On the Problem of Comparing Two Models for Rational Decision Making in Autonomous Agents

Simon D. Parsons [†] Gerardo I. Simari [‡]

[†] Department of Computer and Information Science
Brooklyn College, City University of New York
2900 Bedford Avenue - Brooklyn, NY 11210, USA
Tel: ++1 718 951 5532 - Fax: ++1 718 951 4842
parsons@sci.brooklyn.cuny.edu

[‡] Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)*
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur. Av. Alem 1253, (8000) Bahía Blanca, Argentina
Tel: ++54 291 459 5135 - Fax: ++54 291 459 5136
gis@cs.uns.edu.ar

Abstract

The main objective of this work is to establish a basis for comparing BDI agents with the optimality of decision theory models such as POMDPs. It is argued that a direct comparison is not possible due to the intractability of the algorithms used for solving POMDPs, and therefore an approximation must be used. We propose the reduction of the state space, and the combination of sub-solutions as the main approaches towards this goal. Throughout this work, the TILEWORLD testbed is used as a frame of reference for the discussion of the various concepts.

Keywords: Intelligent Agents, BDI Architecture, Decision Theoretic Models, POMDP.

1 Problem Description and Background

The problem of making decisions in the best possible way is a difficult one. There exists a variety of models that have been proposed to build agents, each of which yields a different strategy for the solution of this problem [BIP91, WJ94, Woo99, FG97]. Few, however, can prove that their strategy in fact produces an *optimal* way of choosing actions in order to reach the specified goals. One such family of models of optimality are called *Markov Decision Processes* (MDPs); however, these models suffer from the intractability of the algorithms that are used to obtain optimal strategies.

This paper's main objective is to establish a basis for comparing the performance of *BDI* model of agency (a heuristic approach to decision making) against the optimality of MDPs. This basis is necessary because such comparison cannot be carried out directly; the intractability

*Member of the IICyTI (Instituto de Investigación en Ciencia y Tecnología Informática) of the process of obtaining a solution for an MDP prescribes a meaningful comparison, as we argue below.

This work is organized as follows: in this section, we present an introduction to the TILEWORLD, a testbed which is widely used to evaluate the performance of agent architectures. Sections 1.2 and 1.3 describe both models and a possible implementation for them in the TILEWORLD. Sections 2 and 3 introduce two approaches for using solutions to MDPs in larger worlds. Finally, section 4 summarizes the results obtained and discusses future work.

1.1 The TILEWORLD Domain

The TILEWORLD testbed, introduced by Pollack and Ringuette [PR90] is a grid environment occupied by agents, tiles, holes, and obstacles. The agent's objective is to score as many points as possible by filling up holes, which can be done by pushing the tiles into them. The agent can move in any direction (even diagonally); the only restriction is that the obstacles must be avoided. This environment is *dynamic*, *i.e.*, holes may appear and disappear randomly in accordance to a series of world parameters, which can be varied by the experimenter.

Because this environment is too complex for the desired experiments, the simplified testbed described in [SW00] was adopted. The simplifications to the model are: tiles are omitted, *i.e.*, an agent can score points simply by moving to a hole; agents have perfect, zero-cost knowledge of the state of the world; and agents build correct and complete plans for visiting a single hole (they do not plan tours for visiting more than one hole). This domain, although simplistic, is useful in the evaluation of the effectiveness of situated agents. One of its main advantages is that it can be easily scaled up to provide with difficult and unsolvable problems.

1.2 Sequential Decision Making

Here, we describe the problem of choosing optimal actions in complex *stochastic* environments. In this sort of environments, actions are not considered to have a unique guaranteed effect; now, actions have a *set* of possible effects, each of which has a *probability* of occurrence associated with it.

Markov Decision Processes are useful in modeling sequential decision making. We will assume that agents using this type of model have perfect sensorial capability, *i.e.*, that they always know *exactly* what state they are in, even though there is uncertainty about the effect of their actions. However, in this model, the agent need not retain any information about the history of its past actions in order to make optimal decisions.

The problem is, then, to find the best way to behave given a complete and correct model of the environment (and of course, a goal). The same problem has been addressed in AI as planning problems, but the consideration of stochastic domains forces us to depart from the traditional model and compute solutions in the form of *policies* instead of action sequences (plans). A policy is a complete mapping from states to actions; once a policy is calculated from the transition model and the utility function (see below), it is trivial to decide what to do. The agent's function is represented explicitly by the policy: it describes a simple reflex agent. The problem of calculating an *optimal* policy in an *accessible, stochastic* environment with a known transition model is called a *Markov Decision Problem* (MDP). In decision problems, the *Markov property* holds if the transition probabilities from any given state depend only on the state and not on the previous history. A Markov Decision Process can be defined [Lit96] as a tuple $M = (S, A, T, R, \beta)$, where:

- S is a finite set of *states* of the environment.
- A is a finite set of *actions*.
- $T : S \times A \rightarrow \Pi(S)$ is the *state transition function*. It gives, for each state and action performed by the agent, a probability distribution over states ($T(s, a, s')$ is the probability of ending in state s', given that the agent started in state s and performed action a).
- $R : S \times A \rightarrow I\!R$ is the *reward function*, which gives the expected immediate reward gained by the agent for taking each action in each state.
- $0 < \beta < 1$ is a *discount factor*. This factor is used to represent that a reward obtained in the future is less valuable than an immediate one. Its use ensures that the reward obtained by a policy is not unbounded, and that the algorithm used to calculate the utilities converges.

One of the most popular algorithms that can be used to obtain an optimal policy is *Value Iteration*, which basically calculates the utility of each state using dynamic programming techniques, and then uses these values in the selection of an optimal action for each state. Because actions have no guaranteed effect, the calculation of utilities is not straightforward, but can be done to any degree of accuracy by using an iterative procedure.

The main drawback of algorithms for directly solving MDPs, is their intractability for even moderatately large problems. A naïve approach for finding an optimal policy (one which tries every possible combination) would be $O(|A|^n)$, where n is the number of steps in the decision problem. This would preclude exhaustive search even for small values of $|A|$ and n.

In the dynamic programming approach, the cost of calculating the utility of one state is $O(|A|)$, and therefore the whole computation is $O(n|A||S|)$, or $O(|A||S|)$ if discounting is used. So, in general, each iteration of the Value Iteration Algorithm takes $O(|A||S|)^2$ steps. The number of iterations required to reach an optimal policy can be proved to be bounded above by a polynomial in $|S|$, $|A|$, B, and $1/(1-\beta)$, where B is used to designate the maximum number of bits needed to represent any numerator or denominator of β, or one of the components of T or R. As a lower bound, Value Iteration has a worst case run time that grows faster than $1/(1-\beta)$ [Lit96, Tse90].

As was described above, the MDP models the world by taking into account *every* possible action in *every* possible state. For the simplified TILEWORLD, this means that for a world of size n (that is, an $n \times n$ grid) there is a set of 8 actions, n^2 possible positions for the agent, and 2^{n^2} possible configurations of holes. This last number is obtained by considering that every position in the grid may contain a hole or not. A *state* in this world consists of a pair (P, H), where $P = (i, j), 0 \leq i, j \leq n - 1$, and H represents a given configuration of holes on the grid. Therefore, the total number of states in this case is $n^2 2^{n^2}$; the following table presents the number of states for some values of n:

n	Number of states
3	4,608
4	1,048,576
5	838,860,800
6	2,473,901,162,496
7	$\approx 10^{16}$
8	$\approx 10^{21}$
9	$\approx 10^{26}$
10	$\approx 10^{32}$

These values show that even for a very small TILEWORLD, the MDP model requires an intractable amount of resources (both time and space) in order to compute an optimal policy. The limit for the tractability of direct calculation seems to be at $n = 4$, or $n = 5$ for a computer with sufficient resources.

This doesn't mean that the MDP model cannot be used at all. The "explosion" in the number of states, as we have seen, depends largely on the amount of holes that can be present at a given moment. This insight led to the consideration of ways in which the intractability can be addressed, which will be discussed in sections 2 and 3.

1.3 The BDI Architecture

the Belief-Desire-Intention architecture, commonly abbreviated "BDI", has its roots in the philosophical tradition of understanding *practical reasoning* [BIP91]. This type of reasoning can be described as the process of deciding what actions to perform in order to reach a goal. Practical reasoning involves two important processes: decide *what* goals to try and reach, and *how* to reach them. The first process is known as *deliberation*, and the second as *means-ends* reasoning.

One of the classic problems in the design of practical reasoners is how to obtain a good *balance* among the different aspects involved in the dynamics of intentions. Specifically, it seams clear that an agent must, at some point, drop some of its intentions (because they have been reached, they *cannot* be reached, or the reasons for adopting them are no longer valid). It follows, then, that it is worthwhile for the agent to stop and *reconsider* its intentions; but reconsideration has its costs, both in time and computational resources. This situation presents an interesting tradeoff:

- An agent that doesn't stop to reconsider often enough will continue trying to reach outdated goals.

- An agent that *constantly* reconsiders its intentions might spend too much time deliberating and will therefore not get anything done. In this case, it is possible that the agent will never reach its goals.

This is essentially the dilemma of balancing *proactive* (goal directed) and *reactive* (event directed) behavior. The situation described above was examined by David Kinny and Michael Georgeff, in a number of experiments performed with a BDI architecture called *dMARS*. They investigated how *bold* agents (those that never stop to reconsider) and *cautious* agents (those that constantly reconsider) behave in a variety of different environments. The most important parameter in these experiments was the *rate of change* of the world, γ. Kinny and Georgeff's key results were [KG91]:

- If γ is low (*i.e.*, the environment doesn't change rapidly), then bold agents work well with respect to cautious ones. This is because cautious agents waste their time reconsidering their commitments while bold agents work to reach their goals.

- If γ is high (*i.e.*, the environment changes frequently), then cautious agents tend to perform better than bold ones, because they can recognize when their intentions are no longer valid, and they can take advantage of new opportunities when they arise.

The result is that different types of environments require different types of decision strategies. In static environments, proactive behavior is adequate. In more dynamic environments, the ability to react to changes by modifying intentions becomes more important. Therefore, there is no "best way" to resolve the balance mentioned. Each application must be *tuned* to its environment, and therefore the best balance depends on the application. The best way to resolve this conflict may be to implement an adaptation mechanism, with which agents recognize how often they must stop to reconsider their intentions.

The practical reasoning process can be broken down into a number of basic components; the following are in general part of a BDI model [Woo99]:

- A set of current *beliefs*, which represents the information the agent currently has about its environment.

- A *belief revision* function, which takes a perceptual input and the agent's current beliefs and, based on this, determines the new set of beliefs.

- An *option generation* function, which determines the options available to the agent based on the current beliefs about the environment and its current *intentions.* This function represents the agent's means-ends reasoning–the process of deciding how to achieve intentions. It maps a set of beliefs and a set of intentions into a set of *desires.*

- A set of *current options*, which represents the agent's possible courses of action.

- A *filter* function, which represents the agent's *deliberation* process, and which determines the agent's intentions based on its current beliefs, desires, and intentions.

- A set of current *intentions*, which represents the agent's current focus, *i.e.*, those states to which it is committed to arrive.

- An *action selection* function, which determines an action to perform based on the current intentions.

The desired behavior will have an impact on how each of these components is implemented. Such variety in possible implementations causes this architecture to be considered a *family* of models, rather than a rigid design method.

A BDI agent for the TILEWORLD can be easily implemented. The agent's beliefs consist of its perceptions of the locations of holes in the world. Various parameters dictate how accurate these beliefs are: *accessibility* (how many positions the agent can see from where it is standing), *dynamism* (how many steps the world takes for each step of the agent), *planning cost* (how many steps the agent must spend in order to build a plan), and the agent's *intention reconsideration* strategy. This last parameter is one of the most important because it has a great influence on how efficient the agent will be; if intentions are reconsidered too often or too soon, this will lead to a waste of effort [WP99].

The next component, *desires*, can be seen under this model as possible plans leading to a selected goal. In our case, any series of actions leading from the agent's current position to a hole constitutes a desire. On the other hand, an *intention* is a desire selected in order to reach a goal. The agent will select one such intention, and will use it to fill holes in the best possible way. Intention reconsideration in this domain corresponds to the frequency in which the agent revises its plans: *bold* agents reconsider after each action taken, while *cautious* ones wait until the current plan is completed to build a new one.

The computational costs associated to this model are low: plans can be built in time linear in the size of the world, and there is no off-line cost because all of the processing is done during execution time (unlike the implementation discussed below). However, this does not mean that BDI agents are *always* efective: the burden lies at the agent's *reconsideration strategy*, which can lead the agent to wasted efforts if it is sub-optimal.

2 Reducing State Information

One possible way of keeping the computation of policies within acceptable bounds is to consider a reduced state space. This means that the agent will no longer have *complete* information regarding the current state of the world, *i.e.*, the current state will be *hashed* into one of the states in the reduced space. Of course, this will generally mean that the agent will no longer be able to select the *optimal* action in each step. Nevertheless, if the hashing is a good one, the action taken by the agent will be optimal with respect to the *reduced* state space, albeit sub-optimal in general.

This hashing can be done in a variety of ways. Each proposal is a tradeoff between number of states (*i.e.*, complexity) and optimality. The following sections describe these proposals.

2.1 The Closest Hole

In this proposal, the agent is only aware of the closest hole. Even though it is simple (and therefore uses few resources), this strategy cannot ensure good results because an agent will not be able, for example, to decide between two closest holes as in the following case ($n = 7$):

	1	2	3	4	5	6	7
1	●	●					
2	●	●					
3							
4				**A**			
5							
6						●	
7							

Here, even though (2,2) and (6,6) are both the closest to the agent, (2,2) is the best option because it is close to a group of holes. This behavior is captured by Value Iteration (position (2,2) will have a greater utility than (6,6)), but will be ignored by this approach. In this case, the advantage is that the number of states is only n^4, which is much smaller than the full state space.

2.2 The k Closest Holes

This approach, a generalization of the previous one, keeps track of the k closest holes. The parameter k can be varied in order to trade efficiency against cost: $k = 1$ yields the previous approach, while $k = n^2$ is the general case discussed above. This generalization is meant to deal with the difficulties that arise in the last section; even though it does smooth out the amount of cases where the agent simply does not have enough information to select a good action, cases like the following ($n = 7$, $k = 4$):

	1	2	3	4	5	6	7
1	●	●					●
2	●	●				●	
3							
4				**A**			
5							
6	●	●				●	
7		●					

still leave the agent in a 'blind spot' with respect to the best hole. The agent will regard the four closest holes as equal and will have to make a random choice among them, even though they all have different utilities. Here, the number of states grows to $n^2 \sum_{i=1}^{k} C_i^{n^2} \leq n^2 2^{n^2}$, where C_k^n represents the number of combinations of size k taken from a set of size n.

2.3 Local Density

This proposal is an extension of the previous one; as before, the agent keeps track of the k closest holes. The difference is that, with every hole, additional information is kept regarding the *local density* of the area in which the hole lies. The local density of a given position is a value that indicates how "dense" (with holes) the area around that position is. The agent now has the possibility of taking this information into account in order to avoid cases like the counter-example presented above.

The local density for a given position (i, j) that contains a hole can be calculated as the sum of the *influence* of every other hole in the world, where the influence is a value obtained as the product of the hole's reward and the inverse of the distance that lies between it and (i, j). More formally,

$$D_{i,j} = \sum_{(s,t)\in H,\ (s,t)\neq(i,j)} R'_{s,t} \frac{1}{Dist_{(i,j),(s,t)}}$$

where $D_{i,j}$ is the *density* of position (i, j), H is a set of ordered pairs containing the location of every hole in the grid, R' is the *reward function* on states, and $Dist_{(i,j),(s,t)}$ represents the distance between positions (i, j) and (s, t); because the agent is capable of moving diagonally, this distance is not equivalent to Manhattan distance (it is smaller in the general case). The following example illustrates this concept:

	1	2	3	4	5
1					●
2	●	●			
3		●			
4	●				
5	●				

If we assume a reward of 1 for each hole, then we have $D_{1,5} = 1 + \frac{1}{3} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \approx 2,416$. On the other hand, $D_{5,1} = \frac{1}{3} + \frac{1}{3} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} \approx 1,416$, which represents that the area around hole $(1, 5)$ is more dense with holes than the area around hole $(5, 1)$.

How can this added information be used? One possibility is that the agent can use it to break ties among closest holes (as presented in the counter-examples above). If the information is only used in this way, the following is an example of a case where the approach will suggest a sub-optimal action ($n = 7$, $k = 4$):

	1	2	3	4	5	6	7
1	●	●					
2	●	●					
3							●
4							
5					**A**		
6							
7			●				●

Even though $(7,3)$, $(7,7)$, and $(3,7)$ are closest to the agent, the tie-breaking information will decide only among these holes. However, the optimal action would be to head NW towards the concentration of four holes. Another possible use of the additional information is to take the *k most locally dense* holes. This, in turn, also has disadvantages; for example ($n = 7$, $k = 4$):

	1	2	3	4	5	6	7
1	●	●					
2	●	●					
3							
4							
5					**A**	●	
6							
7							

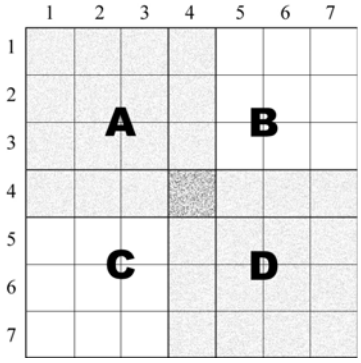
The four holes in the NW corner are the k most dense; nevertheless, $(6,5)$ would be the best option. In this approach, the number of states is the same as in the previous section because the size of the hashed state space has not changed, only the way the hashing is done. The calculation of the density information yields the additional cost. The distance between holes and the reward function on states can be calculated in $O(1)$, so the complete density information for *one* state can be obtained in $O(n^2)$.

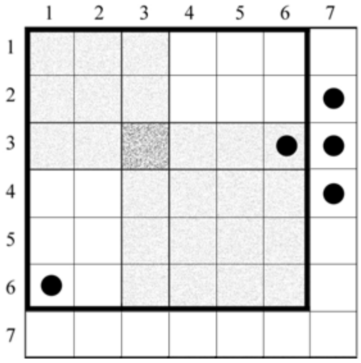
3 Using Optimal Sub-solutions

Another way of addressing the problem of the intractability of policy calculation is by obtaining an optimal policy for the greatest world size that can be solved within a given limit of expended resources, and then using this solution to build policies for greater world sizes. In this section we will describe two approaches for obtaining a policy for a 7 × 7 TILEWORLD by combining optimal policies for 4 × 4 worlds, which is the greatest world size for which a solution can be found with acceptable costs.

3.1 The Moving Window

This approach proposes the combination of four 4 × 4 grids to make up a new 7 × 7 grid, which will be referred to as the *moving window*. This can be done by putting the four grids together so they overlap one row; we will call these sub-grids **A**, **B**, **C**, and **D** (see figure below). This 7 × 7 grid is called a "moving window" because it follows the agent's moves, representing the limit of its perception. The agent occupies the center (*i.e.*, the SE corner of **A**, the SW corner of **B**, the NE corner of **C**, and the NW corner of **D**), and it will only be aware of the holes that fall inside this window. For example, if the agent is in the center of the grid, then it has full accessibility; now, if it moves North one position, it will not be able to see the holes to the far South. The figure below (left) illustrates how the sub-grids are combined to form the moving window: grids **A** and **D** are shaded, while **B** and **C** are white. The figure to the right shows the case in which the agent has moved Northwest from the center of the grid:





The new policy is built using the solved 4×4 sub-grids. For each state, the agent will have four possibly different suggestions (one for each 4×4), given by the policies:

$$\pi_i(s) = \arg\max_a \sum_t T_i(s, a, t) U_i(t)$$

where $i = \mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, or $\mathbf{D}$, T_i is the state transition function for grid i, and U_i is the utility function on states for grid i. The best action is then the one that has the highest expected utility:

$$\pi^*(s) = \pi_i(s)$$

for some i such that $\pi_i(s) = t_i$, where $U_i(t_i) = u_i$ and there is no u_j such that $\pi_j(s) = t_j$, $U_j(t_j) = u_j$, and $u_j > u_i$.

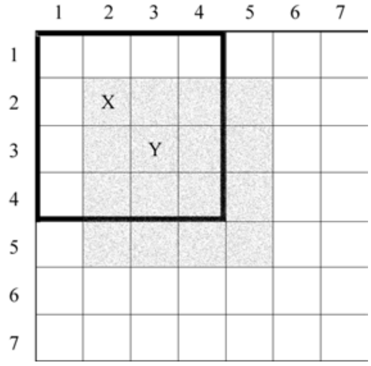
There are cases in which the agent will act sub-optimally under this approach. For example, in the figure above the agent is not in the center, which means that it will not have full accessibility. Indeed, the South row and the East column are now invisible (see figure). Here, grid **B** will suggest action *East*, whereas grid **C** will suggest *Southwest.* The utilities associated to both actions equal; the agent is therefore in a situation in which it cannot choose the best action. Nevertheless, heading East would be the best choice because it brings the agent closer to the three holes that it cannot see.

The computational cost of this approach is not significantly greater than the cost of solving an MDP for a TILEWORLD of size 4×4. Because the policy is built from the policy for a 4×4, the only difference lies in execution time, where the hashing and the selection of the best action among the four suggested take place. The hashing can be done in $O(n^2)$ because each position must be tested to see if it falls inside the moving window; the execution time of both operations are clearly in $O(1)$.

3.2 Limited Accessibility

In the last section we proposed a way in which solutions for a 4×4 grid can be used to build a policy for the 7×7 world. Here, a reduced version of this proposal is introduced, in which a *single* solution is used.

Using a single 4 × 4 solution, a simpler version of the “moving window” can be defined. Because the center for a 4 × 4 grid is not defined, the new version of the window will behave differently. Instead of moving with every step the agent takes, the window will shift position if the agent is moving in a given direction and the window’s border in that direction is two positions ahead of the agent, *and* the border of the window is not being “pushed” beyond the grid’s limit. For example:



The figure describes the following scenario: the agent is in $(2, 2)$, which is marked with an ‘X’; the window’s position is framed by heavy lines. If the agent moves Southeast, the window will shift Southeast along with it, and the agent’s new position will be $(3, 3)$ (marked with a ‘Y’). The shaded area is the window’s new location. On the other hand, if it should move North, the window would not shift, and the agent’s new position would be $(2, 1)$. The agent will then use a single solution to implement a form of *limited accessibility.* The window’s movement is designed to provide the agent with the maximum possible accessibility given the reduced size of the solution being used.

The computational cost of this approach is dominated by the execution time cost of performing the hash and the update of the window’s position according to the rules defined above. The off-line cost is the same as in the last section, *i.e.*, the cost of solving an MDP for a 4 × 4 TILEWORLD.

4 Conclusions

In this work, we established a basis for comparing two models of agency: BDI architectures and Markov Decision Processes. It was shown that a direct comparison of the behavior of such models is not possible because of the intractability of finding solutions for MDPs. Using approximations to solutions for this model is an alternative for such a comparison. We have proposed a variety of models in which approximations were obtained first by *reducing* the state space, and then by using *sub-solutions* to build policies.

Future work involves the implementation of the proposed models in order to empirically evaluate their behavior and compare it to the performance of an agent implemented with the BDI architecture. Such a comparison will be useful for related future work involved in studying the relationship between these models.

References

[BIP91] Michael E. Bratman, David Israel, and Martha Pollack. Plans and resource-bounded practical reasoning. In Robert Cummins and John L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts, 1991.

[FG97] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. *Lecture Notes in Computer Science*, 1193:21–??, 1997.

[KG91] D. N. Kinny and M. P. Georgeff. Commitment and effectiveness of situated agents. In Barbara Grosz and John Mylopoulos, editors, *Proc of the Twelfth International Joint Conference on Artificial Intelligence*, pages 82–88, San Mateo, California, 1991. Morgan Kaufmann.

[Lit96] Michael Lederman Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, Providence, RI, February 1996.

[PR90] Martha Pollack and Marc Ringuette. Introducing the tileworld: experimentally evaluating agent architectures. In Thomas Dietterich and William Swartout, editors, *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, Menlo Park, CA, 1990. American Association for Artificial Intelligence, AAAI Press.

[SW00] Martijn Schut and Michael Wooldridge. Intention reconsideration in complex environments. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 209–216, Barcelona, June 2000. ACM Press.

[Tse90] Paul Tseng. Solving H-horizon, stationary Markov decision problems in time proportional to $\log(H)$. *Operations Research Letters*, 9(5):287–297, 1990.

[WJ94] Mark Wooldridge and Nick Jennings. Agent theories, architectures, and languages: A survey. volume 890, pages 1–32. 1994.

[Woo99] M. Wooldridge. Intelligent Agents. In G. Weiss, editor, *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–78. The MIT Press, Cambridge, Massachussetts, 1999.

[WP99] Mike Wooldridge and Simon Parsons. Intention reconsideration reconsidered. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pages 63–80. Springer-Verlag: Heidelberg, Germany, July 1999.