Heuristics for Planning in Domains with Unbounded, Incomplete Information

Tal Shaked **Daniel S. Weld**
Department of Computer Science & Engineering
University of Washington, Box 352350
Seattle, WA 98195-2350 USA

Introduction

While the Internet Softbots project (Etzioni & Weld 1994) investigated the use of planning algorithms to control software agents (Weld 1996), until recently the area has seen little activity. However, the excitement surrounding the *Semantic Web*, has led to renewed interest in software agents, web service composition, and related topics. With the development of DAML-S (Burstein *et al.* 2002; Ankolenkar *et al.* 2001), a DAML+OIL ontology for describing properties and capabilities of Web services in the form of inputs (similar to preconditions) and outputs (effects), an infrastructure has been created where agents can automatically discover Web Services that are modeled as actions, and then create and execute plans (McIlraith, Son, & Zeng 2001; McDermott 2002).

The major challenge confronting a planning agent operating in an environment such as the Internet is uncertainty about the world-state. But most prior work on planning under uncertainty (*e.g.*, (Peot & Smith 1992; Kushmerick, Hanks, & Weld 1995; Pryor & Collins 1996; Weld, Anderson, & Smith 1998; Bonet & Geffner 2000; Bertoli, Cimatti, & Roveri 2001)) is inadequate because of scalability issues. When a single world-state is as complex as a UNIX file system or the Internet, representing belief states explicitly is out of the question. Furthermore, one requires a sophisticated language simply to represent sensing actions, which like UNIX `ls`, can return an *unbounded amount of information* at execution time.[1] Clearly the *closed world assumption* (Reiter 1978) is no longer valid, yet the open-world assumption leads to paralysis. In this case the agent must execute sensing actions to discover objects and relations while it plans. Furthermore, the agent must maintain a record of where it has *local closed world* (LCW) information (Etzioni, Golden, & Weld 1994; 1997).

Background

The XII and Puccini planners (Golden, Etzioni, & Weld 1994; Golden 1998) controlled an agent that could construct and execute plans to achieve simple goals such as "Print the file, *color-picture*, to a research printer, making sure that the printout will be color, and report the status of the print job," but domain-specific search control rules were required (Golden 1997). More recently Petrick and Bacchus have looked at new ways to represent what an agent knows (Bacchus & Petrick 1998), and have developed a contingent planner that can reason about what it will come to learn from its actions (Petrick & Bacchus 2002). Although both of these planners have been able to solve new kinds of problems, neither one uses sophisticated, general-purpose domain analysis to guide its search for effective plans, and hence they don't scale as well as one would desire.

Fortunately, recent years have seen rapid progress in the speed and quality of plans generated by the newest classical and temporal planners. Many of these advances have resulted from clever ways of developing heuristics to guide the selection of actions, and search through state-space. For example, HSPr does a search through the regression space and calculates a heuristic for each state based on the relaxed step count to reach each subgoal from the start state (Bonet & Geffner 1999; 2001). RePOP uses a planning graph to estimate the cost of achieving a set of subgoals in a partial plan (Nguyen & Kambhampati 2001). Sapa constructs a relaxed temporal planning graph which provides many heuristics based on temporal reachability of goals as well as resource constraints (Do & Kambhampati 2001). LPG does a local search on the planning graph using complicated heuristics that include mutex and precondition counts as well as difficulty in achieving certain propositions when choosing actions to add and remove from an action graph (Gerevini & Serina 2002). Each of these four planners either uses a planning graph directly or computes an estimated step count which is similar to reachability analysis in a planning graph. Heuristic generation by planning-graph analysis has been discussed in (Nguyen, Kambhampati, & Nigenda 2002; Nguyen & Kambhampati 2000), and successfully implemented in many recent planners.

Summary

In this paper, we propose new techniques for computing heuristics that will allow agents to more efficiently plan with unbounded incomplete information. We describe how to adapt a planning graph to handle domains with incomplete

[1]Hence our phrase in the title *unbounded incomplete information*.

```
action ls(d)
     precond:  dir(d)
     effect:  LCW(inDir(!f,d)) ∧
              ∀!f when inDir(!f,d)
                 observe(inDir(!f,d)) ∧
                 observe(type(!f))
```

Figure 1: A simplified representation of the Unix action, `ls`.

information and actions with unbounded sensing capability. Then we consider how such a planning graph can be used to guide the selection of actions when searching for plans, as well as interleaving planning with execution.

Planning Graph Review

Briefly, a planning graph is a leveled structure that begins with the initial state of the world as a list of propositions, and then alternates between action levels and state levels. Action levels contain all actions whose preconditions are satisfied by the previous state level (specified by links), while state levels contain all past propositions plus new ones introduced by effects of the most recent action level (also specified by links). The addition of levels continues until two consecutive levels are identical. Mutexes can be added between propositions and actions on the same level that are mutually exclusive.

Unbounded Information Gathering

We adopt a simplified version of the language and operators used by the first Internet Softbot (Golden & Weld 1996). Specifically we will consider sensing actions with universally quantified effects that can introduce new objects and relations and obtain local closed world information (Etzioni, Golden, & Weld 1997).

Universally Quantified Sensing Actions

Consider the action `ls(d)` shown in figure 1. It has an effect of adding LCW information indicating that the agent will know all files that exist (and do not exist) in the given directory. It will also discover any objects in the directory and add them to its knowledge base. $!f$ acts as a run-time variable and will be bound to each object, and *observe* will add the facts to the database including the type (`file(!f)` or `dir(!f)`), and relation that $!f$ is in d.

Local Closed World Information

Local Closed World (LCW) information allows an agent to have complete information about some part of the world over some set of propositions. This makes it possible for an agent to solve universally quanitifed goals and to avoid repetitive actions that sense the same information. For example, after executing `ls(root)`, an agent knows exactly which files are within *root* and the fact that all other files (including objects not yet discovered) are not in *root*. Note that there are practically an infinite number of facts explicitly listing each file not in *root*. This information is compactly represented by the statement, `LCW(inDir(f, root))`. The idea is that if an agent does not explicitly know some fact, it can check if the information can be inferred from an LCW statement. Specifically any object o that binds with f allows the agent to infer ¬`inDir(o, root)`.

Knowledge Representation

The agent's knowledge will be stored in two databases; F will store facts (can include any literal and its negation), and L will contain LCW information. In general actions can add and remove information from both of these databases. A goal is satisfied when the agent can assert the facts by querying both F and L. For example, consider the case when the agent is in the directory, *root*, and would like to know the value of `inDir(core, root)`. It can begin by executing `ls(root)`. Then the agent can check if M contains `inDir(core, root)`. If neither `inDir(core, root)` or its negation is in M, then the agent can query L, which will contain LCW(inDir(f, root)). Since `inDir(f, root)` unifies with `inDir(core, root)`, the agent can infer that `inDir(core, root)` is false.

Developing Heuristics

There has been considerable work developing heuristics for planners in bounded domains, but extending them to unbounded domains will add new problems that will change and redirect the focus of some heuristics. The general problem will not be confined to an initial state, a goal state, and a finite set of intermediate states, but rather the agent will be in an interactive world where it can increase what it knows about states through sensing actions, and move through intermediate states by executing actions, as it searches for a goal state. We believe a planning graph can effectively capture the structure of such a problem domain.

Extending the Planning Graph

A planning graph will have to be extended to model a domain with unbounded information. Here is a set of issues to consider:

1. How to represent LCW information in the graph
2. Sensing actions that introduce objects and relations
3. Propagating results of sensing actions
4. When to level off graph construction
5. Are there new types of mutexes (*e.g.*, for LCW or sensing)
6. Limiting the graph to relevant information
7. Can the graph be used to derive heuristics to control execution as well as to control the search for a good plan.
8. Learning distributions over the results of sensing actions and incorporating these statistics in the heuristic estimator.

One can see that the facts in M correspond to the propositions in the state levels. A natural way to deal with L is to add LCW information to the state levels as well. Note that unlike propositions, LCW facts may not match directly with preconditions of actions. Instead inference will need to be performed on the LCW facts at a given state level in order to conclude what information (including forall conditions) is available, and which action's preconditions are satisfied.

Sensing actions can vary from distinguishing between one of two states (*e.g.*, is the door open), to unlimited information gathering (*e.g.*, `ls -R` from the root directory). Modeling this in a planning graph can be done by adding a special proposition at the state level that can either represent a set of possible literals, or represent a generic type. A *generic type* can then be *conditionally unified* with preconditions at the next level if some satisfying binding exists. For example, the effects of `ls(root)` could possibly bind to `dir(homes)`, since the action may observe a directory with that name.

As the graph grows, it is important to propagate links from sensing actions. In this way the agent has an idea of which goals are easier than others. Furthermore links (or propositions) can have weights that accumulate how much prior sensing was needed to obtain a certain proposition.

Unlike normal planning graphs, adding generic types can theoretically lead to an infinite amount of bindings and therefore the graph will never level off. This means that some heuristic will be needed to decide when to stop growing the graph, and when to add levels during plan construction. One possibility is to focus on plan construction and execution if all goal conditions are present in a level, or if new actions are all dependent on conditions of sensing actions. Then in some cases the problem may become more of an agent-centered search (Koenig 2001).

Although the standard mutexes can be included, generic types and LCW facts will need new mutex rules (including the related actions). Since a generic type can bind to any number of objects, it seems impractical to add conditional mutexes for all cases. Dealing with LCW mutexes is more reasonable since there are well-defined cases in which LCW information is kept, lost, or modified. Nevertheless, LCW information can get relatively complex so it may be best to use a minimal number of mutexes since it is unclear how useful they will be.

In a world where sensing can produce nearly infinite amounts of knowledge, the agent has to be selective in what to include in the planning graph to keep it tractable. For example, executing `ls` in many directories can produce thousands of files which can all be preconditions for many actions, most likely irrelevant to a given goal. This suggests ideas of abstracting action representations in the graph, or perhaps regressing from the goal to only include potentially relevant actions.

One method to deal with the problem of infinite sensing and interleaved planning with execution is to learn distributions about how likely a sensing action will lead to a goal. This means that as the agent executes sensing actions, it can learn to adjust weights of those actions depending on the success of bindings. This can affect what to include in a planning graph and the weights to links from sensing actions.

Example

To illustrate some of these ideas we consider a simple example. Suppose an agent in the Unix domain starts in the root directory with no other information (*i.e.*, $M = \{$`dir(root)`, `curDir(root)`$\}$). The goal is to find a pdf file (*i.e.*, a file named *.pdf). A planning graph in figure 2 illustrates this initial situation in the first state level. Note the effects of the only action, `ls(root)`. The agent will certainly gain LCW information about what is in the root directory, but it will not know until execution whether it will discover any files or directories, and what their names and types will be. Therefore **f* represents any number of files with potentially any name (**f* is a generic type), and is linked with a dashed line. This means that at the next action level, it will be possible to add an action such as `mv(core, newfile.pdf)` by binding *core* to **f*, which would achieve the goal. Even more direct would be to bind **f* to *somefile.pdf* immediately achieving the goal. Of course a plan is not complete until successfully executed.

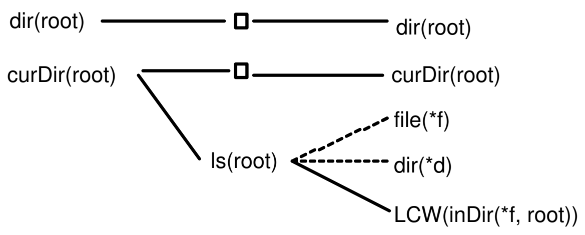

Figure 2: The agent starts with M = dir(root), curDir(root). The only action the agent can execute is `ls(root)`. Notice how the links to `file(*f)` and `dir(*d)` are dashed lines indicating the results of a sensing action.

Suppose the agent executes `ls(root)` and discovers a file, *paper.tex*, and a directory, *papers*, but no **.pdf* file. Then the agent's knowledge increases to approximately what is shown at the first level of figure 3. Note that `ls(root)` is removed from the set of actions since it is irrelevant given the LCW information. Two new actions have been added to the first action level. At the second state level, the goal is not available, and there are no potential bindings that will make it true. In the second level we see two actions, one that has the effect, `file(paper.pdf)`, and the other that has the effect, `file(*f)`, both of which can satisfy the goal. However, one requires an arbitrary binding, while the other does not. Furthermore it is easy to see from the planning graph that creating *paper.pdf* is unlikely to require any sensing actions, which might result in a partial plan choosing to add the action, `dvipdf(paper.dvi)` over `ls(papers)`.

Extracting Heuristics

The extended planning graph will provide an agent with a variety of heuristics to guide planning and control execution. These include the following:

1. Reachability of goals
2. Estimate of sensing

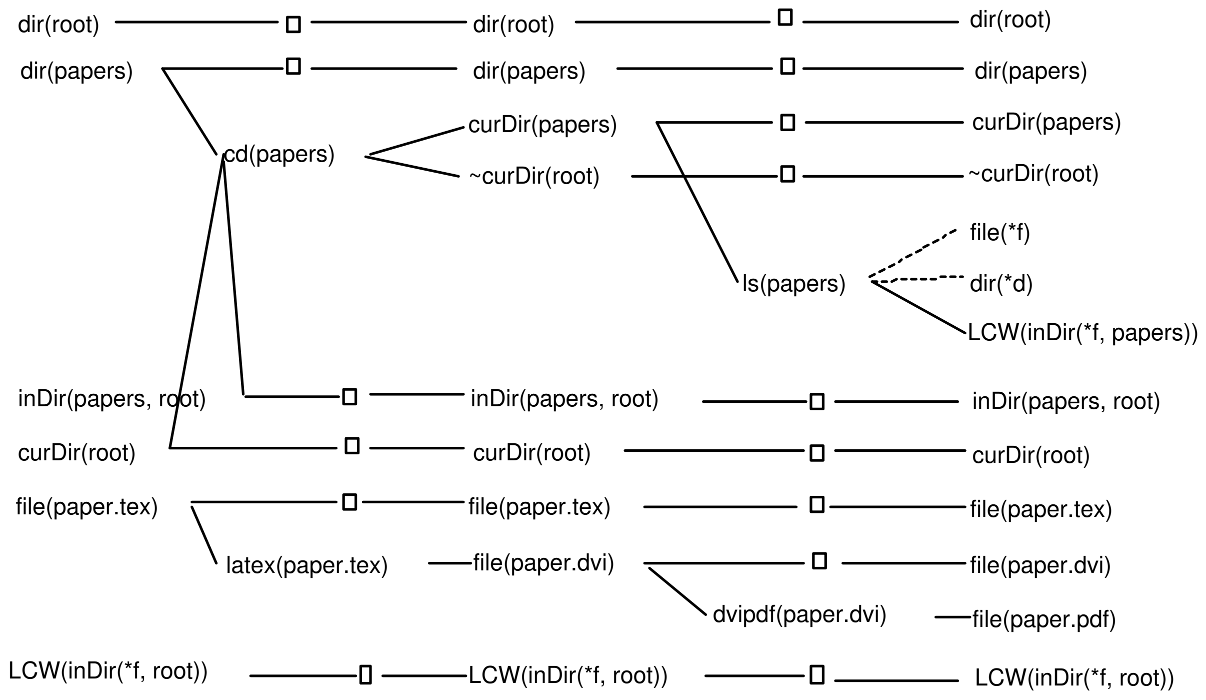

Figure 3: The agent's knowledge expands after executing `ls(root)` and we see how the planning graph is updated.

3. RePOP heuristic (cost of open conditions)
4. LPG-like local search

A planning graph will show the first level at which a proposition appears. This can be quite useful in deciding the difficulty of obtaining such a goal, or if it is even possible to reach a goal state which contains all the goals. An agent can use this information to decide when to focus on constructing a complete plan, or when to execute some actions to get to a better search space.

Apart from finding the first level at which a proposition appears, the agent can also get an estimate of the difficulty of reaching that proposition by seeing how many sensing actions are required. This can guide the agent in choosing some actions over others when working with a partial plan, or preferring one state over another while searching through state space.

As the agent executes sensing actions and updates the planning graph, it can learn how effective some sensing actions are in improving the planning graph, making it more likely to reach the goal. This may give an estimate of when the agent should focus on sensing (and specifically which type of sensing actions) to reach a better proposition level in the planning graph, or how to contract the graph easing future planning.

A more concrete heuristic borrowed from the partial-order planner RePOP involves looking at the set of subgoals in a partial plan. The planning graph can be used to recursively estimate the cost of reaching those goals by seeing at which levels in the graph they are obtainable. By propagating sensing actions and links, the agent may be able to get a more accurate measure of the difficulty of varying goal sets.

Although the proposed planning graph is much less structured than the traditional planning graph due to incomplete and unbounded information, it may be possible to borrow the ideas from LPG and do a local search on the planning graph to find either a complete plan that is expected to reach the goal state, or a subplan that is expected to lead to promising modifications to the graph.

Conclusions

We began by discussing the problem of planning in domains with incomplete information and unbounded sensing. Planners such as the one used for the Internet Softbot have had limited success, showing that in principle there exist techniques that will work, but that in practice they do not scale well and are not domain independent.

We propose methods that an agent can use to extract heuristics from the planning domain by constructing a planning graph and borrowing many of the already existing ideas as well as introducing some new ones. We outline how to make additions to the planning graphs to include LCW and sensing actions, and the need for a dynamic planning graph that learns and quickly updates as the agent interleaves planning with execution.

We conclude by suggesting some specific heuristics the agent can find in the planning graph. These include applying already existing methods such as those used by RePOP and LPG, as well as some new ones that involve learning and deciding when to plan or execute which are specific to domains with incomplete information. We are working on the formalization and implementation of these ideas, and hope to soon demonstrate their effectiveness in detailed experiments.

References

Ankolenkar, A.; Burstein, M.; Hobbs, J.; Lassila, O.; Martin, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Payne, T.; Sycara, K.; and Zeng, H. 2001. DAML-S: A semantic markup language for web services. In *Proceedings of the Semantic Web Working Symposium*, 411–430.

Bacchus, F., and Petrick, R. 1998. Modeling an agent's incomplete knowledge during planning and execution. In Cohn, A. G.; Schubert, L.; and Shapiro, S. C., eds., *KR'98: Principles of Knowledge Representation and Reasoning*. San Francisco, California: Morgan Kaufmann. 432–443.

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *ECP*, 360–372.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2).

Burstein, M. H.; Hobbs J. R.; Lassila, O.; Martin, D.; McDermott, D. V.; McIlraith, S. A.; Narayanan, S.; Paolucci, M.; Payne, T. R.; and Sycara, K. P. 2002. Daml-s: Web service description for the semantic web. In *International Semantic Web Conference*, 348–363.

Do, M., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *ECP*.

Etzioni, O., and Weld, D. 1994. A softbot-based interface to the Internet. *C. ACM* 37(7):72–6.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed-world reasoning with updates. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, 178–189.

Etzioni, O.; Golden, K.; and Weld, D. 1997. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence* 89(1–2):113–148.

Gerevini, A., and Serina, I. 2002. Lpg: a planner based on local search for planning graphs. In *Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02)*.

Golden, K., and Weld, D. 1996. Representing sensing actions: The middle ground revisited. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 174–185.

Golden, K.; Etzioni, O.; and Weld, D. 1994. Omnipotence without omniscience: Sensor management in planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1048–1054. Menlo Park, CA: AAAI Press.

Golden, K. 1997. Planning and knowledge representation for softbots.

Golden, K. 1998. Leap before you look: Information gathering in the PUCCINI planner. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 70–77. Menlo Park, Calif.: AAAI Press.

Koenig, S. 2001. Agent-centered search.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.

McDermott, D. 2002. Estimated-regression planning for interactions with web services. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, 249–257.

McIlraith, S.; Son, T.; and Zeng, H. 2001. Semantic web services. *IEEE Intelligent Systems (Special Issue on the Semantic Web)* 16(2):46–53.

Nguyen, X., and Kambhampati, S. 2000. Extracting effective and admissible state space heuristics from the planning graph. In *AAAI/IAAI*, 798–805.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *IJCAI*, 459–466.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. S. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence* 135(1-2):73–123.

Peot, M., and Smith, D. 1992. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 189–197. San Francisco, Calif.: Morgan Kaufmann.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *AI Planning and Scheduling (AIPS2002)*.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision based approach. *J. Artificial Intelligence Research* 4:287–339.

Reiter, R. 1978. On closed world databases. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. Plenum Press. 55–76.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 897–904. Menlo Park, CA: AAAI Press.

Weld, D. 1996. Planning-based control of software agents. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*.