Relative Relevance of Subsets of Agent's Knowledge

Sławomir Nowaczyk and Jacek Malec[*]
Department of Computer Science
Lund University

October 7, 2008

Abstract

We study agents situated in partially observable environments, who do not have the resources to create conformant plans. Instead, they create conditional plans which are partial, and learn from experience to choose the best of them for execution. Our agent employs an incomplete symbolic deduction system based on Active Logic and Situation Calculus for reasoning about actions and their consequences. An Inductive Logic Programming algorithm generalises observations and deduced knowledge in order to distinguish "bad" plans early, before agent's computational resources are wasted on considering them.

In this paper we present experiments which show that in order for learning to be successful, an agent's knowledge needs to be filtered. We argue that this filtering nicely matches the intuitive notion of "knowledge relevance". We also present a heuristic scheme, combining several natural rules, which can be used to automatically determine which formulae should be used for learning.

1 Introduction

Rational, autonomous agents able to survive and to achieve their goals in dynamic, only partially observable environments are the ultimate goal of AI since its beginning. Quite a lot has already been done towards achieving that dream, but dynamic environments still remain a big challenge for autonomous systems. In particular, nontrivial environments that are only partially observable pose demands which are beyond the current state of the art, possibly except when dedicated solutions are developed for narrow domains.

One of the major ways of coping with uncertainty and lack of knowledge about current situation is to exploit previous experience. Experience may be gathered

[*]This work has been partially supported by the EU-project SIARAS, Skill-Based Inspection and Assembly for Reconfigurable Automation Systems (FP6 - 017146).

in many ways: by memorising, by being organised in some mental models, by creating appropriate probability distributions over possible courses of events, to name just a few approaches to learning.

All realistic agents necessarily have limited resources, both mental (CPU power, memory size) and physical (limited repertoire of possible actions). In our research we are interested in developing rational, situated agents that are aware of their own limitations and can take them into account [2]. To facilitate this, we use Active Logic [7] for knowledge representation, which characterises reasoning as an ongoing process, instead of focusing on a fixed point of entailment relation.

Due to limited resources and to the necessity of remaining responsive in a dynamic world, situated agents cannot be expected to create complete plans for achieving their goals. They need to consciously alternate between reasoning, acting and observing their environment, or even do all those things in parallel. We aim to achieve this by making the agent create short conditional partial plans and execute them, learning more about its surroundings throughout the process.

In our approach, the agent continuously reasons about the world, enriching its knowledge using both observations and deduction. In order to generalise its experience, due to resource limitations, it needs to select only the *most relevant* subset of its own knowledge for learning, as it is not practical to expect ILP algorithms to be able to find useful generalisations among vast amounts of unrelated knowledge.

An important obstacle is the lack of satisfactory objective measure of knowledge relevance. Typically, the most interesting insights from the logical perspective come from *equivalence* relation, where one can say that two sets of formulae denote exactly the same models. This level of abstraction, however, is not quite sufficient for us, since it does not capture the qualities we are interested in: two theories equivalent in the semantic sense can differ greatly in how easy it is to generalise from them and to learn new concepts.

A primary goal of this work is to explore how the agent can automatically choose some parts of its knowledge base in such a way as to maximise the quality of learning. Within the bounded computational resources the agent possesses, it is not feasible to generalise blindly. The generalisation process is inherently difficult, with very high branching factor and therefore limiting input data is of crucial importance for realistic agents.

This field of study has seen a lot of interest in the past years, but the major problem has often been lack of formal model as to what "relevance" formally means. Throughout this paper we would like to argue that the *most relevant* knowledge can be defined as the one which leads to the best learning results.

The paper is divided as follows. First, we briefly describe the domains used in our experiments. Then we present the agent architecture, followed by a discussion of knowledge relevance, which underlies the rest of this work. Afterwards we describe each of the agent's modules, with focus put on learning. Finally, we present the experimental results, followed by a discussion of related work. The paper ends with some preliminary conclusions.

2 Experimental Domains

In the experiments we have used two domains: the game of Wumpus [22] and a chess ending. The Wumpus game is very simple, easy to understand, and people have no problems playing it effectively as soon as they learn the rules. For artificial agents, however, even such a simple game remains a serious challenge.

The game is played on a square board. There are two characters, the player and the Wumpus. The player can, in each turn, move to any neighbouring square, while the Wumpus 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. As Wumpus is a smelly beast, the player can immediately notice if the creature is in the vicinity. The goal of the game is to find out the exact location of the monster, by moving throughout the board and observing on which squares does it smell. At the same time, if the player enters the square occupied by the beast, he gets eaten and loses the game.

Our second domain, which we call Chess, is a modified version of "king and rook vs king and knight" chess ending. Since we are interested in partially unknown environments, we assume for the sake of experimentation that the agent does not know how the opponent's king is allowed to move — *a priori* any move is legal. The agent will need to use learning to discover what kinds of moves are actually possible.

The goal of our research is somewhat akin to the *General Game Playing Competition* [9]: our agent is given some declarative 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.

3 Agent Architecture

The architecture of our agent (see Fig. 1) consists of four 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 among them all.

The *Deductor* module is the one responsible for symbolic reasoning. It uses a logical formalism based on combination of Active Logic and Situation Calculus (as introduced in [17]) in order to figure out consequences of the agent's current beliefs. Based on the domain knowledge and previous observations, it analyses possible actions and predicts what will be the effect of their execution.

The second module is *Planner*, which generates partial, conditional plans applicable in the agent's current situation. In the reported experiments, our planner was a simple one, generating pre-arranged plans only (all imaginable plans for the Wumpus domain, and some arbitrary set of "interesting" plans for Chess).

The third main module, *Actor*, oversees Deductor's reasoning process and evaluates plans that Planner has come up with, trying to find out which is the most useful one to perform. For this paper, Actor, renamed more appropriately to *Plan*

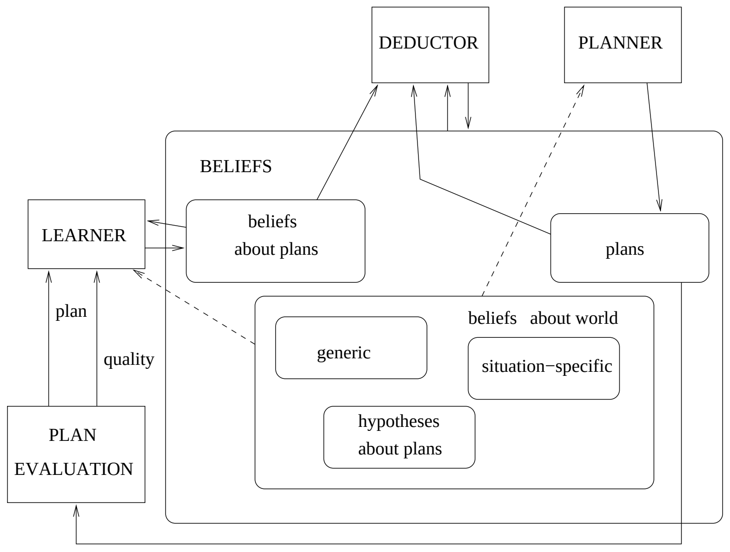

Figure 1: Architecture of the system.

Evaluator, waits until Deductor terminates and only executes plans after this happens, but in general it is Actor's responsibility to balance acting and deliberation.

Finally, the *Learner* module analyses the agent's past experience and induces rules for estimating quality of plans. Results of learning process are used both by Deductor and by 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 can be very difficult to estimate whether a particular plan is a step in the right direction or not. 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 effortsthe vast majority of AI research is being done in specialised subfields and it is our belief that neither of these subfields *alone* can give us truly intelligent, rational agents. Our architecture, which to the best of our knowledge is novel, may be one way to integrate them.

4 Relevance Estimation

Estimating the relevance of a particular formula to the agent's task at hand is a very difficult problem, one which cannot be solved in general. In practice, however, there are certain regularities and conventions which human experts use when encoding domain knowledge for the agent, and it is our belief that some of those conventions can be exploited by the agent.

The basic assumption is that human experts who create agent's initial knowledge base do it in such a way as to maximise its usefulness to the agent (*modulo* mistakes and ignorance, of course). Humans can often reasonably easily determine which parts of the knowledge base are most relevant for solving particular problems, and this is often reflected in the encoded knowledge in many ways. The ability to extract such hints would be very valuable to any rational agent.

We have considered several qualities which can be used as hints that a particular piece of knowledge is "good". We discuss and rationalise them below on a rather abstract level, while in section 8 we will show how we have implement them.

Derived from observations. For logic-based rational agents, there are typically few observations (both because the cost of acquiring them is often high, and because it is well known that they are expensive to reason about), so it is unlikely that they are irrelevant. It is a wise idea, therefore, to try to keep them in the training knowledge whenever possible. Also, it is almost universally the case that the right decisions for an agent *do* depend on the acquired observations.

It is also important to notice that quite often not the raw observations themselves, but rather their aggregations with the rest of the agent's knowledge, form the most desirable input (i.e. $WumpusPosition$ should be used rather than $Smells$; $CanMove$ rather then $Position$).

Consistency between plans. When looking at possible plans within a given situation and analysing their expected outcomes, often similar formulae occur in many cases. Those sometimes encode exactly the same knowledge, and sometimes a different (even opposite) one. For example, it is possible that $Wumpus(a2)$ would be known from executing some branch of plan $p1$, and $\neg Wumpus(a2)$ would be known from executing some branch of plan $p2$.

For the logical sentences most relevant for current problem, one can find multiple situations where the same knowledge holds across the wide spectrum of plans being considered. Having such consistency sufficiently often is one hint that the given class of formulae is actually correlated with agent's actions.

Of course, interesting formulae are not always like this, for many situations the values of any given predicate *need to* vary, which corresponds to the case where the real value is not known and the agent is actively exploring different possibilities. It is important that both ends of the spectrum are represented sufficiently.

Inheritance chains. We assume that the initial domain knowledge is chosen in such a way as to maximise agent's performance, therefore one would expect that the *final* result of reasoning process is likely to be the most important one. One way to capture this would be to assume that the more difficult some sentence is to infer (measured, e.g., as the number of steps needed to deduce it), the more relevant it is. This corresponds to the idea that such formula "contains" more knowledge than easier ones.

This, however, would be overly susceptible to some irrelevant knowledge being present in the expert's description of the domain. On the other hand, we have noticed in the reasoning traces of our agent a related and interesting pattern. It is quite often the case that the deduction tree of some formulae is rather degenerate: it contains a singly branch which is very long, while all the others are very short ones. We call such tree a *chain*. The middle sentences in such chains are very often boring indeed, with the end ones containing everything of interest.

Similarities between plan branches. When looking at conditional plans, there is usually a number of branches which correspond to different possible observations that can be made during execution. Looking for regularities between such branches can provide interesting insight into which parts of the agent's knowledge base are the most relevant to choosing the best plan.

Axioms are boring. Rather obviously, learning from the domain axioms is typically not the best idea. It is important to take advantage of the results of reasoning the agent has performed, and often there are aggregations available which express desired properties in ways much cleaner than by barebone axioms alone.

One exception here, however, is the case when some axioms are never used in the deduction. If a human expert provides a number of formulae which are useless for reasoning, then the agent can reasonably expect them to play a role in learning.

Explore time differences. Given that our agent reasons using Active Logic, we have one more important clue as to interdependences between pieces of knowledge: the differences in step numbers of when particular formulae have been deduced.

One common pattern of reasoning is confined to a short time interval, with sentences inferred in the previous time step being used in the current one, and so on. On the other hand, sometimes an older formula is used in conjunction with the newest one, and we believe that this hints at something interesting. When the old knowledge is relevant to the current situation, it is more likely that the same knowledge will be useful again in the future.

Another interesting idea would be to assume that things deduced earlier are likely to be *generic* knowledge, therefore if they are applicable now, they will be applicable again. At the same time, a large percentage of the most recently inferred formulae will be specific ones, fitting current situation but not useful anywhere else.

Smarter rules of inference. A large appeal of Active Logic is its powerful mechanism of adding domain-specific extensions to the well established sets of inference rules. Typical examples make use of timestamps and observation functions, but it is also possible to specify, among other thing, that some rules of inference are more likely to lead to relevant sentences (for example, *modus ponens* is interesting while *negation introduction* is boring).

Limited in number. On a more technical side, having too many formulae presented to ILP algorithm, even if most of them are "good", means that it will fail to learn anything useful anyway (or, at the very least, that the generalisation process will be very fragile).

When search for solution is not able to meaningfully analyse majority of the hypothesis space, the resulting knowledge will be pretty much random set of clauses which fit learning examples to some minimal degree. With the number of degrees of freedom that ILP algorithms have, such results are typically completely useless.

Covering of the domain. The above qualities mostly analyse sentences in separation, but it is important to also look at the bigger picture. An important issue is that a large portion of both initial and current knowledge base "contributes", one way or another, to the learning process. The assumption throughout this paper is that the domain knowledge is basically expected to be useful, so whichever knowledge we decide to present to PROGOL, it should have at least some chance of being influenced by all parts of it.

There are many possible definitions of what "influence" might mean, but the most obvious one is to track which formulae were used to deduce the set we are interested in (akin a very basic truth-maintenance system). Of course the fact that sentence β is deduced from α does not necessarily mean that the *rationale* for including α in domain description has also been captured by β, but it at least lack one negative indication.

Finally, an important decision is whether to consider each formula separately, or rather cluster them in some way. We have decided to cluster sentences according to the predicates they contain. It can happen, for example, that $Neighbourhood$ relation between two squares is interesting and important for learning, while the same relation between two others is irrelevant. There is a serious risk, however, of significantly distorting the domain knowledge, to the degree where the resulting hypothesis will generalise very badly. We have decided to forfeit some of this expressiveness and we only make decisions as to which predicates are to be included in the training knowledge (i.e. if we decide that $Neighbourhood$ is relevant, then *all* formulae containing $Neighbourhood$ will be used for learning).

We feel the need to stress that our work here is of heuristic nature (in the colloquial sense of the word) and that the rules we present can be easily fooled. It is not our aim, however, to present a bulletproof system, but rather to explore the natural clues left by human experts.

5 Deductor

Deductor performs logical inference and directly reasons about the agent's 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 [20], augmented with some ideas from Situation Calculus [21].

Active Logic is a reasoning formalism which, unlike classical logic, concerns the *process* of performing inferences, not just the final extension 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 the agent aware of the passage time and its own non-omniscience. An in-depth description of Active Logic, and especially its way of handling time, can be found in [20].

In order to properly represent actions we have decided to augment Active Logic with some concepts from Situation Calculus. 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. Details of this approach can be found in [17].

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 where actions depend on agent's observations. We expect that such conditional plans will be, in many domains, much easier to classify as either good or bad ones.

6 Planner and Actor

The Actor module is an overseer of 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 guides the reasoning process by making it focus on the plans most likely to be useful. Second, it decides when enough time has been spent on deliberation and no further interesting results are likely to be obtained. Third, it makes decisions to execute a particular plan from Deductor's repertoire.

In this paper we focus more on the interactions between learning and deduction, so both Planner and Actor have been significantly simplified. Planner does not use any heuristics and simply creates all possible plans, although we aim to use existing planners to efficiently create only the "reasonable" plans. Also, Deductor 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.

7 Learner

The ultimate goal of the learning module is to provide Actor with knowledge necessary to choose the best plan for execution and to stop deliberation when too much time has been spent on it without any new interesting insights.

A step in this direction is to learn how to detect "bad" plans early, so that Deductor does not waste time deliberating about them. We have defined bad plans to be those which can kill the agent (in the Wumpus domain), and those that lead to losing the rook (in the Chess domain).

In the experiments reported here, we assume that the agent has perfect knowledge about which plans (training examples) are bad ones. This is a justified assumption for Chess domain, where the opponent does not make trivial mistakes and whenever it is possible for him to capture the rook, he will do so. In Wumpus, the distinction is not so clear — it is possible that the agent will get lucky and not die even though it executes a dangerous plan, simply because the beast is in a favourable position.

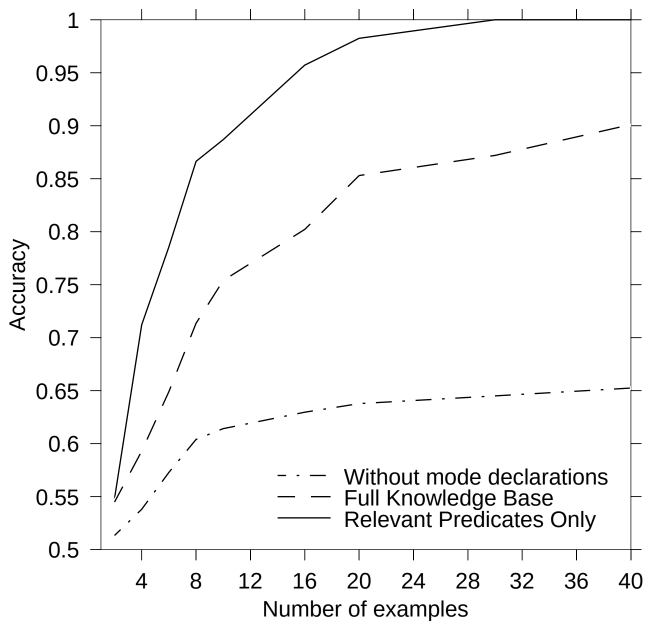

Figure 2: Results of learning in Wumpus domain.

We have made some preliminary tests in a more *simulation-like* environment, where an agent executes a plan and observes its actual effects only. Even though the learning algorithm we used allows for the possibility of noisy data, we have found that rather insufficient for our needs. Thus, the experiments we report here do not contain any noise — we required Deductor to *prove* whether a plan is safe or not before it was used as a training example.

8 Results of the experiments

In a previous paper [18] we have presented the results where the agent learns to distinguish bad plans early. We have shown that PROGOL is able to find the correct definition from as few as 30 randomly-chosen examples. Such a definition allows the agent to save up to 70% of its reasoning time. In order to obtain such results, however, we had to provide additional domain knowledge specifically for the purpose of learning.

This extra knowledge consisted of two parts. The first were so called *mode declarations*, which serve to reduce the hypothesis search space by limiting types of predicate arguments, as well as by specifying which ones are input and which are output arguments and whether variables or constants should be used. This part is mostly mechanical, and although we envision some interesting issues with possibility of automating it, we do not focus on it here.

The other part is what inspired us to the work reported here. It turned out

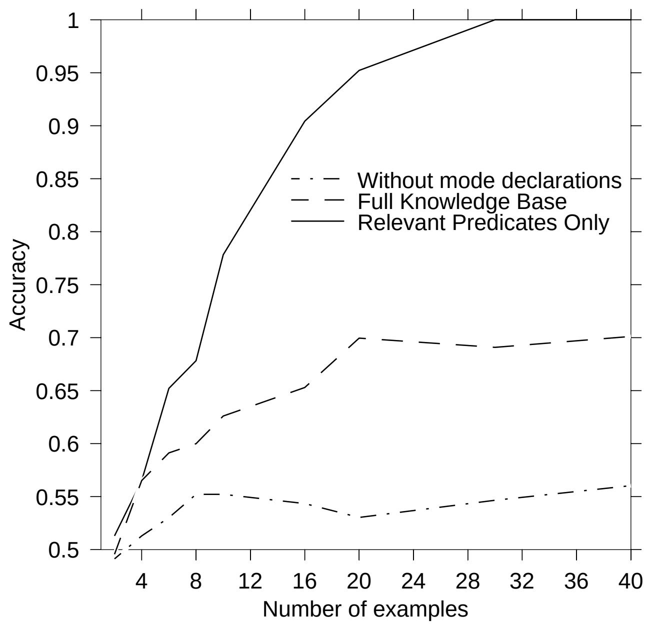

Figure 3: Results of learning in Chess domain.

that PROGOL was unable to perform meaningful generalisation because it was given *too much* knowledge. Therefore, in order to obtain good learning results, we needed to specify which parts of the knowledge base are the most *relevant* ones.

The results of learning we have achieved can be seen in Figures 2 and 3, for *Wumpus* and *Chess* domains, respectively. It can be easily seen that selecting the right subset of the whole knowledge base can be very beneficial. However, in the reported work this had to be done by hand.

As explained in section 4, there is a number of guidelines the agent could use to automatically determine which predicates are the most relevant. In order to check those assumptions, we have implemented a system for evaluating subsets of knowledge according to those guidelines.

We start with qualities that can be evaluated on a per-formulae basis, i.e., those where for each formula we can estimate to what degree it can be considered a “good” one. One of the most successful formalisms for dealing with this class of problems, in particular one allowing for systematic aggregation of several independent criteria, is the *fuzzy sets* approach. We have decided to use it in this work.

Derived from observations. For every formula, we count the number of *observations* in its complete deduction tree and calculate the membership value according to the function presented in Figure 4.

Consistency between plans. For every atomic formula (i.e. in the form $Predicate(args)$) we look at all the other plans in the same situation, and determine the ratio of those which contain $Predicate(args)$ to those which con-

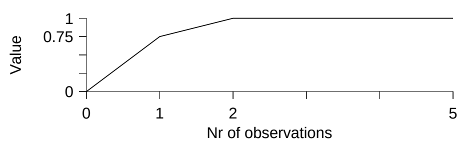

Figure 4: Membership function "Derived from observations".

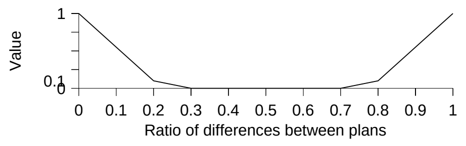

Figure 5: Membership function "Consistency between plans".

tain $\neg Predicate(args)$. We then calculate the membership value according to the function presented in Figure 5.

Inheritance chains. For every formula we analyse its deduction tree and check whether it is a *chain* (very unbalanced tree). If yes, we calculate the ratio of formula's position in it to the total length of the chain. We calculate the membership value according to the function presented in Figure 6.

Axioms are boring. For every axiom, we count the number of other formulae which are inferred from it, both directly and indirectly. We calculate the membership value according to the function presented in Figure 7.

Explore time differences. For every formula, we check the timestamp of the oldest of the sentences it was inferred from, excluding initial domain axioms, and subtract it from the timestamp of the formula itself. We calculate the membership value according to function presented in Figure 8.

To restate, we are interested in figuring out which predicates should be presented to the learning algorithm. To this end we evaluate all subsets of predicates and choose the best one. For any set of predicates, we start the evaluation by creating the set of all formulae in the knowledge base which contain only the predicates from this set.

For each criterion, then, we calculate the average estimate (value of membership function) of a sentence in this set. This tells us to what degree does the current

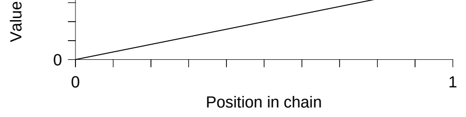

Figure 6: Membership function "Inheritance chains".

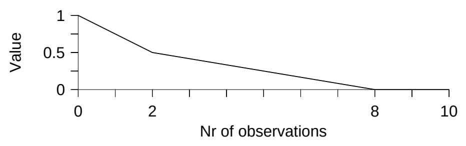

Figure 7: Membership function "Axioms are boring."

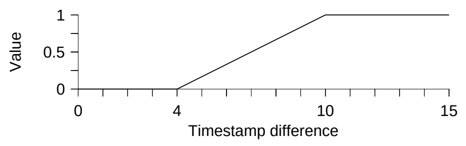

Figure 8: Membership function "Explore time differences."

set of predicates fulfill each of the qualities we have identified. In order to aggregate all five of them into one number we have used the *product t-norm*. Even though its properties do not match the problem perfectly, using anything more complex is not justified given how approximate are the parameters we use.

The seventh quality, **limited in number**, did not really fit the fuzzy set approach. Besides, we have decided to use it as a normalising factor, since different sets of predicates can result in vastly different numbers of formulae being evaluated. Again, we have decided upon the simplest scheme where we divide the aggregate fuzzy membership value by the number of formulae. We call this value *usefulness* of a predicate set.

At this point we are able to evaluate any predicate set, although this cannot yet be used as the final measure, since the optimum value is always achieved by a set of cardinality equal exactly one[1]. The measure works quite well for separately estimating how good each predicate is all by itself. It is very rarely the case, however, that a single predicate would be good for learning — there is almost always a need for having several different predicates.

The final step is to use the **covering of the domain** idea to arrive at a set of predicates which are *together* most likely to provide good learning results. Intuitively, we aim at adding predicates with high individual *usefulness* as long as they are reasonably independent of each other (i.e. as long as they originate from different "parts" of agent's knowledge).

Therefore, we calculate *coverage* of a set of formulae A as the ratio of all formulae in the knowledge base which appear in the deduction tree of at least one formula from A. We arrive at our final *relevance* score by multiplying the *usefulness* of a predicate set by its *coverage*.

We have implemented the above scoring method in a domain-independent way

[1]For any two predicates p_1 and p_2, the *usefulness* of the set $\{p_1, p_2\}$ always lies somewhere in between the usefulness for p_1 and for p_2.

in our agent. We have run it on both the $Wumpus$ domain and on the $Chess$ domain. As we hoped, the sets of predicates with the highest calculated *relevance*, in both domains, turned out to be the ones which lead to good learning results.

It is not obvious how general our results are, since two example domains is too few to reach definitive conclusions. It is enough, however, to suggest that our ideas have merit.

9 Related work

We present an overview of work done on the border of planning and learning, relevant to the approach we have adopted here.

The first to mention is [5], which presented results establishing conceptual similarities between explanation-based learning and reinforcement learning. In particular, they discussed how Explanation-Based Learning can be used to learn action strategies and provided important theoretical results concerning its applicability to this aim.

A lot of work has been done in learning about what actions to take in a particular situation. E.g. [10] showed important theoretical results about PAC-learnability of action strategies in various models. In [16] author discussed a more practical approach to learning Event Calculus programs using Theory Completion. He used extraction-case abduction and the ALECTO system to simultaneously learn two mutually related predicates ($Initiates$ and $Terminates$) from positive-only observations. Recently, [13] developed a system which learns low-level actions and plans from goal hierarchies and action examples provided by experts, within the SOAR architecture. Yet another fresh work close to this approach is documented in [14], where *teleoreactive logic programs*, possibly even recursive ones, are used for representing the action part of an agent. On top of it a learning mechanism, quite similar to ILP, is employed for improving the existing action programs.

In order to handle large search space [6] used relational abstractions to substantially reduce cardinality of search space. Still, this new space is subjected to reinforcement learning, not to a symbolic planning system. A similar idea, but with relational representation being learned via behaviour cloning techniques, is presented in [15].

Outside the domain of planning, there is a lot of important research being done in the learning paradigm. [4] showed several ideas about how to learn interesting facts about the world instead of predefined concepts. A similar result related to planning has been presented in [8], where the system learns domain-dependent control knowledge, beneficial in planning tasks.

From another point of view, [11, 12] 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 dominates the field of machine learning.

Yet another approach 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 [3, 19, 23, 1] to denote in principle the same idea: generating a plan which is "prepared" for all possible reactions of the environment. We are not aware of research that would attempt to integrate learning into this approach.

10 Conclusions

We are developing an architecture for rational agents that combine planning, deductive reasoning, inductive learning and time-awareness in order to operate successfully in dynamic environments. 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 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.

We argue that an important part of this additional knowledge corresponds, to some degree, to the intuitive notion of *relevant knowledge*. We claim that our architecture, and in particular the learning part of it, can provide interesting ways to formalise the notion of "relevance".

We also propose a simple scheme for automatic evaluation of such relevance, which uses *fuzzy sets* to aggregate results from several criteria expressed as natural rules. We show that good results can be obtained on our two example domains.

The research presented here can be continued in many different directions. The most obvious one is to improve the learning algorithm, by making it aware of the actual meaning and origins of its input data. By exploring the difference between *fluents* and *non-fluents*, for example, the hypothesis can better match different situations. Similarly, before learning begins it is too early to commit to a subset of relevant predicates — an ILP algorithm which would take relevance estimations as input and use them to *guide* the generalisation search would be useful.

The exact limits of applicability of the measures we propose here are still to be evaluated, but we show that they are useful, and we are convinced that theoretical discussions touch a very important and difficult topic, therefore we believe they will be interesting for other researchers.

Finally, the architecture we are presenting here is still evolving and the functionality of every module will be expanded in the future.

References

[1] Piergiorgio Bertoli, Alessandro Cimatti, and Paolo Traverso. Interleaving execution and planning for nondeterministic, partially observable domains. In *European Conference on Artificial Intelligence*, pages 657–661, 2004.

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

[3] Alessandro Cimatti, Marco Roveri, and Piergiorgio Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1-2):127–206, 2004.

[4] Simon Colton and Stephen Muggleton. ILP for mathematical discovery. In *13th International Conference on Inductive Logic Programming*, 2003.

[5] Thomas G. Dietterich and Nicholas S. Flann. Explanation-based learning and reinforcement learning: A unified view. In *International Conference on Machine Learning*, pages 176–184, 1995.

[6] Saso Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52, 2001.

[7] Jennifer Elgot-Drapkin, Sarit Kraus, Michael Miller, Madhura Nirkhe, and Donald Perlis. Active logics: A unified formal approach to episodic reasoning. Technical report, University of Maryland, 1999.

[8] Alan Fern, SungWook Yoon, and Robert Givan. Learning domain-specific control knowledge from random walks. In *International Conference on Automated Planning and Scheduling*, 2004.

[9] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72, 2005.

[10] Roni Khardon. Learning to take actions. *Machine Learning*, 35(1):57–90, 1999.

[11] Roni Khardon and Dan Roth. Learning to reason with a restricted view. In *Workshop on Computational Learning Theory*, 1995.

[12] Roni Khardon and Dan Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, 1997.

[13] Tolga Könik and John E. Laird. Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, 64:263–287, 2006.

[14] Pat Langley and Dongkyu Choi. Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, 7:493–518, 2006.

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

[16] Stephen Moyle. Using theory completion to learn a robot navigation control program. In *ILP*, 2002.

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

[18] Sławomir Nowaczyk and Jacek Malec. Learning to evaluate conditional partial plans. 2007. Submitted to Conference on Artificial Intelligence and Soft Computing, ASC 2007, Palma de Mallorca, Spain, August 2007.

[19] Ronald P. A. Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *International Conference on Automated Planning and Scheduling*, pages 2–11, 2004.

[20] Khemdut Purang, Darsana Purushothaman, David Traum, Carl Andersen, and Donald Perlis. Practical reasoning and plan execution with active logic. In John Bell, editor, *Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality*, pages 30–38, 1999.

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

[22] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in AI, 2nd edition, 2003.

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