatives in a way that* "*properly*" *accords with the preferences and beliefs of a decision-maker.*

The decision-maker can be an individual or a group making a joint decision. In this paper, we only consider cases in which the decision-maker is an individual. The subject of decision-making has received attention from a variety of research areas: economics and psychology (Von Neumann & Morgenstern, 1944) and decision and game theory (Luce & Raiffa, 1957). The most abstract notion of rational decision-making (Wilson & Keil, 1999) is concerned with alternatives, preferences that reflect the desirability of the alternatives, and rationality criteria. On a more concrete level, we are concerned with possible actions in the world, the desirability of states those actions would lead to and some ordering on desirability. The basis for decision theory is utility theory and probability theory. *Utility theory* provides us with a framework that formalises how desirable different states are, and *probability theory* provides us with a way to formalise the probability that the world will be in a certain state.

By the term "utility theory", we understand the following:

Definition 2. Utility theory *is concerned with the measurement and representation of preferences*.

The term *utility* refers to the measurement scale of preferences. Although in its initial definition, utility only applied to monetary values (Von Neumann & Morgenstern, 1944), it can be applied to all sorts of outcomes. The basis of utility theory is a preference order $\preceq$, which is normally a complete preorder over an outcome space O. The order of preferences is then represented by a *utility function* $U: O \rightarrow \mathbb{R}$. If y is weakly preferred over x, then we indicate this by $x \preceq y$, which can be represented by $U(x) \leq U(y)$; if y is always preferred over x, then $U(x) < U(y)$. Utility is thus measured on an ordinal scale (although represented cardinal).

When faced with a decision, we have to select a choice from a set of possible choices. A specific choice will have a certain relevant outcome o for the decision-maker (assuming the environment is deterministic). Let O be the set of all possible outcomes. If we make a choice, this will lead to a certain outcome. When a utility function is defined over all possible outcomes, we want to make a choice that leads to the outcome with the highest utility. However, if the outcome of a choice is not completely certain, we also have to take into account the probability distribution over the possible outcomes. A well-known decision theoretic model that captures exactly this idea is the *expected utility theory*: we calculate the expected utility of every possible choice and we make the choice with the highest expected utility. Let $P(o \mid c)$ denote the probability of outcome o given that the agent makes choice c and $U(o)$ be the utility of o. Then the expected utility $EU(c)$ of choice c is

$$EU(c) = \sum_{o \in O} P(o \mid c) U(o)$$

Under this simple and idealised model, a "rational" agent is one that chooses c so as to maximise $EU(c)$. Appealing though this model of rational choice is, it is frequently not consistent with the behaviour of individuals in practice: people seem to act irrationally from time to time. This apparently irrational behaviour of people in practice needs some explanation. At least two explanatory factors can be identified (from Gleitmann, 1991): *experience*, because while our mental machinery might only have some limited capacity for reasoning, this capacity needs experience to come to its full purpose; and our use of *heuristics and biases*, because while we might be educated with science and statistics,

we occasionally still use heuristics and fall prey to biases. This does not mean that individuals indeed are irrational; at worst, our rationality is bounded – resulting from circumstance, habit and limited processing capacity. This leads to the viewpoint that decision-makers in practice do not optimise, which gives rise to the study of "bounded rationality". Whereas a decision theoretic model gives us an *normative* view of decision-making, bounded rationality offers us a *prescriptive* one. In other words, decision theory aims to answer the question of *what* to decide and bounded rationality aims to answer the question of *how* to decide. Naive attempts to "implement" decision theory tend to be search-based, and such search-based algorithms are rarely usable if decisions are required in anything like real-time.

2.2 Bounded rationality

It is clear that when AI problems are moved towards reality, decision situations will become real-time. An important characteristic of a real-time decision situation is that the utility of an action varies over time. On the one hand, the utility might *increase* over time, e.g. the output quality of an anytime algorithm (described in Section 3.1) increases when it is executed for longer. On the other hand, the utility might *decrease* over time, e.g. when the agent misses an opportunity like a deadline. An exact method to resolve a decision situation is inherently intractable, because it typically involves search, which is usually intractable. Instead, an agent uses an algorithm with one of two properties: *approximation*, where the solution is guaranteed to be within some $\in$ of the optimal solution, or *probably correct*, where the algorithm will return the optimal solution with probability p. Research on these kinds of algorithm, generally referred to as *bounded rationality*, was initiated by Simon (1982a) in the early 1950s. This work is described in this section. In the 1960s Good (1971) distinguished a "type II" rationality from classical "type I" rationality. Type II rationality maximises expected utility taking into account deliberation costs.

In this paper, the term "bounded rationality" is used for indicating the field of research that is concerned with the problems that real-time agents are faced with: finite computational power and finite time in which to reason. Bounded rationality is defined as follows (Wilson & Keil, 1999):

Definition 3. Bounded rationality *is rationality as exhibited by decision-makers of limited abilities*.

Bounded rationality has received attention from a variety of research fields: economics (Simon, 1982c), philosophy (Dennett, 1987), cognitive science (Wilson & Keil, 1999) and artificial intelligence (Russell, 1997). It is closely related to the disciplines of rational decision-making, decision theory, utility theory and meta-reasoning. As mentioned previously, rational decision-making is choosing among alternatives in a way that corresponds with the preferences and beliefs of a decision-maker, but implicitly assumes infinite time, infinite computational power and complete information. Simon (1982b) proposed a behavioural model of rational choice, the "administrative man", which replaces the concept of "economic man" with a kind of rational behaviour that is compatible with the access to information and the computational capacities that organisms (including man) possess. Experimental evidence for this model is given in Simon (1982c). It documents cases in which decision-makers do not live up to the ideal of rational decision-making. Ideal probability and utility distributions imply a degree of omniscience that is incompatible with the psychological limitations of the organism. Organisms adapt well enough to "satisfice" (concatenation of *satis*fy and suf*fice*); they do not "optimise" as postulated by decision theory (Simon, 1982c). Simon presented a simulated organism that could survive in its environment, despite the fact that its perceptual and choice mechanisms were very simple. In the same work, he identified some structural characteristics that are typical of the "psychological" environments of organisms, which refer to the relation between the organism's goals and access to information in the agent's environment. Examples of such characteristics are that access to information and the environment limit the planning horizon for the organism, and that the organism's needs and the environment separate "means" from "ends" very naturally. Some of these characteristics reappeared some 30 years later in the agents literature on environmental properties (Russell & Norvig, 1995):

- *Accessible vs. inaccessible*
  If an agent has access to the complete state of the environment, the environment is accessible.
- *Deterministic vs. non-deterministic*
  If the next state of the environment is uniquely determined, the environment is deterministic.
- *Episodic vs. non-episodic*
  If the agent's experience is divided into "episodes" which consist of perceiving and acting, and episodes do not depend on past or future episodes (regarding quality and action), the environment is episodic.
- *Static vs. dynamic*
  If the environment cannot change while an agent is deliberating, the environment is static.
- *Discrete vs. continuous*
  If there are a limited number of distinct, clearly defined perceptions and actions, the environment is discrete.

A subtle distinction must be made between bounded rationality as used in economics or cognitive science and the way we use it here. In economics and cognitive science, bounded rationality is used to explain why human behaviour is irrational. In this paper, bounded rationality is more narrowly defined: it only addresses the issue of resource bounds (in terms of time and computational power) in decision-making. When designing agents that are in continuous interaction with a real-time environment, we have to account for the fact that these agents have to consider their own resources. An agent cannot deliberate indefinitely, because at some point in time it has to act. Agents have to be able to control their deliberation in order to seize opportunities and to act effectively in their environment. A well-known mechanism that could be used as control is meta-reasoning, which is discussed in the next section.

2.3 Meta-reasoning

Meta-reasoning, or meta-level reasoning, means "reasoning about reasoning" (Wilson & Keil, 1999). Meta-level reasoning is distinguished from its counterpart, *object-level reasoning*. Object-level reasoning is deliberation about *external* entities, e.g. considering which action to take, whereas meta-level reasoning is deliberation about *internal* entities, e.g. deciding whether it is worth deliberating about a specific action. If the universe of discourse is a game of chess, object-level reasoning might for example be concerned with which opening move to make and meta-level reasoning with deciding whether it is worth deliberating about which opening move to make. Russell gives the following definition of meta-reasoning (Wilson & Keil, 1999):

Definition 4. Meta-reasoning *is any computational process that is concerned with the execution of other computational processes within the same agent*.

Meta-reasoning serves two important purposes in an intelligent agent (Wilson & Keil, 1999). First, it gives the agent control over its (object-level) deliberation. Second, it increases the flexibility of the agent in the way it enables the agent to recover from errors in its object-level deliberation.

We touch upon using meta-reasoning for agents by describing three meta-reasoning architectures (from Russell & Wefald, 1991): TEIRESIAS, MRS and SOAR. TEIRESIAS (Davis, 1982) is built upon a MYCIN-type rule-based system (Buchanan & Shortcliff, 1984a) and provides explanation, knowledge acquisition and strategy knowledge facilities to control the reasoning in MYCIN. Like MYCIN, TEIRESIAS is used for giving consultative advice on diagnosis and therapy for infectious diseases. One limitation of MYCIN is that it is not able to deal with time considerations. For example, when it takes 48 hours to positively identify whether some specimen is infected, MYCIN cannot make a decision before those 48 hours have passed. However, in reality, a physician often must make a decision based on early evidence of bacterial growth in the specimen. In that case, TEIRESIAS can assist the physician by explaining why MYCIN waits 48 hours and then the physician can take appropriate action to further the reasoning in MYCIN. The meta-reasoning level in TEIRESIAS decides which rule to execute in MYCIN. In TEIRESIAS, object-level rules are given a value; the

concept of values is used for comparing various possible computation steps. The meta-level rates values of applicable object-level rules and can decide that some rule should be applied. Meta-reasoning in TEIRESIAS is done over the values of object-level rules (or possible computation steps) and is not used for describing the outcomes of computation steps.

In the MRS architecture (Genesereth & Smith, 1981), meta-reasoning selects computational tasks and methods to be carried out. Task selection happens through a preference mechanism similar to the one used in TEIRESIAS: a task is selected if no runnable task is preferred to it. There is a method selection component in MRS that reduces abstract tasks to concrete procedures; these procedures are directly executable whereas abstract tasks are not. Reasoning about preferences is done using a backward-chaining theorem-prover (Russell & Norvig, 1995) and object-level inference steps are selected depending on the probability of success of a proof beginning with the given step.

The SOAR (Newell *et al.*, 1989) system uses a goal-based execution architecture: reasoning is done in problem states defined by a goal, initial state and set of operators. Example domains of SOAR include the well known Eight Puzzle, industrial expert systems, natural language parsing and AI weak methods (such as and/or search and hill-climbing). In SOAR, operators are selected in three stages: (1) elaboration, in which all the rules in long-term memory are triggered, if matching the current state, to provide preferences; (2) decision, in which preferences are resolved to select an operator to apply; and (3) subgoaling – if resolution is unsuccessful, the system tries to solve it and commits itself to this subgoal. The automatic subgoaling selection method is the most innovative feature in SOAR. The key point about SOAR is that it can conduct a simulation of object-level computation steps in order to select among them, unlike TEIRESIAS and MRS. However, the SOAR architecture cannot reason with uncertainty and the model SOAR uses for computational actions cannot trade time for accuracy in solutions. The ability to trade time for accuracy in solutions is possibly the most important issue when we want to use meta-reasoning to model a resource-bounded agent.

2.4 Resource-bounded agents

We conclude this section with a brief overview of issues involving the resource-boundedness of agents. We return to these issues in the next section, in which we discuss frameworks for time-dependent planning. We show in the next section that every framework approaches the issues mentioned here from different perspectives.

To start off on the highest level of abstraction, we are concerned with a *system*. The components in such a system are *agents*[1] and an *environment*. In this paper, we are not concerned with systems containing more than one agent, i.e. multi-agent systems, but limit ourselves to single-agent systems. The interaction between an agent and its environment is meant for the exchange of information.[2] This view corresponds with the most-accepted definition of an agent. The two primary issues that are characteristic for situated resource-bounded agents are *time* and *access to information*. Both these issues are to be taken into consideration by the agent, but are properties of the environment. This concludes the issues that are *external* to the agent.

The two main processes within the agent are *reasoning* and *acting*. We distinguish two kinds of reasoning: *deliberation* – reasoning about what to do, and *means-ends* – reasoning about how to do it. The components that enable deliberative reasoning are an *evaluation mechanism* – on the basis of which the agent decides what to do, and a *control mechanism* – which enables the agent to deal with time-dependency. The evaluation mechanism is typically utility-based. However, it is open to discussion over what one should define the utility, e.g. previous models have defined utilities over environment states, actions, action histories and action sequences. The control mechanism is typically based on some sort of meta-reasoning.

[1] In a decision-theoretic context, an agent is called a *decision-maker.*

[2] This issue is open to discussion: for example, decision theory views the system's decision process as the interaction between an agent and its environment. However, this means the decision process is external to the agent, which we regard as unintuitive.

The main issue we then concern ourselves with in this paper is how an agent deals with time-dependency, which is the essential issue underlying the frameworks discussed in the next section. These frameworks assume that information about *deadlines* is available, on the basis of which optimal actions (or action sequences) have to be generated. A disadvantage of this assumption is that it makes the frameworks only directly applicable in deliberative agents, i.e. reactive behaviour – the immediate responding to events – cannot be easily modelled using this approach. This means that in order to model a wider range of agents – including reactive agents – these frameworks have to be changed.

To summarise, in this section we discussed some background issues in the control of reasoning in resource-bounded agents. The popular way to theorise about decision-making in agents is classic decision theory. A limitation of decision theory is that it does not gives us any method for *how* to make a decision; it merely tells *what* to decide. All naive ways of implementing classic decision theory tend to be search-based and therefore inherently intractable and not likely to be very useful in practice. Both the decision-maker and the environment constrain the decision process: the decision-maker has bounded resources, specifically computational power, and a real-world environment is obviously real-time, i.e. decisions have to be made within a certain amount of time. This means the decision-maker must control its decision-making, or, in a more general context, control its reasoning. To do this, it needs to reason about reasoning, i.e. it needs to meta-reason.

3 Time-dependent planning

Time-dependent planning is concerned with determining how best to respond to predicted events when the time available to make such determinations varies from situation to situation (Boddy & Dean, 1994). The intuition behind time-dependent planning is that an agent should make optimal use of its available time. In this section, we discuss three time-dependent planning frameworks. Two are based on the idea of scheduling the necessary deliberation – continuous and discrete; the third, bounded optimality, extends discrete deliberation scheduling to apply it to agents. Throughout the discussions of these frameworks, we use the TILEWORLD planning scenario (Pollack & Ringuette, 1990) to illustrate how to apply the frameworks. Finally, we discuss the Belief-Desire-Intention (BDI) agent architecture and the general Markov Decision Process (MDP) planning framework, in which time-dependent planning algorithms can be applied.

3.1 Continuous deliberation scheduling

In Boddy & Dean (1989), a framework is introduced called *expectation-driven iterative refinement*. This framework enables one to construct solutions to time-dependent planning problems. The planning in this framework is done using a set of decision procedures called *anytime algorithms*. A *decision procedure* is a procedure used by an agent to select an action which, if executed, changes the world (Zilberstein 1996). Two types of action are distinguished. The first are *inferential*, which denote purely computational actions. The second type are *physical* actions, that change the state of the external world, and may require some computation. Essentially, anytime algorithms are algorithms whose quality of results improves monotonically as computation time increases. The characteristics of these algorithms are (1) they can be suspended and resumed with negligible overhead; (2) after termination at any point, they will return an answer; and (3) answers returned improve in some well-behaved manner as a function of time. Many conventional algorithms satisfy these characteristics. Zilberstein (1996), for example, shows how the solution of randomised tour improvement (Lawler, 1985) to the Travelling Salesman Problem (TSP) can be used as an anytime algorithm.

Boddy and Dean investigated the randomised tour improvement as an anytime algorithm in a problem involving a robot courier assigned the task of delivering packages to a set of locations. The robot's only concern here was time: it tried to minimise time consumed deliberating about what to do. This task actually involves two primary tasks: tour improvement and path planning. Anytime algorithms are employed for solving both problems and statistics are gathered on their performance to be used at run time in guiding deliberation scheduling. The process of *deliberation scheduling* is the

"explicit allocation of computational resources based on the expected effect of those allocations on the system's behaviour" (Boddy & Dean, 1994). Deliberation scheduling is accomplished by a sequence of allocation decisions made by the system as time passes, as events happen and as new information becomes available. The gathered statistics represent the performance of the algorithm and are called *performance profiles*. A performance profile of an anytime algorithm is the expected output quality as a function of run time (Zilberstein 1996).

Figure 1 shows a simple example of continuous deliberation scheduling. The problem in this example is as follows: assume the current time is $\hat{t}$ (in Figures 1.iii and 1.iv indicated by "now"), at which the agent has two events to respond to, c_1 and c_2, respectively, and construct a schedule for deliberation to respond to those events, maximising the quality of the responses. Figures 1.i and 1.ii show the performance profiles for the decision procedures for c_1 and c_2 respectively. Figure 1.iii shows the allocation of time per decision procedure after executing the continuous deliberation scheduling algorithm, and Figure 1.iv shows the allocation of time after collecting and sorting the time slices for the decision procedures for c_1 and c_2. The algorithm works from right to left and starts by allocating all time between c_1 and c_2 to the decision procedure for c_2, because there is no use in spending time on the decision procedure for c_1 when event c_1 has occurred. The algorithm then decides which decision procedure to allocate time for, based on the increase in quality of the solution of the decision procedures. This is done for an interval of time over which the increase is continuous. In this example, time is allocated first to the decision procedure for c_1, then for c_2, for c_1 and the rest for c_2. This is shown in Figure 1.iii. After all available time is allocated, the time slices are collected and ordered and the agent can start executing the anytime algorithms, which results in the graph shown in Figure 1.iv. Here, the agent executes the decision procedure for c_1 until a certain moment and then starts executing the decision procedure for c_2. Because the decision procedures are anytime algorithms, the agent can respond to event c_1 by giving the solution it reached when the decision procedure for c_1 was stopped.

In the remainder of this section, we explain how Boddy and Dean formalise continuous deliberation scheduling. This formalisation results in a deliberation schedule procedure *DS*, that is described below. An important assumption Boddy and Dean make is that at any moment in time the agent knows about

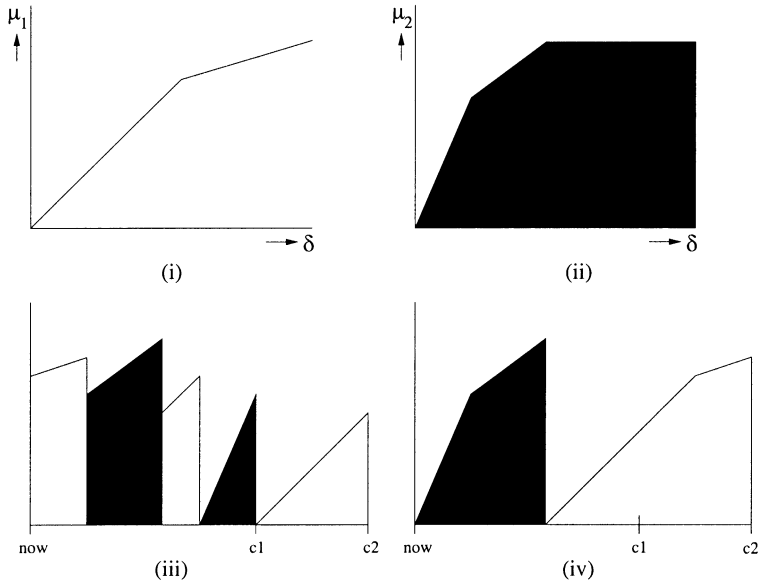

Figure 1 A simple example of continuous deliberation scheduling (from Boddy & Dean, 1994). Assume that an agent has two future events, c_1 and c_2, to respond to, and aims to maximise the value of both responses. The agent has at its disposal decision procedures for responding to events c_1 and c_2. In (i) and (ii) the respective performance profiles of these decision procedures are shown. Such a performance profile defines the expected response value (μ_c) as a function of the allocated time (δ) for that decision procedure. Panel (iii) shows the allocation of time for the respective decision procedures, resulting from the continuous deliberation scheduling algorithm. Panel (iv) shows the final deliberation schedule, after collecting and sorting the timeslices

a set of pending conditions it has to respond to.[3] A second assumption they make is that the value of the agent's response to one condition is independent of the response to other conditions. The decision about how long to execute which decision procedure is based on the values of the responses to conditions. Let *Response*(*c*) denote the response to condition *c*, and let *V*(*Response*(*c*) | *c*) be the value of responding to condition *c* with *Response*(*c*) given the occurrence of *c*. The total value of the response to a set of conditions *C* is the sum of all values of the conditions, given that that condition occurs:

$$\sum_{c \in C} V(Response(c) \mid c)$$

The value of a response is the output quality of a performance profile and, in terms of decision theory, denotes the utility of a response. The agent has a decision procedure *dp*(*c*) for each condition $c \in C$. A decision procedure is an anytime algorithm (as described above). A performance profile (as described above) for this algorithm is formalised by a function $\mu_c : \mathbb{R} \rightarrow \mathbb{R}$. An example of such a function is shown in Figure 1.i. Let *alloc*(*δ*, *dp*(*c*)) denote the allocation of δ time units to *dp*(*c*) and let *V*(*Response*(*c*) | *c*, *alloc*(*δ*, *dp*(*c*))) be the value of the response to *c*, given the occurrence of *c* and the allocation of δ time units to *dp*(*c*) to calculate *Response*(*c*). Then μ_c takes the amount of time, δ, and returns the expected value of the response to *c* with *dp*(*c*) running for the specified amount of time:

$$\mu_c(\delta) = E(V(Response(c) \mid c, alloc(\delta, dp(\mathrm{c}))))$$

Because we assume that the result quality of anytime algorithms improves monotonically as run time increases and performance profiles denote the expected output quality of run time, the results of performance profile functions will also improve monotonically. This means that performance profile functions have *diminishing returns*, i.e. slopes of consecutive line segments must be decreasing: $\forall c \in C : \exists f,\ \mu_c(t) = f(t)$ such that *f* is monotonically increasing, continuous and piecewise differentiable and $\forall x, y \in \mathbb{R}^+$ such that $f'(x)$ and $f'(y)$ exist, $(x < y) \Rightarrow (f'(y) \leq f'(x))$.

The idea behind the *DS* procedure is to work backwards, starting at the time of occurrence of the last condition to respond to. Let $C = \{c_1, \ldots, c_n\}$ be the set of conditions to be responded to. Let *time*(*c*) be the time of occurrence of condition *c*, and let $\wedge(t)$ be the set of all conditions whose time of occurrence is later than some particular time *t*:

$$\wedge(t) = \{c \mid (c \in C) \wedge (time(c) \geq t)\}$$

Next, let *last*(*t*) be the time of occurrence of condition *c* that is not in $\wedge(t)$:

$$last(t) = \max\{time(c) \mid c \in (C - \wedge(t))\}.$$

In *DS*, *t* is initially set to $+\infty$. In that case, we actually do not need to be concerned with $\wedge(t)$, because there are no conditions with time of occurrence later than $+\infty$. Clearly, if *t* takes any value other than $+\infty$, then it is possible there are conditions with a time of occurrence later than *t* – these conditions are taken into account using $\wedge(t)$.

Next, we have to decide how to allocate processor time to decision procedures. This is based on the expected gain in value for the decision procedures for conditions in $\wedge(t)$. Let $\gamma_i(x)$ be the gain of the *i*th decision procedure having already been allocated *x* amount of time. The term $\gamma_i(x)$ is formulated as the slope of the linear segment of μ_i at *x*. If μ_i is discontinuous at *x*, then $\gamma_i(x)$ is the slope of the linear segment at the positive side of *x*. Finally, let *min_alloc*($\{\delta_i\}$) be the minimum of time interval

[3] The term *event* is used interchangeably with *condition* in the formalisation, because an event can be viewed as a condition for the agent to act.

lengths for the next linear segments for performance profiles given the time allocated thus far. This is required, because when selecting decision procedures, the current gains have to be constant.

Now we can introduce the deliberation schedule procedure *DS*. *DS* consist of three main loops:

1. Initialise allocation variables (use $last(-\infty)$).
2. Decide how much time to allocate per decision procedure (use $last(t)$, $min\_alloc(\{\delta_i\})$, $\gamma_i(\delta_i)$).
3. Decide when to run the decision procedures: for all selected decision procedures, run the ith decision procedure from t till $t + \delta_i$ and increase t by δ_i.

Boddy and Dean prove (1994) that *DS* is optimal, i.e. it generates $\{\delta_i\}$ which maximises $\Sigma_i^n \mu_i(\delta_i)$. For the detailed formulation of *DS* and the optimality proof, we refer the reader to Boddy & Dean (1994).

The deliberation schedule procedure *DS* assumes that there is no uncertainty about the times of occurrence of events. Boddy and Dean (1994) describe an extension of *DS*, which assumes there is only uncertain information about the times of occurrence of events. This procedure is called *DS′*. In *DS′*, conditions are assumed to have probability distributions defined over them and to have some earliest and latest time of occurrence. Then the total value of the response to a set of conditions is the combination of the probability that some event happens and the value of the response to that event at the time of occurrence. As *DS* allocated processor time intervals to a decision procedure, we now have to account for the possibility that the event, corresponding to the decision procedure, might already have occurred. *DS′* computes an optimal sequence of processor time allocation for which the sum of *expected* values of responses to the conditions to occur is maximal. An important difference between *DS* and *DS′* is that *DS′* works with processor allocation *windows* instead of allocating all processor time at once. This *multi-pass* approach enables *DS′* to take advantage of the fact that it is no use deliberating about an event after its occurrence. Again, a detailed description of *DS′* is given in Boddy & Dean (1994).

To conclude, the continuous deliberation scheduling procedure discussed in this section originated in research on search algorithms. It defines a class of search algorithms, called anytime algorithms, with certain properties, like increasing solution quality over time. Boddy and Dean argue that most search algorithms can be implemented as anytime algorithms. These algorithms form the basis of the continuous deliberation scheduling procedure: they are the decision procedures that enable an agent to respond to events in its environment. When the agent has a set of future events it needs to respond to, and decision procedures are available to respond to the events, the procedure schedules the agent's deliberation by maximising the quality of the response for each single event. We later discuss how the procedure relates to the other algorithms discussed in this paper.

Example. We briefly illustrate the continuous deliberation scheduling procedure in an examplar TILEWORLD planning scenario. The TILEWORLD is an agent testbed that is used for experimentally evaluating agent architectures. Consider the following description which explains the TILEWORLD scenario:

> The TILEWORLD consists of an abstract, dynamic, simulated environment with an embedded agent. It is built around the idea of an agent carrying "tiles" around a two-dimensional grid, delivering them to "holes", and avoiding obstacles. During the course of a TILEWORLD run, objects appear and disappear at rates specified by the researcher. The objects include tiles, holes, obstacles, tile stores and a gas station. The "lifetime" of any given object is determined by user-specified appearance and disappearance rates for that type of object. The researcher can also specify other properties of the objects, such as their size and score. The agent's primary task is to fill holes with tiles. To do this, it must pick up tiles, either from a tile store or from wherever it has previously dropped them, carry the tiles to a hole, and deposit a tile in each cell of the hole. If the agent successfully fills all the cells in the hole with tiles that match the hole's shape, it is awarded with the full amount of the hole's score. A lesser score is received for filling the holes with non-matching

> tiles. The agent is responsible for maintaining its fuel level. It consumes fuel as it moves around the world; the more tiles it is carrying, the more quickly it burns fuel. To obtain more fuel, it must travel to the gas station and fill its tank. If the agent runs out of fuel, it cannot move for the duration of the run.
>
> (Pollack *et al*., 1994)

The robot courier example that Boddy and Dean used to illustrate continuous deliberation scheduling, as mentioned above, can be easily represented in the TILEWORLD scenario. Assume that some number of holes are scattered around in the world and the agent has the task of delivering tiles to the appropriate holes; the agent is currently in the tile store and must deliver its tiles to the holes. This is a typical Travelling Salesman Problem – which rules out a brute force approach to the problem. We use continuous deliberation scheduling to find a suitable solution. The agent cannot deliberate indefinitely, because of the dynamic structure of the TILEWORLD – holes might disappear before the agent reaches them.

The two problems the agent now faces are, first, *tour improvement*, constructing a minimal distance tour that brings it to all holes in the world and, second, *path planning*, figuring out for every hole the fastest way to get from that hole to the next hole in the tour. For both problems we assume the agent has at its disposal an anytime algorithm with a performance profile with the desired performance profile properties, i.e. monotonically improving quality over time and diminishing returns. For reasons of space, we assume that the performance profile for tour improvement is given as performance profile μ_1 as shown in Figure 1.i and the profile for path planning μ_2 as shown in Figure 1.ii. We also use events c_1 and c_2 from Figure 1 here. Assume that c_1 indicates the earliest time that the courier might have to start delivering, and c_2 the latest time that he has to start delivering. At c_1, the courier must thus have some tour readily available for execution, although that tour might be far from optimal. At c_2, the courier has no time left to improve the delivery time and has to start delivering. The *DS* algorithm then works as follows. It starts allocating time for the algorithms backwards, starting from the time point of the last event, here c_2. It continually selects an amount of time, selects the algorithm to allocate this time to, and adds the selected time to the execution time for the selected algorithm. The methods of selection are explained in detail above and in Boddy & Dean (1994). Basically, the time between c_1 and c_2 is allocated to the path-planning algorithm, since no tour improvement can be done after c_1. The time from the current time to c_1 is then divided up in turns between tour improvement and path planning, depending on which algorithm gives the maximum gain. When *DS* has finished, the deliberation schedule resembles Figure 1.iv.

Having completed its deliberation scheduling, the agent is still in front of the tile store, and now starts executing the algorithms by using the deliberation schedule just computed. Then, at some time between c_1 and c_2, the agent has to start delivering by executing the tour as computed by the algorithms. Continuous deliberation scheduling is thus used as a meta-reasoning method to control the search algorithm for finding a best tour, given time constraints.

3.2 Discrete deliberation scheduling

The term *discrete deliberation scheduling* comes from Boddy & Dean (1994). It denotes a kind of allocation of deliberation which treats deliberation as being divisible into discrete chunks, such that the allocation of each chunk is a separate decision. Work on this subject has been carried out by Russell and Wefald (1991) and Etzioni (1989). We discuss the work of Russell and Wefald here. This work preceded Russell's theory on bounded optimality, which is discussed in Section 3.3.

The idea behind discrete deliberation scheduling is that at any moment in time, the agent has to choose between performing a default action α and a computational action from a set of computational actions S_j. Performing a computational action might cause the agent to change its default action. In this way, computations are treated as actions, and are selected on the basis of their expected utilities. The utility of a computation depends on the passage of time (because of possible changes in the environment) and the possible revision of the agent's intended actions in the real world. Then it follows

that the utility of an action outcome is uncertain, since we do know beforehand how the environment changes. We assume that the outcome of *external* actions is known.

Russell and Wefald (1991) distinguish three progressively more specific models of deliberation that form the foundation of their theory:

- *External model*
  Analyse the system as an external object by ascribing utilities and probabilities to the system's actions and internal states. The goal of further computation is to refine the choice of the default action.
- *Estimated utility model*
  An agent might select its current best action by making explicit numerical estimates of the utilities of actions. The best action is then the action that currently has the highest utility estimate. Further deliberation is done in order to revise and refine utility estimates.
- *Concrete model*
  Here, the decision algorithm is specified as far as the results of a computation step revise the agent's intended action. Russell and Wefald implemented meta-reasoning systems up to the concrete model in forward-search programs – programs that revise the utility estimates for outcome states by generating and evaluating their successors. Because this model involves the object-level reasoning of the agent, we do not discuss it in much detail in this paper.

In this section, we progressively build up to the concrete model by discussing the three models. Throughout the discussion we use an example, similar to one presented in Russell and Wefald, which is as follows:

> A CEO is faced with an unpopular management policy, for example the closing down of factories, and might decide to run a coarse-grained simulation model; if the results are equivocal, a more detailed model might be run, but eventually the policy will have to be executed.

Two important assumptions are made. The first is that the outcomes of external actions are known at the time when the agent is choosing among them. The second is that the utility of each outcome state is not immediately known, but some computation might be necessary. Decision theory tells us the agent should choose the action which maximises the agent's expected utility, as discussed in Section 2.1. Let A be the set of possible actions, assume some action $A_i \in A$, and let $[A_i]$ be the world state that results from taking action A_i in the current state. Let $P(W_k)$ denote the probability that the current state is W_k and $[A_i, W_k]$ is the result of taking action A_i in world state W_k. The expected utility of an action is then

$$E[U([A_i])] = \sum_k P(W_k)\,\mathrm{U}([\mathrm{A}_i, \mathrm{W}_k])$$

In the management example, this would mean that the action is chosen with the best outcome. However, this outcome might still be considered bad, e.g. cutting spending. The calculation is easy to perform for physical actions, but slightly more complicated for computational actions: when calculating the expected utility for computational actions, we have to take into consideration the fact that the world changes during computing and that the agent's future action might change. This translates into the value of a computational action being the utility of the computation itself minus the utility of the current default action. In our example, if the simulation takes a week, then its value is the difference between doing α (cutting spending) now and closing down factories a week later. We call this value the *net value* of a computational action S_j and define it as follows:

$$V(S_j) = U([S_j]) - U([\alpha]).$$

But it is not certain that a computation will immediately result in an action; a distinction must be made between *complete computations*, which result in a commitment to an external action, and *partial computations*, which do not result in a commitment. In the example, if the simulation results in one

single decision, it is a complete computation; if it does not, it is partial. When a computation is complete, the utility of S_j is solely the utility of the action committed to after the computation: α_{S_j}. *Let* $[\alpha_{s_j}, [S_j]]$ denote the outcome state of action α_{s_j} that resulted from computation S_j. Hence

$$V(S_j) = U([\alpha_{s_j}, [S_j]]) - U([\alpha])$$

When the computation is partial, it will change the internal state of the agent, which affects the value of further computational actions. We thus have to define the utility of the internal state in terms of how it affects the agent's ultimate choice of action – the expected utility of the action the agent ultimately takes, given its internal state. We thus have to take into account all possible computation sequences following S_j. Let a computation sequence be represented by T and let the external action resulting from T be denoted by α_T. Let $P(T)$ be the probability that the agent will perform T. Then

$$U([S_j]) = \sum_T P(T)U([\alpha_T, [S_j, T]]).$$

The problem stated above translates to taking the action (being either physical or computational) with the maximum expected utility from the set $\{\alpha, S_1, \ldots, S_k\}$. The *ideal control algorithm* is then defined as follows:

1. Keep performing that S_j with highest expected net value, until none has positive expected net value.
2. Commit to action α.

In a real-time environment, we are concerned with the time cost of computational actions, that is, we want to capture the dependence of utility on time as the *cost of time*. In our formalisation so far, this cost has been included implicitly in the utility function of the agent. In order to make the analysis less complicated, we want to represent the time cost explicitly. Therefore we have to distinguish between the *total utility* and the *intrinsic utility*: the total utility is the time-dependent utility of an action; the intrinsic utility is the utility of an action if it is performed immediately. Let U_I denote the intrinsic utility and let C express the difference between the total and intrinsic utilities. The total utility and intrinsic utility are related as follows:

$$U([A_i, [S_j]]) = U_I([A_i]) - C(A_i, S_j)$$

We can draw a parallel with the previous section on continuous deliberation scheduling. This function could be an anytime algorithm: it defines in what way the utility of an action is discounted over time. But note that here the utility of an action normally decreases as time progresses, whereas the utility of an action in an anytime algorithm increases. If we want the cost of time to be independent of the agent's choices, we require the identity of the best action to remain fixed over time. This means that the agent's optimal action is always that with the highest intrinsic utility, and it suffices to require that the cost of the computation is independent of the action under evaluation:

$$U([A_i, [S_j]]) = U_I([A_i]) - C(S_j).$$

Furthermore, the utility of an action that occurs during S_j only depends on the length of S_j (in elapsed time) and the course of events happening in the world during that time (because computations only change the internal state). In our example this means that while running the simulation model, events happen in the world; however, one cannot respond to these events, because the simulation is still in progress. The computation S_j will not affect that course of events, thus S_j only depends on its own length. We can then calculate the time cost TC of S_j as a function of its length, denoted by $|S_j|$. Thus TC gives the loss in utility when delaying an action. Hence

$$U([A_i, [S_j]]) = U_I([A_i]) - TC(|S_j|).$$

Alternatively, we can easily separate the benefit from the cost of a computation. The net value of a computation is the difference between (1) the action that results from the computation and the current

default action, i.e. the *benefit* of the computation, and (2) the time-cost of the computation, i.e. the *cost* of the computation. Now, let $\Delta(S_j)$ denote the *estimated benefit* of the computation:

$$\Delta(S_j) = U([\alpha_{S_j}]) - U([\alpha])$$

Then it is possible to rewrite the definition of the net value for a complete computation as its benefit minus its cost:

$$V(S_j) = \Delta(S_j) - TC(|S_j|).$$

To summarise, the model developed so far gives us a way of formalising the decision-making process under certain assumptions. This can be illustrated using our management example. The basic idea is that the decision-maker (the CEO) is faced with doing a physical action (closing down factories) or a computational action (running the simulation model). For now, we assume that we know the utilities of the possible outcomes of actions, i.e. we know what conclusions to draw from the results of the simulation model – we later drop this assumption to maintain consistency with the claim that the model assumes we do not know the utilities. However, we do not necessarily know the outcomes of internal actions, i.e. we do not know the outcome of the simulation, otherwise we would not need to run it. We keep performing the computational action with the highest expected value until no computational actions have a positive expected value; we then commit to performing the physical action. We distinguish between complete (immediately resulting in a physical action) and partial (not resulting in a physical action) computations and we are able to explicitly represent the loss in action utility over time.

An important assumption in the external model, as described above, is that utilities are available at the time the agent must make a choice. But this assumption is hardly feasible for a "realistic" agent, i.e. an agent that is non-omniscient and resource-bounded. If we do not assume the utilities to be available, the agent has to *estimate* utilities before making a choice. This refines the external model and is called, as mentioned before, the estimated utility model: we replace the utility function U by a function $\hat{U}$ that represents the estimated utility function. We assume that the object level has a *current estimate* of the utility of each action. Let the computation sequence to date be S and the evidence generated by that sequence be e. Then the utility estimate of action Ai after computation S is then:

$$\hat{U}^S([A_i]) = E(U([A_i])\,|\,e).$$

Let $S\,.\,S_j$ denote carrying out computation S_j after computation sequence S and let e_j denote the evidence generated by S_j. When computation S_j has been carried out:

$$\hat{U}^{S\,.\,S_j}([A_i]) = E(U([A_i])\,|\,e \wedge e_j).$$

But this function implicitly incorporates the cost of time and, as in the external model, we want to be able to represent this explicitly. Therefore, assuming a time cost is available, the expected value of complete computation S_j, given evidence e, is

$$\hat{V}([S_j]) = E[(U([\alpha_{S_j}]) - U([\alpha]))\,|\,e \wedge e_j] - TC(|S_j|).$$

This value resides in the probability distributions for the effect of evidence for the external actions. Let $u = \{u_1, \ldots, u_n\}$ where u_1 to u_n are new utility estimates for actions A_1 to A_n. Let $p_j(u)$ be the joint probability distribution for the new estimates; this results in a probability distribution over the new utility estimates for every action. Thus there exists such a probability distribution for the current best action α, since $\alpha \in \{A1, \ldots, A_n\}$. Let this probability distribution be denoted by p_{α_j}. Finally, let $\max(u) = \max\{u_1, \ldots, u_n\}$. Then we have:

$$E[\hat{V}(S_j)] = \int_u \max(u)\, p_j(u)\, du - \int_{-\infty}^{\infty} u p_{\alpha j}(u)\, du.$$

This equation says that the expected value of a computational action is the expected value of the external action with maximum utility minus the expected utility of the current best action; this idea

agrees with what we have formally represented in the discrete deliberation scheduling model. Probability distributions may be obtained by gathering statistics on past computations in the same way we obtain performance profiles of anytime algorithms by gathering statistics on past computations.

The estimated utility model thus drops the assumption that utilities are available at the time an agent makes its choice. However, assessing the expected value of all continuations of a computation is in practice still infeasible, because computations can be arbitrarily long. By making two simplifying assumptions, we avoid important issues concerning the tractability of the model. The first assumption is that the algorithms used are *meta-greedy*, in that they consider single primitive steps, estimate their ultimate effect and choose the step appearing to have the highest immediate benefit. The second assumption is the *single-step assumption*: a computation value as a complete computation is a useful approximation to its true value as a possibly partial computation.

The next step of refinement – the concrete model – depends on the domain in which the model is applied. Russell and Wefald applied the model in search algorithms used in game-playing programs. The concrete model makes assumptions about the object-level decision mechanism, and not about the meta-reasoning mechanism. The meta-reasoning mechanism is specified completely in the external model and the estimated utility model. In the concrete model, the object-level reasoning is structured in a way that makes it suitable for meta-level control. However, the focus in this paper is on the modelling of the meta-level reasoning mechanism and not on the object level, and therefore a discussion of the concrete model is beyond the scope of our interest.

To conclude, the discrete deliberation scheduling model discussed in this section basically gives a decision-maker the choice between executing a physical action and a computational action at any moment in time. A physical action changes the external state of the system, i.e. the environment of the agent, whereas a computational action only changes the internal state of the agent. The model is decision-theoretic in that it bases the agent's decision-making process on the expected utilities of actions, being either physical or computational. An important property of the model is that it enables the agent to explicitly represent its knowledge about the relation between action and time. We show later how discrete deliberation scheduling relates to the other algorithms discussed in this paper.

Example. Assume the TILEWORLD as explained in Section 3.1. In the context of discrete deliberation scheduling, we first have to distinguish between external actions and computations of the agent. Let an external action be a move by the agent (up, down, left, or right) and let a computation be the planning of a path to some location in the world, typically with a hole. We let utility be represented as the result of an order-reversing mapping on the distance between the agent and the hole it is currently closest to. It is clear that the agent tries to maximise its utility. Let a world state be a cell in the grid of the TILEWORLD. The agent adopts some default external action, i.e. to move up, down, left or right. As explained above, the purpose of a computation is to revise the default action, presumably to a better one. This is obvious here; without computation the default action is simply a random move action, but a computation leads to actions that direct the agent to a hole.

The discrete deliberation scheduling framework enables us to define expected utilities for external actions and computations, compare these and, consequently, by executing the ideal control algorithm, perform either a move action or compute a path to get to a hole. We restrict our attention to complete computations, i.e. after having carried out a computation, there is immediately an external action to be executed. The utility of an external action is simply the inverse of the distance between the agent and the closest hole after performing that action; the utility of a computation is then the utility of the action that results from the computation. The net value of a computation is then the difference between the utility of the computation and the utility of the default external action. With these definitions, the agent can at any moment in time determine if it is best to move or to plan a path.

But, corresponding to the discussion of the theory of discrete deliberation scheduling, until now it is assumed that performing a computation in the TILEWORLD, i.e. path planning, does not have a time cost associated with it. As in applying continuous deliberation scheduling in the TILEWORLD, the real-time aspect of the world is that holes appear and disappear throughout the existence of the world. We can easily encode the cost of path planning by introducing a new parameter, namely one that

represents the number of time steps it takes to construct a path. Now it is straightforward to separate the *benefit* of planning (the difference between the utility of the revised action and the utility of the default action) from the *cost* of planning (the time it takes to plan). The value of a computation is then its benefit minus its cost. Alternatively, we can represent the value of a computation as the *intrinsic*, or time-independent, utility of the action that results from the computation, minus the planning cost. Carrying out these calculations of utilities and expected values is easy in the TILEWORLD, since they only involve calculating distances between cells on the grid.

But although calculating utilities is easy, since they are based on simple distances in the grid, the agent still needs to perform some computation in order to find out the utility of its actions. For example, utility depends on the distance to the closest hole, and the agent needs to find out what the closest hole is. Although this exploration might be trivial in the TILEWORLD, it is a computation nevertheless. It is in the interest of the agent to have *estimates* of these utilities, collected from previous computations. In the TILEWORLD, these estimates might, for example, result from distributions that indicate what the distance is to a closest hole from the agent's current location. These distributions can be easily generated for the TILEWORLD. As the agent is then able to revise these estimates by means of exploration, it is obvious that the agent can now estimate utilities by performing computations. Thus we have shown how to construct an estimated utility model for the TILEWORLD using discrete deliberation scheduling.

3.3 Bounded optimality

Russell (1997) gives four possible formal definitions of rational action:

- *Perfect rationality*
  The system always maximises its expected utility, given what it knows about its environment.
- *Calculative rationality*
  The system maximises its expected utility, based on the state of the environment before deliberation.
- *Meta-level rationality*
  The system optimises over object-level computations for selecting actions.
- *Bounded optimality*
  The system behaves as well as possible given its computational resources.

The key notion in bounded optimality is the move from optimisation over actions or computations to optimisation over programs. Russell argues that too much emphasis in AI has been given to techniques that will select the correct action *in principle*, as opposed to techniques that will be capable of selecting the correct action *in practice*. He suggests that bounded optimal agents – those that select the best action possible, given their computational resources – are therefore a more appropriate goal for AI research.

Let an *abstract agent* be defined as a mapping from percept sequences to actions and let a *physical agent* consist of an architecture (by which Russell means an actual computational device) and a program. An architecture is responsible for interfacing between environment and program and for running the program. A program implements the abstract agent and is constrained by the environment and the architecture it is run by. Based on this intuition, Russell *et al*. (1993) define bounded optimality as follows:

Definition 5. *An agent is* bounded optimal *if its program is a solution to a constraint optimisation problem presented by its architecture and the task environment*.

The idea behind bounded optimality is to formalise on the one hand an abstract agent and on the other hand a physical agent (a "real" agent). We then show that a perfect rational agent is an optimal abstract agent and a bounded optimal agent is an optimal physical agent.

We first formalise an abstract agent as a mapping from percept sequences to actions. Besides formalising this mapping in the model, we want to explicitly represent time in the model of an abstract

agent. The primitives we thus need to specify an agent are drawn from a set of time-points **T**, actions **A** and perceptions **O**. The set **T** is a totally ordered by a relation $<$ with a unique least element. We model percept sequences as percept *histories*, i.e. a complete sequence of percepts indexed by time. A history *prefix* is a projection of a history, thus a partial sequence, until a certain time. For reasons of completeness, for both actions and perceptions, we define sets of histories and history prefixes:

$\mathbf{O}^T = \{OT: T \rightarrow O\}$ denotes the set of percept histories,

$\mathbf{O}^t = \{O^t: t \in T, O^T \in \mathbf{O}^T\}$ denotes the set of percept history prefixes,

$\mathbf{A}^T = \{AT: T \rightarrow A\}$ denotes the set of action histories, and

$\mathbf{A}^t = \{A^t: t \in T, A^T \in \mathbf{A}^T\}$ denotes the set of action history prefixes.

Then an abstract agent is a mapping from a set of percept history prefixes to a set of possible actions: an abstract agent receives at a certain time a percept history prefix and generates an action history based on this. Hence

Definition 6. *An* agent function *has the signature* $f: \mathbf{O}^t \rightarrow A$, *where* $\mathbf{A}^T(t) = f(O^t)$.

This function says that an agent maps partial percept sequences to some action; if we apply this function to the set of all percept prefixes, this generates the set of all action histories. But this function does not reflect the fact that an agent is situated in an environment: let $\mathbf{X}^T = \{X^T: T \rightarrow X\}$ be the set of environment state histories. The model represents the fact that the agent might not have complete access to its environment by a perceptual filtering function f_p that determines the perceptions of the agent. A transition function f_e represents the effects of the agent's actions; it specifies the next state given the current state and the agent's action. Then we can define an environment as follows:

Definition 7. An environment E is a set of states X with initial state X_0 and functions f_p and f_e, such that

$$\begin{aligned} X^T(0) &= X_0, \\ X^T(t+1) &= f_e(A^T(t), X^T(t)) \text{ and} \\ O^T(t) &= f_p(X^T(t)). \end{aligned}$$

Notice that the agent's environment can, to some extent, be inaccessible but is assumed to be deterministic. In the model, *effects*(f, E) denotes the state history generated by agent function f in environment E; and $[E, A^T]$ denotes the state history from applying action history prefix in the initial state of environment E. We use this notation later in the model.

Now that we have defined an abstract agent – the agent function – we continue with the definition of an agent program. An agent program l is an implemented agent function on an architecture M. With M, we define a programming language $\mathcal{L}_M$ and $l \in \mathcal{L}_M$. An agent program receives a percept as its input and has an internal state. In order to formalise this internal state, let

$\mathbf{I}^T = \{I^T: T \rightarrow I\}$ be the set of internal state histories and

$\mathbf{I}^t = \{I^t: t \in T, I^T \in \mathbf{I}^T\}$ the set of internal state history prefixes.

Then

Definition 8. *An architecture M is a fixed interpreter for an agent program that runs the program for a single time step*, *updating its internal state and generating an action*:

$$M: \mathcal{L}_M \times \mathbf{I} \times \mathbf{O} \rightarrow \mathbf{I} \times \mathbf{A},$$

where $\langle I^T(t+1), A^T(t)\rangle = M(l, I^T(t), O^T(t))$.

The signature of this function shows that an architecture takes an agent program l (defined in the programming language $\mathcal{L}_M$), a percept and an internal state as its input and on the basis of those generates an internal state and an action.

Now we have both defined the abstract agent (the agent function) and the "real" agent (the agent program) and are ready to relate the two as follows. We implement the agent function $f = Agent(l, M)$ by an agent program l on architecture M. The basic idea then is that agent function f is constructed by specifying the action sequences produced by an agent program l on architecture M for all possible percept sequences. Thus an agent function f can be defined as follows:

Definition 9. *An agent function $f(O^t)$ is an action history $A^T(t)$ such that the following holds for any environment $E = (\mathbf{X}, f_e, f_p)$*

$$\langle \mathrm{I}^T(t+1), A^T(t)\rangle = M(l, I^T(t), O^T(t)) \quad (1)$$

$$O^T(t) = f_p(X^T(t)) \quad (2)$$

$$X^T(t+1) = f_e(A^T(t), X^T(t)) \quad (3)$$

$$X^T(0) = X_0 \quad (4)$$

$$I^T(0) = I_0 \quad (5)$$

In this definition, equation (1) comes from the definition of an architecture; equations (2), (3) and (4) come directly from the definition of an environment; and equation (5) initialises the internal state of the agent. Note that in definition 9 we formalise an agent function and not a real agent, which, in other words, means that we define the set of all possible agents, implementable or not implementable.

It is important to note that not every agent function f maps to an agent program $l \in \mathcal{L}_M$. This is because some agent programs cannot be implemented on a particular architecture (they may require more memory than is available on the architecture, for example). This leads to an important observation: the set of agent programs is a subset of the set of agent functions. If an architecture M and corresponding language $\mathcal{L}_M$ are given, we can even constrain the set of agent programs: the remaining set of agent programs are called *feasible* agent programs. Again, the set of feasible agent programs is a subset of the set of agent programs. A formal notion of feasibility is necessary to denote all implementable agent functions on a given architecture M and language $\mathcal{L}_M$:

$$Feasible(M) = \{f : \exists l \in \mathcal{L}_M, f = Agent(l, M)\}.$$

The model developed thus far does not enable us to measure the agent's performance. To be able to evaluate the agent's performance, a utility function $U : X^T \rightarrow \mathbb{R}$ is introduced, which maps environment state histories to utilities. The combination of an environment and a utility function is called a *task environment* – the utility function is thus external to the agent and environment. The value of an agent function f in environment E is the utility of its state history:

$$V(f, E) = U(effects(f, E)).$$

Similarly, the value of program l executed in architecture M is based on the utility of agent function f implemented by l:

$$V(l, M, E) = V(Agent(l, M), E) = U(effects(Agent(l, M), E)).$$

If there is a probability distribution defined over a set of environments **E**, one can easily adapt the above definitions to capture expected values.

We are now ready to define a perfectly rational agent and a bounded optimal agent. A perfectly rational agent selects the action that maximises its expected utility, given the percepts so far. In the model, this corresponds to an agent function that maximises $V(f, \mathbf{E})$ over all possible agent functions. However, the problem with perfect rationality is that the optimisation over the agent functions is unconstrained. Instead, we have to account for a machine-dependent kind of rationality that optimises constraints on programs. A bounded optimal agent thus maximises V over the set of agent functions $Feasible(M)$ that are implementable. Then we can define for a set of environments **E**:

- a *perfectly rational* agent: $f_{opt} = \arg\max_f V(f, \mathbf{E})$ and
- a *bounded optimal* agent: $l_{opt} = \arg\max_{l \in \mathcal{L}_M} V(l, M, \mathbf{E})$.

The most important difference between a perfectly rational agent and a bounded optimal agent is the fact that the former is unconstrained, while the latter is constrained by its architecture.

Example. Consider the TILEWORLD scenario as used above. As we here want to implement a bounded optimal agent that operates in the TILEWORLD, we first define an abstract agent for the TILEWORLD. We then constrain this agent by acknowledging the real-time characteristics of the TILEWORLD and the bounded resources of the agent.

Let us first identify the sets of observations **O** and actions **A**. An observation denotes an observational action: the agent acquires knowledge about the location of holes in the world. An action is to move up, down, left or right, thus $\mathbf{A} = \{Up, Down, Left, Right\}$. Combining the set of perceptions and actions with a set of timepoints **T**, results in a set of percept histories $\mathbf{O}^T$ and a set of action histories $\mathbf{A}^T$, and relevant histories as shown above. Now an abstract agent, according to definition 6, is a mapping from observation history prefixes to actions. It is indeed intuitive that when the agent observes some hole at some location in the world, it bases the actions to perform to reach the hole on that observation. An environment state is here a configuration of the TILEWORLD, i.e. a description of the world containing the location of the agent, holes and other objects in the world. It is obvious how we can model the agent's incomplete knowledge of the world using the perceptual filter function f_e, which takes as input an environment state and outputs observations to the agent. The world changes after the agent acts, and the transition function f_a defines how it changes. Since we do not assume a static environment – holes can appear and disappear while the agent moves – here, this characteristic is captured by f_a as well. With definition 7 we then define what an environment E encompasses. The agent receives its utility from the environment; here, when the agent ends up on a location in the world with a hole, it receives a reward. Thus an agent's utility is defined over environment state histories as generated by that agent. Then a *perfectly rational* agent is an abstract agent with maximum utility.

But because of the constraints the environment puts on the agent, not all of these abstract agents are either "useful" or implementable in practice. Although trivial in nature, an agent in the TILEWORLD operates based on the amount of fuel it has. An abstract agent does not take this into account. We introduce an architecture, with corresponding language, on which an abstract agent must be implemented as an agent program. Such an agent program does take its bounded resources into account. In the TILEWORLD, we can define the architecture and language relatively easily, since precise metrics for computing the fuel level are available. Therefore we identify the set of *feasible* agent programs as those programs that maintain an acceptable fuel level. It is clear that this set is a subset of the set of abstract agents, as explained in the theory above. An agent is then executed on the architecture, where execution simply boils down to a mapping between the observations made and the best action as computed by the program. Here, an observation is a TILEWORLD configuration; this observation corresponds to a possible observation (otherwise it could not have been observed); and this possible observation has been mapped to a best action; this action is consequently executed.

Russell *et al*. (1993) recognise that the computation and specification of these bounded optimal agents can still be very hard, and this is already apparent in this simple TILEWORLD application domain. The approach to achieving bounded optimality is then also very different from the continuous and discrete deliberation scheduling methods. Russell *et al*. (1993) show that, with additional assumptions, these computation and specification problems can be tackled. Because it is not the main intention of this paper to show how the methods behave computationally, we do not discuss these assumptions and alternative methods here; they can be found in Russell *et al*. (1993).

3.4 The belief-desire-intention model

One popular approach to the design of autonomous agents that emerged in the late 1980s is the *belief-desire-intention* (BDI) model (Bratman *et al*., 1988; Georgeff & Lansky, 1987). The BDI model gets its name from the fact that it recognises the primacy of beliefs, desires, and intentions in rational action. Intuitively, an agent's *beliefs* correspond to information the agent has about the world. These beliefs may be incomplete or incorrect. An agent's *desires* are states of affairs that the agent would, in an ideal world, wish to bring about. Finally, an agent's *intentions* represent desires that it has

committed to achieving. The idea is that an agent will not be able to deliberate indefinitely over which states of affairs to bring about; ultimately, it must fix upon some subset of its desires and commit to achieving them. These chosen desires are *intentions*. The BDI methodology enables an agent to constrain its reasoning by clearly separating the process of choosing which intentions to achieve and the process of deciding how to achieve an intention.

A major issue in the design of BDI agents is that of when to *reconsider* intentions (Kinny & Georgeff, 1991; Wooldridge & Parsons, 1998). An agent therefore needs to reason about its intentions from time to time, and change its current intentions by dropping them and adopting new ones. However, intention reconsideration is a computationally costly process, and is a kind of meta-level reasoning. It is therefore necessary to fix upon an *intention reconsideration strategy* that makes optimal use of the available computational resources. Kinny and Georgeff (1991) conducted research into different intention reconsideration strategies. The results of their experimental study show that *dynamic* environments – environments in which the rate of world change is high – favour *cautious* intention reconsideration strategies, i.e. strategies which frequently stop to reconsider intentions. The intuition behind this is that such agents do not waste effort attempting to achieve intentions that are no longer viable, and are able to exploit new opportunities as they arise. *Static* environments – in which the rate of world change is low – tend to favour *bold* reconsideration strategies, which only infrequently stop to reconsider intentions. The results of this study are shown in Figures 2(a) – the result plots for a bold agent – and 2(b) – the result plots for a cautious agent. Parameter p indicates the cost of planning. The issue of how an agent should commit to its intentions is essentially one of balancing *deliberative reasoning* (the process of deciding what to do) and *means-ends reasoning* (the process of deciding how to do it).

Intention reconsideration has been modelled on a *conceptual* level (Wooldridge & Parsons, 1998), and only recently research has been undertaken to actually *implement* the intention reconsideration process (Schut & Wooldridge, 2000). We propose investigation on using the models surveyed in this paper, to serve as an implementation for intention reconsideration. Thus the BDI model must not be seen as another time-dependent planning model, but rather as a model in which it is useful to incorporate time-dependent planning.

3.5 Markov decision processes

We briefly touch upon Markov Decision Processes (MDPs) here, since these processes are widely used to model general decision problems (Boutilier *et al*., 1999). The MDP model is another framework in

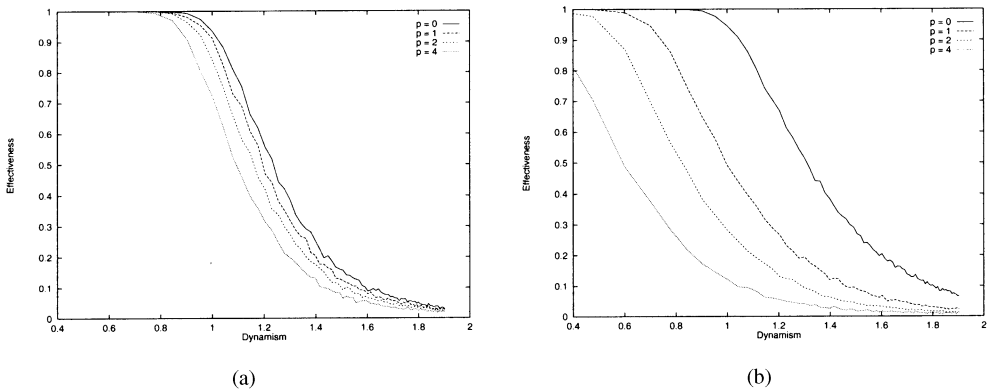

Figure 2 Experimental results on intention reconsideration strategies. The dynamism of the environment, representing the action ratio between the agent and the environment, is varied, and the agent's effectiveness is measured. The cost of planning is represented by p. In (a) the results are shown for a *bold* agent – an agent that executes its complete plan before reconsideration; in (b) the results are shown for a *cautious* agent – an agent that reconsiders at every possible moment. Two observations can be made directly: (1) the planning cost influences the agent's effectiveness: as planning cost increases, effectiveness decreases; and (2) the cautious agent is much more affected by an increasing planning cost than the bold agent is

which the models discussed can be placed to create a more general decision-theoretical planning structure.

An MDP can be understood as a very general decision-theoretic planning approach to agent design; it is basically a system that at any point in time can be in one of a number of distinct states and in which the system's state changes over time resulting from actions. The main components in the MDP model are: a state space S, in which all possible world states are contained; a set of actions A, which contains all actions that can be performed by the agent; a set of observations Ω, a set of "messages" sent to the agent after an action is performed; a value function $V: S \times A \rightarrow \mathbb{R}$, which maps state histories into utilities; and a state transition function $\tau: S \times A \rightarrow \Pi(S)$. Classic AI planning models can be represented as an MDP. A *policy* $\pi: S_t \rightarrow A$, where $S_t \in S$ denotes the possible states at time t, is a mapping from states per time point to actions. In an optimal policy, these actions are optimal, such that the value of the state history generated by that policy is maximal. Assuming that appropriate descriptions of states, actions and value function are available, *dynamic programming* gives algorithms that find optimal policies (Bellman, 1957). The standard dynamic programming algorithms are based on backwards induction; value iteration and policy iteration are the best-known algorithms for solving MDPs. The value iteration algorithm is shown in Figure 3. The algorithm computes the policy value function $Q: S \times A \rightarrow \mathbb{R}$, based on the value of state s and action a, i.e. $V(s, a)$, and the value of the future policy. The future policy value is the expected value of the successor states s', reachable by executing action a, of state s, discounted by a factor γ (where $0 < \gamma < 1$). The probability that performing action a in state s results in state s' is denoted by $p(s' \mid s, a)$ and this probability can be computed using $\tau(s, a)$; the value of a state s is the maximum Q value of all actions in s (computed on line 8 in Figure 3). The Q function has some maximum, denoted by Q^*, and the algorithm stops when it achieves this maximum (or approximates it sufficiently). A major problem in the MDP model is the complexity that is involved with computing optimal policies, which is typically intractable. Therefore appropriate forms of representing the components of an MDP are required and specific knowledge of the application domain is used to speed up the computations. As such, the time-dependent planning models discussed here can be used for that purpose. In this section, we discuss how the planning models are related to the MDP model.

Comparing continuous deliberation scheduling with the MDP model is not immediately evident, since the former originates in search algorithms and the latter in decision theory. But since the models both compute optimal policies, or schedules, off-line, i.e. before execution, some cautious comparison is possible. Whereas the MDP model considers individual actions to reach goals, continuous deliberation scheduling is concerned with decision procedures to achieve some level of optimisation. Such a decision procedure can be seen as a complex action with appropriate properties, i.e. monotonically improving quality over time and diminishing returns, defined by its performance profile.

```
Algorithm:  Value iteration
1.
2.  Arbitrary initialisation of v on S
3.  repeat
4.       for s ∈ S
5.            for a ∈ A
6.                 Q(s,a) ← V(s,a) + γ · Σ_{s'∈S} p(s'|s,a) · v(s');
7.            end-for
8.            v(s) ← max_a Q(s,a);
9.       end-for
10. until Q converges to Q*;
11. return Q*
```

Figure 3 The value interation algorithm for constructing optimal MDP policies

Then we can replace the set of MDP actions with a set of decision procedures and base the MDP utilities on the performance profiles of these procedures. This requires a proper adjustment of the MDP model in order to achieve an efficient representation of the problem. Assuming utilities for actions can be accurately derived from performance profiles, dynamic programming algorithms can be used to compute optimal schedules. However, integration of the two models means that a discrete time scale is used, whereas the time scale in continuous deliberation is, of course, continuous.

Discrete deliberation scheduling is clearly decision-theoretically founded, and therefore it is not hard to relate the model to the MDP model. Using the typical notions of states, actions and utilities, the models are identical up to some level. To our best knowledge, no formal investigation has been undertaken to represent discrete deliberation scheduling as an MDP. But the intuition behind an integration of the two models is straightforward: replacing the set of (external) actions in a standard MDP by the discrete deliberation scheduling action set – a default external action and the set of computational actions – suffices to create an initial MDP representation of discrete deliberation scheduling. We then use the concept of utilities as explained in Section 3.2, and a dynamic programming algorithm to compute optimal policies.

The approach that the MDP model takes towards decision-making is most similar to the bounded optimality model. The most distinctive difference between the two is that bounded optimality explicitly adopts a methodology to deal with computational resources, and a basic MDP does not. The abstract agent in the bounded optimality model can be formalised as an MDP, in which we can identify sets of percepts, actions and environment states. The optimal abstract agent is then one that optimises the utility of the state history it generates, which is equivalent to the definition of an optimal policy in an MDP. But the notion of bounded optimality cannot be directly represented in terms of an MDP, because this notion prescribes the manipulation of computational resources, and this kind of meta-level reasoning is not present in an MDP. Whereas the issue of meta-level reasoning has been applied to MDPs in order to reduce MDP complexity (Boutilier *et al*., 1999), none of these approaches have looked into the explicit allocation of the agent's resources. In MDPs, the construction of optimal policies happens before policy execution (i.e. off-line). However, in the bounded optimality model a policy is dynamically constructed while executing a feasible agent program (i.e. on-line). As mentioned above, the computation and specification of bounded optimal agents is still very hard to perform for real-world situations. For this matter, the bounded optimality model could take the example of the MDP model and focus on the *representations* of states, actions and values, because these are not explored in the bounded optimality model.

3.6 Resource-bounded control of reasoning

In this section we discuss the similarities and differences between the models, and summarise the relative advantages and disadvantages of each framework. We denote the continuous deliberation scheduling procedure by CDS; discrete deliberation scheduling by DDS; and bounded optimality by BO.

The most obvious similarity between the models is that they are based on some notion of agency (although this is not mentioned explicitly in the discussion of CDS). If we consider an agent as an entity situated in an environment that generates actions based on received percepts (Russell & Norvig, 1995), all three models adopt this entity as the decision-maker. This similarity is obvious, because the models are basically decision-theoretic and, as pointed out in Section 2.1, the process of deciding is essentially an interaction, in terms of percepts and actions, between a decision-maker and its environment.

It is obvious as well that all three models are able to control the amount of necessary reasoning before making a decision. Although it is less obvious, the mechanism that accomplishes this is meta-reasoning. For CDS and DDS it is clear that they use meta-reasoning to control reasoning; the meta-reasoning mechanism in BO is effectively the procedure that constructs sequences of decision procedures. This procedure is similar to the CDS procedure.

The models agree that the problem we face when designing a situated resource-bounded agent can conceptually be split into two subproblems: time pressure is a property of the environment and task the agent is to perform, while resource-boundedness is a property of the agent. The property of time pressure corresponds with the environment characteristic of dynamism (mentioned in Section 2.2). The models agree, explicitly or implicitly, on the other environment characteristics: environments are completely accessible (not for BO – see the next section) and deterministic.

All three models adopt utility theory as a means of evaluating the performance of the agent. This illustrates the key point in the design of situated resource-bounded agents: the model must enable the designer to represent the effectiveness of the agent. The models disagree, however, on whether the utility function is a property of the agent, the environment or neither.

The models all recognise that deadlines are the most important concept resulting from incorporating time into decisions.[4] All three models can deal with uncertain deadlines, i.e. deadlines that are stochastic rather than fixed. But the way in which the models deal with deadlines is still static: it is only possible to schedule deliberation if a set of pending events is known. In this way, the behaviour is completely deliberative and it is not possible for the agent to exhibit "reactive behaviour" (Brooks, 1991).

The decision processes in the models are based on compilation rather than run time. This point is closely related to the similarity mentioned in the last paragraph – that agents are not able to react to their environment. Compilation means that all information must be available beforehand and a consequence of this is that during run time it is not possible to make use of newly available information. In this way it is difficult to use the models if the agents are in continuous interaction with their environment. However, continuous interaction with an environment is a generally accepted characteristic of agents in general (Wooldridge & Jennings, 1995).

Finally, the models approach the decision process at a rather high level. Even though the models are applied in real-world situations, the formalisation of the environment and agent are kept high-level, meaning that the focus was on the application rather than the model. A possible explanation for this might be that the models were developed with the application as their main goal rather than the model itself.

A key difference between the models is the way in which the utility of actions is discounted over time. In CDS, actions are formulated as anytime algorithms. A property of an anytime algorithm is that the quality of its solution improves as time progresses, which means, in a broader context, that the utility of an action increases as time progresses. In BO, actions are formulated as anytime algorithms as well, which also means that the utility of an action increases as time progresses. In DDS, however, it is not specified whether an action's utility increases or decreases over time, but it is mentioned that the longer one waits until performing an action, so its utility decreases. But the model does not guarantee that the utility of the default action always increases over time either.

We now weigh up the relative advantages and disadvantages of each model. The advantages of CDS are:

- The model contains a well-defined procedure for execution. Assuming that the decision procedures (anytime algorithms plus their performance profiles) are available, it is straightforward to implement CDS.
- There has been further work done on metrics of performance measurement. Currently, performance profiles only indicate the improvement in accuracy of the solution, but Zilberstein (1996) has made a further categorisation of measuring the performance of an anytime algorithm: certainty (degree of certainty that the result is correct); accuracy (how close the approximate result is to the exact answer); and specificity (metric of the level of the result).
- The procedure is shown to be faster than conventional search algorithms, such as A* (Russell & Norvig, 1995). The CDS procedure has been applied in game-playing programs.

[4] In CDS deadlines are not explicitly mentioned, but conditions themselves can be viewed as deadlines.

The disadvantages of CDS are:

- Because the footing of the procedure is so strongly in search algorithms, the model is only limitedly applicable to more general decision problems. Since the development of anytime algorithms in the late 1980s, they have not been used in a more general context.
- The procedure is less suitable for reasoning. It is not clear how to incorporate the procedure in more sophisticated, intelligent, reasoning systems. The emphasis in further research on CDS seems to have been on how and when to use anytime algorithms rather than the procedure itself.
- The model puts strong requirements on the properties of the performance profiles of anytime algorithms. For example, if the slopes of the performance profiles are not decreasing over time, it cannot be guaranteed that the procedure returns the optimal deliberation schedule.
- The procedure depends very much on the environment. First, much information is needed from the environment before executing the procedure; second, the procedure is very sensitive to its environment in the sense that if something minor changes, it is not flexible enough to react to that.

With respect to DDS, the advantages are:

- Although the theory was initially applied to search algorithms, it is moving to a broader application area. An example of this is the BO model, which is partially based on DDS and is applied to a broader area of problems.
- The model delivers an expressive language, that could be used for other areas of research. The model itself does not emphasise the language, but development of it is an interesting issue for further work.
- Similar to CDS, when applied in game-playing programs, the algorithm is shown to be faster that conventional search algorithms. The CDS and DDS models have not been compared to each other.

The disadvantages of DDS are:

- The main application area of the model is search algorithms. Although the model is moving to a broader application area, there is still much further work.
- The model avoids long-term reasoning, because it adopts the single-step assumption. This means that the model works on the basis of short-term goals.

The advantages of BO are:

- Unlike the other two models, BO does not assume complete accessibility of the environment: the function f_p reduces the available information to a subset of all accessible information in the environment.
- The model is applicable to a wider range of problems; a good example of this is the mail-sorter application.
- When a BO problem is modelled, the BO theory delivers an executable model. This brings the BO model much closer to real-world applications.

The disadvantages of BO are:

- The model has only been applied in episodic environments.
- The basic intuition behind the BO model is that instead of working bottom-up in program design, one should work top-down, e.g. assume all possible sequences of memory configuration in a computer and search for the sequence that solves the problem you have. The main problem with this approach is that one loses control over the program: one ends up with a program that works, but one does not know how it works. In this way, one surpasses one of the basic aims of AI, i.e. to control intelligence in order to manipulate it. Of course, this point can be made about decision theory in general – to know the definition of an optimal agent does not tell us how to implement it. Further discussion about this point is more philosophical than technical and therefore does not fall within the scope of this paper, but certainly needs more attention in further work.

4 Conclusions

In this paper we discussed how agents control their own reasoning, because their resources are bounded. We began by discussing the foundation of theoretical decision-making: decision theory. We concluded this introduction by noting that, in practice, humans do not live up to the ideals of decision theory. This observation is the foundation of bounded rationality. We cannot call this a theory, because in the literature there is currently no agreement yet on a theory of bounded rationality. Bounded rationality is a concept concerned with the limitations of an agent described above and specifically with resource-boundedness. It is closely related to the concept of meta-reasoning. A meta-reasoning agent is conscious of the decisions it makes, i.e. it can control its own reasoning regarding its own decisions. Although there is no satisfying theory of bounded rationality, the concept has been applied to planning. One approach to applying bounded rationality is time-dependent planning, which we discussed in this paper. This kind of planning enables an agent to make optimal use of its available time. We discussed three time-dependent planning frameworks: continuous deliberation scheduling, discrete deliberation scheduling, and bounded optimality. Continuous deliberation scheduling is rooted in search algorithms. Discrete deliberation scheduling is a more decision-theoretic approach. The difference between the two approaches becomes clear if one considers discrete deliberation scheduling as a way of constructing an optimal anytime algorithm. Finally, bounded optimality marries the concepts of continuous and discrete deliberation scheduling by letting agents select the best action possible, given their computational resources, in terms of a more general system.

Much further work can be done on this subject. The main emphasis in this work should be on the development of a general theory on decision-making under bounded resources. This is a high-level and long-term goal, but a first step towards it could be the development of a conceptual framework for situated resource-bounded agents.

References

Allen, JF, Hendler, J and Tate, A, (eds.), 1990, *Readings in Planning* Morgan Kaufmann Publishers.

Bellman, R, 1957, *Dynamic Programming* Princeton University Press.

Boddy, M and Dean, T, 1989, "Solving time-dependent planning problems" *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (IJCAI-89) 979–984.

Boddy, M and Dean, T, 1994, "Decision-theoretic deliberation scheduling for problem solving in time-constrained environments" *Artificial Intelligence*, **67**(2) 245–285.

Boutilier, C, Dean, T and Hanks, S, 1999, "Decision-theoretic planning: structural assumptions and computational leverage" *Journal of AI Research* **11** 1–94.

Bratman, ME, Israel, DJ and Pollack, ME, 1988, "Plans and resource-bounded practical reasoning" *Computational Intelligence* **4** 349–355.

Brooks, RA, 1991, "Intelligence without representation" *Artificial Intelligence* **47** 139–159.

Brooks, RA, 1999, *Cambrian Intelligence* MIT Press.

Buchanan, BG and Shortcliff, EH, 1984, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project* Addison-Wesley.

Bylander, T, 1994, "The computational complexity of propositional STRIPS planning" *Artificial Intelligence* **69**(1–2) 165–204.

Chapman, D, 1987, "Planning for conjunctive goals" *Artificial Intelligence* **32** 333–378.

Davis, R, 1982, "Applications of meta level knowledge to the construction, maintenance, and use of large knowledge bases" in R Davis and DB Lenat (eds.) *Knowledge-based Systems in Artificial Intelligence* McGraw-Hill.

Dennett, DC, 1987, *The Intentional Stance* MIT Press.

Etzioni, O, 1989, "Tractable decision-analytic control" in RJ Brachman, HJ Levesque and Raymond Reiter (eds.) *KR'89: Principles of Knowledge Representation and Reasoning* Morgan Kaufmann.

Genesereth, MR and Smith, D, 1981, "Meta-level architecture" Memo HPP-81–6, Computer Science Department, Stanford University, Stanford, California.

Georgeff, MP and Lansky, AL, 1987, "Reactive reasoning and planning" *Proceedings of the Sixth National Conference on Artificial Intelligence* (AAAI-87) 677–682.

Gleitmann, H, 1991, *Psychology* Norton And Company, Inc.

Good, IJ, 1971, "Twenty-seven principles of rationality" in VP Godambe and DA Sprott (eds.) *Foundations of Statistical Inference* Holt Rinehart Wilson.

Kinny, D and Georgeff, M, 1991, "Commitment and effectiveness of situated agents" *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (IJCAI-91) 82–88.

Lawler, EL, 1985, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* Wiley.

Luce, RD and Raiffa, H, 1957, *Games and Decisions* John Wiley & Sons.

Mueller, JP, 1997, *The Design of Intelligent Agents* (LNAI Volume 1177) Springer-Verlag.

Newell, A, Rosenbloom, PJ and Laird, JE, 1989, "Symbolic arcthitectures for cognition" in MI Posner (ed.), *Foundations of Cognitive Science* MIT Press.

Pollack, ME, Joslin, D, Nunes, A, Ur, S and Ephrati, E, 1994, "Experimental investigation of an agent commitment strategy" Technical Report 94–31, Department of Computer Science, University of Pittsburgh.

Pollack, ME and Ringuette, M, 1990, "Introducing the Tileworld: experimentally evaluating agent architectures" *Proceedings of the Eighth National Conference on Artificial Intelligence* (AAAI-90) 183–189.

Russell, SJ, 1997, "Rationality and intelligence" *Artificial Intelligence*, **94**(1–2) 57–78.

Russell, S and Norvig, P, 1995, *Artificial Intelligence: A Modern Approach* Prentice-Hall.

Russell, SJ, Subramanian, D and Parr, R, 1993, "Provably bounded optimal agents" *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (IJCAI-93) 338–344.

Russell, SJ and Wefald, E, 1991, *Do the Right Thing: Studies in Limited Rationality* MIT Press.

Schut, MC and Wooldridge, M, 2000, "Intention reconsideration in complex environments" *Proceedings of the Fourth International Conference on Autonomous Agents* (Agents 2000) 209–216.

Simon, HA, 1982a, *Models of Bounded Rationality*, Volume 2 MIT Press.

Simon, HA, 1982b, "A behavioral model of rational choice" in *idem*, *Models of Bounded Rationality*, Volume 2.

Simon, HA, 1982c, "Rational choice and the structure of the environment" in *idem*, *Models of Bounded Rationality*, Volume 2.

Von Neumann, J and Morgenstern, O, 1944, *Theory of Games and Economic Behaviour* Princeton University Press.

Wilson, RA and Keil, F, (eds.), 1999, *MIT Encyclopedia of Cognitive Sciences* MIT Press.

Wooldridge, M and Jennings, NR, 1995, "Intelligent agents: theory and practice" *Knowledge Engineering Review* **10**(2) 115–152.

Wooldridge, M and Parsons, SD, 1999, "Intention reconsideration reconsidered" in JP Müller, MP Singh and AS Rao (eds.), *Intelligent Agents V* (LNAI Volume 1555) Springer-Verlag.

Zilberstein, S, 1996, "Using anytime algorithms in intelligent systems" *AI Magazine*, **17**(3) 73–83.