GTD-POP: Bridging the Gap between Soundness and Efficiency in Practical Planners

Daniel M. Gaines
dmgaines@cs.uiuc.edu
Computer Science Department
University of Illinois at Urbana-Champaign

Abstract

This paper presents GTD-POP, a planning methodology based on the Generate, Test and Debug paradigm. GTD-POP's goal is to achieve a compromise between planning efficiency and soundness by using associational knowledge to guide its search for interaction checks in a partially-ordered plan. This paper describes the GTD-POP and discusses issues involved in evaluating the behavior of practical planners.

Introduction

As AI planners leave the realm of toy domains and are brought to bear on complex planning problems, limitations of existing formalisms have become apparent. Designing a domain-independent, sound planner which can work efficiently in complex domains is an infeasible goal. Previous planners have given priority to a few of these characteristics at the expense of others. This paper proposes a model of planning which provides a compromise between soundness and efficiency, allowing a tradeoff between these properties to meet the needs of a given problem.

The proposed framework, GTD-POP (Generate, Test and Debug - Partial Order Planner), uses a combination of heuristic and causal domain knowledge to construct partially-ordered plans for practical problems. GTD-POP follows the philosophy of GORDIUS (Simmons 1988), which presented the Generate, Test and Debug paradigm as a means of incorporating associational and causal knowledge to create an efficient and robust planner. GTD-POP extends these ideas by allowing this knowledge to direct the search for interactions in a partially-ordered plan. GTD-POP has applications for human-computer collaborative problems, by allowing the user to suggest areas of the plan to explore more deeply.

The paper begins by outlining some metrics for evaluating the behavior of a planner. These metrics describe tradeoffs to be made when designing a planning system. Related work is described in terms of the choices made with respect to these metrics. Next, the GTD-POP model is described in followed by a discussion on how the performance of this type of system can be evaluated.

Characterizing the Behavior of a Planner

This section presents 5 metrics for describing the behavior of a planner. The list is not intended as a complete characterization of planners, but to provide some useful terminology for discussing the tradeoffs involved in moving AI planners to real world domains.

Completeness: Given a problem to be solved, a complete planner will generate a solution to the problem, if one exists, or determine that no solution is possible.

Soundness: Any plan created to solve a problem by a sound planner is a correct solution.

Generality: The generality of a planner refers to the family of domains in which it can solve problems. A general planner can work in a large number of domains, whereas, a less general planner may be applicable to only a small family of closely related domains.

Expressiveness: The expressiveness of a planner refers to its ability to represent the intricacies of the dynamics of a given domain. For example, an expressive planner would be able to reason about actions with context-dependent effects.

Efficiency: The efficiency of a planner can be described in terms of how long it takes to produce a plan relative to the number of steps in the plan. A planner which takes an exponential or greater amount of time with respect to the number of plan steps is considered inefficient.

Ideally we would like a complete and sound planner with an expressive representation and capable of planning efficiently in a large variety of domains. However, this is not reasonable. These characteristics describe a set of tradeoffs to be made in designing a planner. One must decide which of these properties should be relaxed so that others can be achieved.

Chapman has already shown that a complete domain-independent planner is unrealistic (Chapman 1987). How far should the scope of the planner be restricted in order to obtain a complete planner? Decidability can only be obtained in a trivial domain. Semi-decidability may be more likely, but at what cost to generality and expressiveness?

Clearly, it is desirable that our planner produce correct plans, but which of these properties are we willing to sacrifice to achieve it? Does it make sense to have a sound but inefficient planner? How useful will such a system be on large-scale planning tasks? As expressiveness is gained, it becomes more expensive to produce sound plans. An increase in planning efficiency often comes with the price of decreased generality.

Related Work

This section briefly summarizes related planners in terms of the tradeoffs made with respect to the metrics of the previous section.

Early total-order planners, such as STRIPS (Fikes & Nilsson 1971) and HACKER (Sussman 1974) were designed to be general and sound, but were not very expressive or efficient as the search space of total-order planners is very large.

Sacerdoti's NOAH (Sacerdoti 1975), provided a representation of plans in the form of partially-ordered plan steps which drastically reduced the size of a planner's search space, but poses a challenge for efficiently reasoning about these plans. In order to efficiently detect interactions, both NOAH and TWEAK (Chapman 1987) restrict their representation to context-independent actions. Thus, these planners gain soundness, generality and efficiency, but are not expressive enough to handle complex domains.

Pednault presents a truth criterion which is capable of detecting interactions in an expressive, sound planner (Pednault 1991). *Secondary preconditions* are added to an action which describe the situation in which the action will have the intended effect. However, as Pednault points out, a single set of secondary preconditions cannot, in general, cover all possible execution sequences in a partially-ordered plan. For these cases, the plan is "decomposed" into multiple alternative plans. In the worst case, this will result in the same search space as in total-order planning.

SIPE (Wilkins 1984; Wilkins 1988) extends the expressiveness of STRIPS-style operator descriptions to include operators with conditional effects in order to account for practical problems. As Chapman, (Chapman 1987), points out, SIPE's treatment of conditional effects is not sound in general. Furthermore, to achieve efficiency, SIPE does not fully exploit the expressiveness offered by its representation of conditional effects.

GORDIUS (Simmons 1988) presents the Generate, Test and Debug (GTD) paradigm for planning and uses an interesting combination of associational and causal knowledge for achieving efficiency and robustness. A total-ordering is selected from a plan as the hypothesis for the final solution and tested and debugged with causal knowledge.

The proposed model of planning presented in the following section, continues along the lines of SIPE and GORDIUS. However, instead of strictly preferring efficiency over soundness, or vice versa, the framework provides a tradeoff between these two metrics, allowing the degree to which efficiency is preferred to soundness to be adjusted.

GTD-POP: A Compromise between Soundness and Efficiency

GTD-POP (Generate, Test, Debug - Partial Order Planner) is intended for use in planning domains in which there are complex interactions among plan actions. In domains such as these, a large amount of the planning time is spent in checking for interactions among plan steps. Testing for interactions is not necessarily a matter of matching effects of one actions with preconditions of another. Actions may have conditional effects and extensive computational effort may be required to detect interferences. A simulation with a totally ordered action sequence may be needed. The goal of GTD-POP is to use associational knowledge, derived from causal relations among domain objects, to guide the search for interactions, and reduce the number of interactions actually checked.

GTD-POP uses domain knowledge to guide the search for interactions among plan steps in a partially-ordered plan. Borrowing ideas from GORDIUS, GTD-POP makes use of a combination of associational and causal domain knowledge. Associational knowledge is used to suggest areas of the plan in which a search for interactions would be beneficial and to generate sequences of actions to be tested, causal knowledge is used to simulate these actions and debug the results.

Figure 1 shows the architecture of the system. A partially-ordered plan represents multiple possible action sequences for achieving goals. Because the effects of actions may be dependent upon the situation in which they are executed, interaction detection cannot simply check the preconditions of the action against the effects of the other actions, as in the case of the modal truth criterion. Instead, the Generator identifies segments of the partially-ordered plan as potentially interacting. Interesting total-orderings of this segment are selected and passed to the Simulator. Problems with the simulation are detected and resolved by imposing constraints on the partially-ordered plan, by the Debugger.

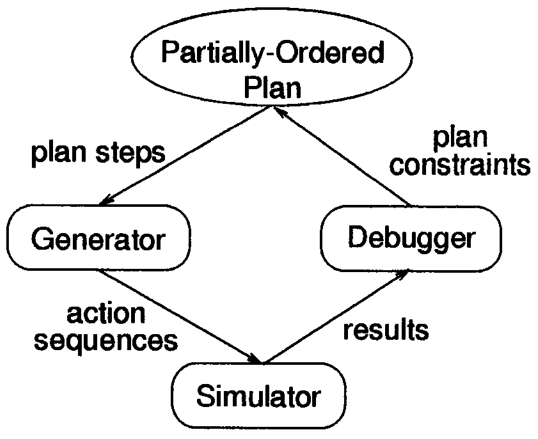

Figure 1: GTD-POP architecture.

Figure 2 describes the algorithm which GTD-POP will use in a bit more detail. The main cycle of plan construction consists of selecting a subgoal to achieve and an action for achieving it.

1. Select a subgoal, g, to achieve.
2. Choose an action, a, to achieve it.
3. Test plan soundness:
   (a) Identify candidate actions, $a_1 \ldots a_n$, possibly interacting with a.
   (b) *Generate* total orderings of actions.
   (c) *Test* these total orderings through simulation.
   (d) *Debug* any problems with results by constraining the actions.

Figure 2: GTD-POP algorithm.

Next, the planner must check to see if the newly added action interacts with other steps in the plan. To be completely sound, the planner would need to check the new action in every possible totally-ordered sequence represented by the partially-ordered plan. Unfortunately, this is computationally prohibitive as there are, in general, an exponential number of total orderings in a partially-ordered plan.

To avoid an exhaustive search for interactions, GTD-POP makes use of heuristic knowledge about how objects in the domain are likely to interact. Each action in the plan describes the use of domain objects for achieving goals. The Generator uses heuristics based on the way objects are likely to interact with one another to identify other actions in the plan which might cause an interaction with the newly added action. A set of total orderings are selected to test for these possible interactions. The architecture allows for user input in these two steps. If the user feels that some portion of the plan merits deeper investigation, this portion can be pointed out to the Generator.

In summary, the idea is to gain efficiency by only considering a limited set of totally-ordered action sequences. Associational domain knowledge is used in step 3a as an inexpensive means of identifying likely candidates for interactions and in step 3b to create interesting total orderings of the plan. Causal knowledge is used to simulate the total orderings and debug any problems. The user will be able to control the extent to which efficiency is preferred over soundness. This makes GTD-POP an effective solution for human-computer collaborative tasks.

GTD-POP and Planning Efficiency

Efficiency in GTD-POP is gained by using heuristic knowledge about how objects in the domain are likely to interact to avoid checks for interactions in a plan which seem unlikely to cause problems. This efficiency comes at the cost of the plan's soundness. The planner may overlook interactions which will lead to problems when a certain total ordering of the plan is carried out.

However, the degree to which soundness is preferred over efficiency can be controlled with steps 3a and 3b in Figure 2. By considering interactions among more of the planning actions, step 3a, or more of the total orderings in the plan, step 3b, the soundness of the plan will increase as the efficiency of the planner decreases.

This approach has many similarities to the incremental reasoning technique proposed by Chien (Chien 1991). In his thesis, Chien suggests overlooking conditional interaction checks until more time is available to the planner. Chien has shown that incremental reasoning will converge on a sound plan.

There are some interesting distinctions between GTD-POP and incremental reasoning. First, incremental reasoning requires a sound and complete domain theory. The intended use of GTD-POP is in domains in which it is not easy, or even possible, to construct a sound and complete domain theory. In this case, the best GTD-POP could do is achieve the soundness of the domain as described by the represented domain theory.

A further distinction between these approaches is GTD-POP's use of domain knowledge to guide the search for likely interactions. This knowledge allows GTD-POP to focus its search for interactions on areas of the plan which are most likely to cause troubles. This has the potential of making more effective use of planning time.

Discussion

As has been discussed, insisting on a complete and sound planner able to perform efficiently in a large number of domains is unrealistic. Some of these characteristics will need to be relaxed so that others can be achieved. A planning model has been described which allows a tradeoff between two of these properties: efficiency and soundness.

This raises the question about the appropriate means of evaluating the performance of this type of planner. Describing a planner as sound or unsound is too coarse of a discritization. It is necessary to give up some amount of soundness in order to perform efficiently in a practical domain. However, labeling such planner's as unsound does not adequately describe its planning behavior. Perhaps, for the majority of planning problems it sees, the planner produces reliable plans.

A theory of planning analogous to PAC (Probably Approximately Correct) learning (Valiant 1984) may be in order. With PAC planning one could refer to the expected correctness of a solution in relation to how much time was spent in planning. For example, one could say something about the expected correctness of plans a system could produce in polynomial time, or the soundness of a plan with a polynomial number of interaction checks with respect to the number of steps in the plan.

This brings up the need for a means of describe the soundness of a plan. The classification of a plan as either correct or incorrect is, again, too coarse. A plan which is close to a correct solution may still be useful, for example as a means of communication (Agre & Chapman 1990). A way to describe the "closeness" of a plan to a correct solution would provide a more representative evaluations.

Conclusion

This paper has proposed GTD-POP, a partial order planner which uses the Generate, Test and Debug paradigm to apply domain knowledge to the construction of plans for complex domains. Rather than strictly preferring soundness to efficiency, GTD-POP uses a combination of heuristic and causal knowledge to achieve a compromise between these properties. GTD-POP provides the ability for a user to direct the application of causal reasoning as needed for a particular problem.

This paper has discussed how the GTD-POP architecture will use domain knowledge to gain efficiency in planning. Future work will concentrate on the exact form this knowledge will take for a particular domain. This work will include developing an implementation of GTD-POP in the domain of manufacturing planning. Ideas on PAC planning will be explored to develop a formalism for evaluating the performance of practical planners.

Acknowledgments

Discussions with Caroline Hayes provided helpful comments for this paper.

References

[1] Agre, P. E., and Chapman, D. 1990. What are plans for? *Robotics and Autonomous Systems* 6:17–34.

[2] Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32.

[3] Chien, S. 1991. *An Explanation-Based Learning Approach to Incremental Planning.* Ph.D. Dissertation, University of Illinois at Urbana-Champaign.

[4] Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5(2).

[5] Pednault, E. P. D. 1991. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *IJCA 91*, 240–245.

[6] Sacerdoti, E. D. 1975. The nonlinear nature of plans. In *IJCAI 1975*, 206–214.

[7] Simmons, R. G. 1988. Using associational and causal reasoning to acheive efficiency and robustness in problem solving. In *IMACS Transactions on Scientific Computing.*

[8] Sussman, G. J. 1974. The virtuous nature of bugs. In *The First Conference of the Society for the Study of AI and the Simulation of Behaviour.*

[9] Valiant, L. 1984. A theory of the learnable. *Communications of the Association for Computing Machinery* 27:1134–1142.

[10] Wilkins, D. E. 1984. Domain-independent planning: Representation and plan generation. *Artificial Intelligence* 22(3).

[11] Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Paradigm.* Morgan Kaufman.