Knowledge Representation for Learning How to Evaluate Partial Plans

Sławomir Nowaczyk
Lund University
Slawomir.Nowaczyk@cs.lth.se

Abstract

In this paper we present some ideas for knowledge representation formalism suitable for rational agents which learn how to choose the best conditional, partial plan in any given situation. In our architecture, the agent uses an incomplete symbolic inference engine, employing Active Logic, to reason about consequences of performing actions — including information-providing ones. It utilises a simple planner to create conditional partial plans, i.e. ones which do not necessarily lead all the way to the ultimate goal. Finally, a learning module — based on ILP mechanisms — provides, from experience, knowledge on how to choose which of those plans ought to be executed.

We discuss principles which should guide design of knowledge representations in order to best fit the requirements of learning process. Clearly, simply presenting all of agent's knowledge to the ILP algorithm is very inefficient. On the other hand, for many particular applications some very effective representations are known. We compare several approaches, analysing the tradeoff between amount of domain specific knowledge provided and the quality of solutions obtained.

In the experiments presented we used PROGOL for learning, and one of the conclusions of this paper is that some algorithm better suited for the particular problem of evaluating plans could significantly improve the competitiveness of domain-independent solutions.

Introduction

In our research we are interested in creating rational, autonomous agents that are able to survive in a dynamic world. In order to be practical, such agents need to be modelled as having limited computational resources, but they also need to be aware of their own limitations and take them into account (Chong *et al.*, 2002). To facilitate this, we use a formalism known as Active Logic, which characterises reasoning as an ongoing process, instead of focusing on a fixed point of entailment relation.

Due to limited resources and the necessity to stay responsive in a dynamic world, situated agents cannot be expected to create a complete plan for achieving their goals. They need to consciously alternate between reasoning, acting and observing their environment. In order to achieve this, in our approach the agent creates short partial plans and executes them, learning more about its surroundings in the process.

One special case are "information-providing" actions and plans, which allow an agent to greatly simplify subsequent planning process — it no longer needs to take into account the vast number of possible situations which will be inconsistent with newly observed state of the world. Thus, it can proceed further in a more effective way, by devoting its computational resources to more relevant issues.

Our goal is to create an agent able to function in an adversary environment which it can only partially observe and which it only partially understands. In order to succeed in this, the agent must be allowed to experiment in a number of episodes, learning from its mistakes and improving itself.

Active Logic, when used as a main logical formalism for our agent, allows us to do many of those things. It was designed for non-omniscient reasoners and we believe it is a good reasoning technique for versatile agents. Moreover, we require our agent to combine planning, deductive reasoning and inductive learning with time-awareness. We believe that the interactions among those three aspects are crucial for developing truly intelligent systems.

Thus, the agent we are discussing will create several partial plans and reason about usefulness of each one — including what knowledge can be acquired by executing it. Further, it will judge whether it is more beneficial to begin executing one of those plans immediately or rather to continue deliberation. In other words, the agent will be performing on-line planning and interleaving it with plan execution.

Moreover, we expect it to live significantly longer than any single planning episode lasts, so it should generalise every solution it finds. In particular, the agent needs to extract domain-dependent control knowledge and use it when solving subsequent, similar problem instances. To this end we introduce an architecture consisting of three modules, which we believe will allow combining state-of-the-art solutions from several fields of Artificial Intelligence, in order to provide the synergy of features our agent requires to achieve the desired functionality.

In the next section we introduce an example domain on which we present our ideas. In section *Architecture* we describe the organisation of our agent. The three following sections introduce each of agent's functional modules in more detail: *Deductor*, *Actor* and *Learner*. After that, we discuss some of the *Related Work* and finish with some *Conclusions*.

Wumpus

In this paper we will be using a simple game called Wumpus, the well-known testbed for intelligent agents (Russell & Norvig, 2003), to better illustrate our ideas. The game is very simple, easy to understand, and people have absolutely no problems playing it effectively as soon as they learn the rules. For artificial agents, however, this game — and other similar applications, including numerous ones of practical importance — still remain a serious challenge.

The game is played on a simple square board. There are two characters, the player and the Wumpus. The player can, in each turn, move to any neighbouring square. The Wumpus, in the simplest version of the game at least, does not move at all. Position of the monster is not known to the player, he only knows that it hides somewhere on the board. Luckily, Wumpus is a smelly beast, so upon entering a square the player can immediately observe whether the creature is in the vicinity or not. The goal of the game is to find out the location of the monster, by moving throughout the board and observing on which squares does it smell, and to finally shoot the creature with a bow. However, if the player enters the square occupied by the beast, he gets eaten and loses the game.

In order to understand the goal of our research, it can be helpful to imagine the setting somewhat akin to the *General Game Playing Competition*: our agent is given some knowledge about the domain and is supposed to act rationally from the very beginning, while becoming more and more proficient as it gathers more experience.

Agent Architecture

The architecture of our agent consists of three main functional modules. Each of them is responsible for a different part of agent's rationality, but the overall intelligence is only achievable by the interactions of them all.

The *Deductor* module is the one responsible for classical "thinking". It uses logical formalism based on combination of Active Logic and Situation Calculus (as introduced in Nowaczyk, 2006) in order to reason about consequences of agent's current beliefs. Based on domain knowledge and previous observations, it analyses possible actions and predicts what will be the effect of their execution. In particular, it accounts for the fact that some actions may be information-providing ones. We describe Deductor in more details in the following section.

In addition to the typical reasoner functionality, the Deductor module also includes a simple planner, which generates partial, conditional plans applicable in agent's situation. In our current implementation the planner is a trivial one and it simply generates all possible plans of a given length. We are looking into a ways of hooking up a more efficient planner instead, but doing so is not trivial, since domain knowledge in our case is, typically, expressed in a language richer than what classical planners accept.

The second main module, *Actor*, oversees Deductor's reasoning process and evaluates plans the latter has come up with. Taking into account the possible consequences of each of them, it tries to find out which is the most valuable one to execute. At the moment, the Actor relies on incompleteness of the Deductor, and on the fact that the inference is guaranteed to always terminate. Only after this happens, the Actor chooses the best plan. Our ultimate goal, however, is to have it monitor the Deductor and estimate its reasoning progress, so that it can break the deliberation when a particularly interesting plan has been discovered or when it decides that nothing worthwhile is likely to be deduced anymore.

Finally, the *Learner* analyses agent's past experience and induces rules for estimating quality of plans. Results of learning process are used both by the Deductor and by the Actor. In particular, since the plans Deductor reasons about are partial (i.e. they do not — most of the time — lead all the way to the goal) it is very difficult, in general, to estimate whether a particular plan is a step in the right direction or not. Using machine learning techniques is one way in which this could be achieved.

In general, the ultimate goal of this architecture is to allow putting together state-of-the-art solutions from several different areas of Artificial Intelligence. Despite multiple efforts, both ones done in the past and those still in progress, the vast majority of AI research is being done in specialised subfields and — while such research is very important and often greatly successful — it is this author's belief that neither of these subfields *alone* can give us truly intelligent, rational agents.

Deductor

In some sense, the Deductor forms the core of our agent, since it performs the logical inference and directly reasons about knowledge. In particular, it is the module which analyses both current state of the world and how it will change as a result of performing a particular action. To this end, the agent uses a variant of Active Logic, augmented with some ideas from Situation Calculus.

Active Logic (Elgot-Drapkin *et al.*, 1999) is a reasoning formalism which, unlike classical logic, concerns the *process* of performing inferences, not just the final outcome (fixed point) of the entailment relation. In particular, instead of classical notion of theoremhood, AL has i-*theorems*, i.e. formulae which can be proven *in i steps*. This allows an agent to reason about *difficulty* of proving something, to retract knowledge found inappropriate and to resolve contradictions in a meaningful way, as well as makes it aware of the passage time and its own non-omniscience.

To this end, each formula in AL is labelled with the step number when it was derived. Therefore, the *modus ponens* inference rule looks like this:

$$\frac{i : \alpha, \alpha \Rightarrow \beta}{i + 1 : \beta}$$

and means "if at step i formulae α and $\alpha \Rightarrow \beta$ are known, then at step $i + 1$ formula β will be known." Moreover, there is a special inference rule $\frac{i : Now(i)}{i+1 : Now(i+1)}$, which allows an agent to refer to the current moment and to explicitly follow the passage of time. A more in-depth description of Active Logic, and especially its way of handling time, can be found in (Purang *et al.*, 1999).

Following ideas of (Nowaczyk, 2006) we have decided to augment Active Logic with some concepts from Situation Calculus (Reiter, 2001). In particular, in order to have the agent reason about changing world, every formula is indexed with current situation. Furthermore, since the agent needs to reason about effects of executing various plans, we additionally index formulae with the plan the agent is considering. Therefore, a typical formula our agent reasons about looks like this:

$$Knows(s, p, Neighbour(a2, b2))$$

and mean "an agent knows that after executing plan p in situation s, squares $a2$ and $b2$ will be adjacent" (we use chess-like notation for naming squares, with letters designing columns and numbers designing rows). This formula is only mildly interesting, as its validity depends on neither s nor p, at least in the Wumpus domain. But:

$$Knows(s, p, \neg Wumpus(b2))$$

which means "an agent knows that after executing plan p in situation s, Wumpus will not be on $b2$," *does*, obviously, depend on s, since agent's knowledge changes as it acts in the world. It still does not, however, depend on p itself. Clearly, no new knowledge can be obtained by simply *considering* some plan (without actually executing it). If an agent $Knows(s, p, \neg Wumpus(b2))$, then it must also $Know(s, \emptyset, \neg Wumpus(b2))$, where $\emptyset$ stands for an empty plan[1]. Therefore, the really interesting formulae are the ones like:

$$Knows(s, p, Wumpus(b3)) \vee Knows(s, p, Wumpus(c2))$$

which means "an agent knows that after executing plan p in situation s, it will *either* know that there is Wumpus on $b3$ or that there is Wumpus on $c2$". As an example of reasoning by cases and predicting action results, this is exactly the kind of knowledge that we are interested in agent inferring — it *does* tell important things about quality of the plan being considered. If all the agent knew before was:

$$Knows(s, \emptyset, Wumpus(b3) \vee Wumpus(c2))$$

than clearly executing p is useful. For a human "expert," such p looks like a good plan. The goal of our research is to make *an agent* be able to reason about plans in exactly this way. It is our intuition, supported by preliminary experiments reported on in this paper, that creating a *domain independent* Actor module which would efficiently select good plans by learning from experience using formulae like the one above is possible.

One more thing worth mentioning is the planner submodule. As we stated earlier, our agent creates conditional, partial plans. For example, if in an initial situation the agent is on square $a1$ and it does not smell there, the total plan for solving 3x3 Wumpus game has 13 steps (if we consider all possible positions of the monster). This is not much, of course, but it grows very fast as we increase the size of the board or complicate rules of the game even slightly.

Therefore, since having an agent create complete plans is infeasible in many domains, we have decided to settle for *partial* plans. In the experiments reported in this paper, we consider plans of length one and two only. In order to make plan evaluation more meaningful, we allow those plans not only to be simple (sequential) but also *conditional*, i.e. to have branches which depend on agent's observations. It is our intuition that such conditional plans will be, in many domains, much easier to classify as either good or bad ones.

For example, with an agent on square $a1$, one simple plan is "$a2$", meaning "go to $a2$", and another is "$a2a3$", meaning "go to $a2$ and then go to $a3$". A conditional plan could be "$a2\ ?\ a1 : b2$", meaning "go to $a2$ and if it smells there go back to $a1$, else go forward to $b2$". In Wumpus domain it is difficult to find a simple plan longer than one step which is good, while finding a good conditional one is much easier.

Actor

The Actor module is an overseer of the Deductor and works as a controller of the agent as a whole. In its ultimate form, it is expected to do three main things. First, it should guide the reasoning process by making it focus on the plans most likely to be useful. Second, it should decide when enough time has been spent on deliberation and no more interesting results are likely to be obtained. Third, it should make decisions to execute a particular plan from Deductor's repertoire.

In this paper we have decided to focus more on the interactions between learning and deduction, so Actor's functionality has been simplified significantly. The Deductor module uses an incomplete reasoner which always terminates, therefore Actor does not need to decide *when* to begin plan execution — it simply lets Deductor infer everything it can about each of the available plans and chooses the best one based on all the available information.

Learner

The ultimate goal of the learning module is to provide Actor with knowledge about how to choose the plan to be executed next, which plans are most likely to lead to good results (and thus should be reasoned about) and when enough time has been spent on deliberation and the agent should start acting.

In the current setup we use the ILP algorithm called PROGOL (Muggleton, 1995), since it is one of the best known ones and its author has provided a fully-functional, publicly available implementation. PROGOL is based on the idea of *inverse entailment* and it employs a covering approach similar to the one used by FOIL, in order to generate hypothesis consisting of a set of clauses which cover all positive examples and do not cover any negative ones. An important feature are mode declarations, where user specifies which predicates can be used in the hypothesis being learned, as well as their arity and argument types.

In our case, we were interested in learning, first of all, to distinguish "bad" plans early, so that Actor can spot them

[1] Since in our game the player has no way of changing Wumpus' position, the actual validity of "$Wumpus(b2)$" remains constant, only agent's knowledge is changing

and instruct Deductor not to waste time deliberating about them. In this section we describe our experiments on how to represent Deductor's knowledge base in a way accessible to the PROGOL algorithm. In particular, our goal was to analyse the relationship between quality of learning and the amount of domain specific knowledge put into data transformation between Active Logic and PROGOL.

The first major obstacle was the closed world semantics used by PROGOL. In the Deductor, in order to deal with incomplete knowledge the agent has about the world, we had to employ open-world semantics — from the mere fact that agent is unable to prove something it does not follow that it is false.

As a data set we have used three example runs of a Wumpus game on a very small, 3x3 board. The player had, in each case, considered 134 plans, which is the total number of length 2 plans (both simple and conditional ones) in four situations: the player started on $a1$, first moved to $a2$, then to $b2$, and finally to $c3$. In the first run, it noticed that it smells on $b2$, and after moving to $b3$ and not dying, it figured out that Wumpus is on $c2$. The second and third runs were similar, except Wumpus was on $a3$ and $c1$, respectively.

In the experiment reported here, we have assumed the agent has perfect knowledge about which plans (training examples) are potentially bad. It is possible since our reasoner, even if incomplete, is powerful enough to eventually discover whether there is any possibility of agent dying due to executing a particular plan in a given situation. It is not the most useful setup, however. We have made some preliminary tests in a more *simulation-like* case, where an agent executes a plan and observes whether it is a fatal one or not, and learns based on that experience. The problem is that the agent may just get lucky when executing a dangerous plan. Even though PROGOL caters for the possibility of noisy data, we have found its features rather insufficient for this concrete application.

Our first approach was to use as little domain-specific knowledge as possible, so we have purposefully left PROGOL mode declarations open for all background predicates, since they can not be automatically extracted from the knowledge available to the agent. We have also decided not to filter agent's knowledge in any way, except for removing Active Logic specific extensions and the axioms which remain constant throughout all situations and all plans (like geometrical relationships). The one point worth noting is that we have introduced five predicates ($position$, $final$, $finalSmell$, $finalNoSmell$ and $passThrough$) describing the plan itself. They mean, respectively, "agent's initial position," "agent's final position" (for simple plans), "final position if it smells" and "final position if is doesn't smell" (for conditional ones), and "the square which the player will pass" (for plans of length two or more).

We have run PROGOL ten times, with different numbers of training examples. We started with one positive and one negative example only, and kept increasing their numbers up to 20 positive and 20 negative examples (using different number of positives and negatives did not lead to any interesting results). We present the accuracy achieved by the learned hypothesis (an average of 50 trials) as the lowest curve (marked "without mode declarations") in figure 1. It can be easily seen that learning quality is too low to be practically useful.

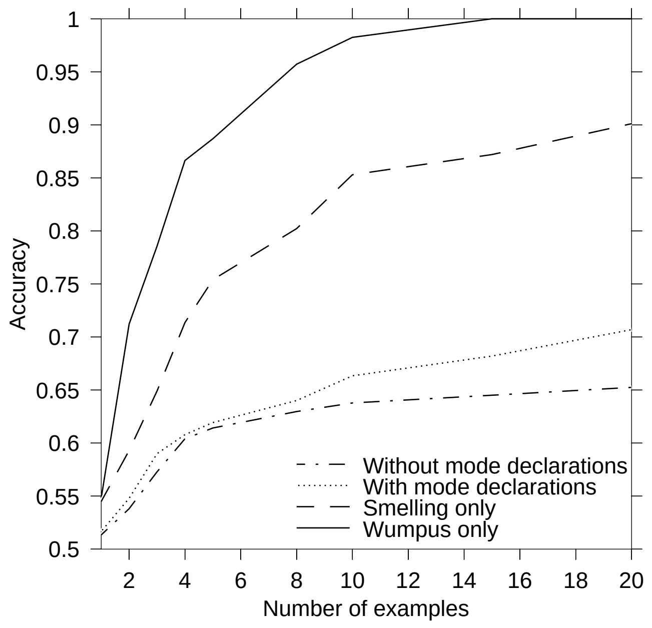

Figure 1: Results of learning

This result is not really surprising as it is a well known fact that specifying appropriate mode declarations is very important for PROGOL. Therefore, our next step was to provide exactly them. The task of writing (by hand) the necessary declarations is not particularly difficult for a user who understands the domain well, but it can be tiresome nevertheless. In addition, requiring an expert to provide mode declarations for *all* predicates in the domain is somewhat pointless, as he certainly has enough knowledge to only label those that are actually appropriate for defining target predicate.

Because of that, in our second experiment we have provided mode declarations for the meaningful predicates only. In particular, we have introduced three predicates describing the smelling phenomenon ($maybeSmells$, $knowsClear$ and $knowsSmell$), as well as three predicates describing the potential positions of the Wumpus ($maybeWumpus$, $noWumpus$ and $knowsWumpus$), all with board squares as arguments. As mentioned above, due to the closed-world semantics of PROGOL we actually need three predicates to describe all possibilities [2]. As can be easily seen on figure 1 (curve marked "with mode declarations"), even such a small amount of domain knowledge is enough to greatly improve quality of learned hypothesis. We have again run the experiment ten times, for different number of examples, and present average accuracy of 50 trials.

It is also important to note that it is not necessary to achieve 100% accuracy in our application. One interesting

[2]We do not, actually, *need* all three — two would be, theoretically, sufficient, but would make learning much mode difficult since good formulae would become more complex

feature of our learning setting is that false negatives are not overly problematic: the point is to save some computations by discarding useless plans early, so if some bad plans are *not* detected, the worst that can happen is that some computations will still be wasted. False positives, on the other hand, are much more dangerous, since if an Actor removes a useful plan from considerations, the overall quality of the solution can deteriorate. However, there is no way to express this distinction in PROGOL terms, so we have decided not to separate accuracy into positive and negative parts.

Encouraged by the success with mode declarations, we have decided to perform two more experiments. In the background knowledge which we have identified as relevant for the concept of bad plans, there were two separate components: information about squares where it smells and information about squares on which Wumpus might hide. In principle, each one of them contains, by itself, enough information to express target concept. Therefore, in the third experiment, we have only used predicates $maybeSmells$, $knowsClear$ and $knowsSmell$. As can be seen from curve "smelling only" in figure 1, expressing the notion of bad plans using only those three predicates proved too difficult for PROGOL.

On the other hand, in our forth and final experiment, PROGOL managed to learn to perfectly identify bad plans using predicates $maybeWumpus$, $noWumpus$ and $knowsWumpus$ from as few as 30 examples chosen at random. The learned definition of a bad plan looked like this:

$$\begin{aligned}
badPlan &\Leftarrow final(A), maybeWumpus(A).\\
badPlan &\Leftarrow finalSmell(A), maybeWumpus(A).\\
badPlan &\Leftarrow passThrough(A), maybeWumpus(A).
\end{aligned}$$

It is interesting to note that as few as five *hand-chosen* example plans suffice for PROGOL to learn the correct definition, which opens up interesting possibilities for an agent to *select* learning examples in an intelligent way.

Having established that successful learning is possible, one more thing that should be shown is whether it is actually *useful*. In our implementation (which is designed for flexibility of reasoning rather than its speed) analysing a complete game of Wumpus takes (depending on monster's real position) on the order of 15 hours. If the Actor knows how to identify bad plans and forces Deductor to ignore them, the total time drops down dramatically, to about *six hours*. This is a clear confirmation of our claim that knowledge gained due to learning from experience can be very useful to improve efficiency of reasoning.

Wumpus position	Full time (hours)	Improved time (hours)	Time decrease (percent)
c2	16.07 h	4.41 h	72.58%
a3	14.72 h	5.52 h	62.49%
c1	15.23 h	7.18 h	52.84%

Table 1: Usefulness of learning

Finally, we would like to point out that PROGOL algorithm, while a very efficient one, is rather poorly suited for the class of problems we face. It was enough as a proof of concept and to show the general usefulness of learning as such, but our next step will be to find a different one, better adapted to the particular needs of evaluating plans.

Related Work

Combination of planning and learning is an area of active research, in addition to the extensive amount of work being done separately in those respective fields.

The first to mention is Dietterich & Flann (1995), which presented results establishing conceptual similarities between explanation-based learning and reinforcement learning. In particular, they discussed how EBL can be used to learn action strategies and provided important theoretical results concerning its applicability to this aim.

There has been significant amount of work done in learning about what actions to take in a particular situation. One notable example is Khardon (1999), where author showed important theoretical results about PAC-learnability of action strategies in various models. In Moyle (2002) author discussed a more practical approach to learning Event Calculus programs using Theory Completion. He used extraction-case abduction and the ALECTO system in order to simultaneously learn two mutually related predicates ($Initiates$ and $Terminates$) from positive-only observations. Recently, Könik & Laird (2004) developed a system which is able to learn low-level actions and plans from goal hierarchies and action examples provided by experts, within the SOAR architecture.

The work mentioned above focuses primarily on learning how to act, without trying to reach conclusions in a deductive way. In a sense, the results are more similar to the reactive-like behaviour than to classical planning system, with important similarities to the reinforcement learning.

One attempt to escape the trap of large search space has been presented in Džeroski, Raedt, & Driessens (2001), where relational abstractions are used to substantially reduce cardinality of search space. Still, this new space is subjected to reinforcement learning, not to a symbolic planning system. A conceptually similar idea, but where relational representation is actually being learned via behaviour cloning techniques, is presented in Morales (2004).

Outside the domain of planning, there is a lot of important research being done in the learning paradigm.

Recently, Colton & Muggleton (2003) showed several ideas about how to learn interesting facts about the world, as opposed to learning a description of a predefined concept. A somewhat similar result, more specifically related to planning, has been presented in Fern, Yoon, & Givan (2004), where the system learns domain-dependent control knowledge beneficial in planning tasks.

From another point of view, Khardon & Roth (1995, 1997) presented a framework for learning done "specifically for the purpose of reasoning with the learned knowledge," an interesting early attempt to move away from the *learning to classify* paradigm, which appears to dominate the field of machine learning.

Yet another track of research focuses on (deductive) planning, taking into account incompleteness of agent's knowledge and uncertainty about the world. Conditional plans, generalised policies, conformant plans, universal plans are the terms used by various researchers (Cimatti, Roveri, & Bertoli, 2004; Petrick & Bacchus, 2004; van der Hoek & Wooldridge, 2002; Bertoli, Cimatti, & Traverso, 2004) to denote in principle the same idea: generating a plan which is "prepared" for all possible reactions of the environment. This approach has much in common with control theory, as observed in Bonet & Geffner (2001) or earlier in Dean & Wellman (1991). We are not aware of any such research that would attempt to integrate learning.

Conclusions

We have presented an architecture for rational agents that combine planning, deductive reasoning, inductive learning and time-awareness in order to operate successfully in a dynamic environment. Our agent creates conditional, partial plans, reasons about their consequences using an extension of Active Logic with Situation Calculus features, and employs ILP learning to generalise past experience in order to distinguish good plans from bad ones.

In this paper we report on our initial experiments with using PROGOL learning algorithm to identify bad plans early, in order to save agent the (wasteful) effort of deliberating about them. We analyse how the quality of learning depends on the amount of additional, domain-specific knowledge provided by the user. Finally, we show that successful learning can result in a dramatic decrease of agent's reasoning time.

Several ideas for future work have been mentioned throughout the text, for example the need for a more efficient planner, the ability of Deductor to prioritise most interesting plans, allowing an agent to estimate its own reasoning progress and to consciously choose between deliberation and acting, finally, learning rules for helping an Actor choose the best plan. We want to reiterate, however, that our next step will be to find learning algorithms better suited for the particular needs of evaluating plans, since PROGOL does not appear to be appropriate for it.

Acknowledgements

I would like to thank my supervisor, ..., for his input into this work and for many discussions about the concepts of rational agents in general, as well as for allowing me to work on issues that genuinely interested me, even when it looked (or, sometimes, still look) like they lead nowhere near the PhD thesis I should be writing.

References

Bertoli, P.; Cimatti, A.; and Traverso, P. 2004. Interleaving execution and planning for nondeterministic, partially observable domains. In *European Conference on Artificial Intelligence*, 657–661.

Bonet, B., and Geffner, H. 2001. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence* 14(3):237–252.

Chong, W.; O'Donovan-Anderson, M.; Okamoto, Y.; and Perlis, D. 2002. Seven days in the life of a robotic agent. In *GSFC/JPL Workshop on Radical Agent Concepts*.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159(1-2):127–206.

Colton, S., and Muggleton, S. 2003. ILP for mathematical discovery. In *13th International Conference on Inductive Logic Programming*.

Dean, T., and Wellman, M. P. 1991. *Planning and Control*. Morgan Kaufmann.

Dietterich, T. G., and Flann, N. S. 1995. Explanation-based learning and reinforcement learning: A unified view. In *International Conference on Machine Learning*, 176–184.

Džeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43(1/2):7–52.

Elgot-Drapkin, J.; Kraus, S.; Miller, M.; Nirkhe, M.; and Perlis, D. 1999. Active logics: A unified formal approach to episodic reasoning. Technical Report CS-TR-4072, University of Maryland.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *International Conference on Automated Planning and Scheduling*.

Khardon, R., and Roth, D. 1995. Learning to reason with a restricted view. In *Workshop on Computational Learning Theory*.

Khardon, R., and Roth, D. 1997. Learning to reason. *Journal of the ACM* 44(5):697–725.

Khardon, R. 1999. Learning to take actions. *Machine Learning* 35(1):57–90.

Könik, T., and Laird, J. E. 2004. Learning goal hierarchies from structured observations and expert annotations. In *ILP*.

Morales, E. F. 2004. Relational state abstractions for reinforcement learning. In *ICML-04 Workshop on Relational Reinforcement Learning*.

Moyle, S. 2002. Using theory completion to learn a robot navigation control program. In *ILP*.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming* 13(3-4):245–286.

Nowaczyk, S. 2006. Partial planning for situated agents based on active logic. In *Workshop on Logics for Resource Bounded Agents, ESSLLI 2006*.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *International Conference on Automated Planning and Scheduling*, 2–11.

Purang, K.; Purushothaman, D.; Traum, D.; Andersen, C.; and Perlis, D. 1999. Practical reasoning and plan execution with active logic. In Bell, J., ed., *Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality*, 30–38.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in AI, 2nd edition.

van der Hoek, W., and Wooldridge, M. 2002. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.