Adapting Research Planners for Applications

Robert P. Goldman
SIFT, LLC
2119 Oliver Avenue South
Minneapolis, MN 55405
rpgoldman@sift.info

Introduction

I am currently working to adapt a "theoretical" AI planner, SHOP2 (Nau *et al.* 2001a; 2001b) to an applied domain: control of teams of autonomous aerial vehicles (UAVs). This project provides a very particular applications-oriented viewpoint on the relationship between AI planning research and applications. One can readily see the gaps (and bridges) between research and applications when trying to bring the fruits of research to an application. Let me emphasize that my remarks here will be very applications-oriented (although I will remark on some needs for theory), and that directly using research software is far from the only way that research results can reach applications!

Our research aims at using planning systems as a component in larger intelligent control systems. In previous work, my colleagues and I have argued that planners can provide a bridge between intuitively understandable human task models and complex control systems (Miller & Goldman 1997; Goldman *et al.* 2000). We can make complex automated control systems more usable if we can align their use with standard procedures that are familiar to users. For example, it will be easier to handle automated aerial vehicles if one can interact with them as agents performing reconnaissance missions that consist of ingress, observation, and egress steps, instead of interacting directly with a mathematical optimization system for route planning.

We build "playbook" interface systems, where users can call familiar "plays" or procedures, for automation to perform. A planning system will stand between the user and the control automation, translating these plays into sequences of primitive actions that can be executed by the control system.

In this brief position paper, I review the pragmatic advantages and disadvantages of employing an AI planner from the research community. I also suggest some areas for planning research that would be helpful for control applications that use planners.

The Advantages of research planners

There are a number of advantages to adopting a planner developed in the research community. For one thing, to paraphrase the old joke, they have the immense advantage of being *there*. The AI planning competition (Bacchus 2001; McDermott 2000) led to the development of a great many high-performance planning systems that are readily available. I have been using SHOP2 (`http://www.cs.umd.edu/projects/shop/`), which is available under the MPL or LGPL open source licenses. In contrast, the most prominent and mature applications-oriented AI planners, Sipe (Wilkins 1988), and O-Plan (Currie & Tate 1991), are not as readily available to anyone who wishes to use them. O-Plan and its successor I-X are freely available for academic, research, and evaluation purposes (`http://www.aiai.ed.ac.uk/project/oplan/` and `http://www.aiai.ed.ac.uk/project/ix/`).

A second great advantage of research planners is that they are smaller, and thus easier to integrate as a component of a larger system. To contrast such systems with Sipe and O-Plan once again, one can readily imagine incorporating them as a small component of a larger application; Sipe and O-Plan on the other hand, are more likely to be applied as environments within which to solve the totality of a problem.

Finally, when integrating a research planner into a larger application, one generally has the advantage of a crisply-defined and comprehensible algorithm, often published and peer-reviewed. It should not be necessary to argue for such algorithms to this community, but it is worth stressing that such algorithms, and the code for them, are hard to come by.

Disadvantages of research systems

There are also some pragmatic disadvantages to adopting research software. Probably the greatest practical disadvantage is the lack of domain engineering support. Another problem is that many research planning systems have adopted the PDDL language(Ghallab *et al.* 1998) for their plan libraries, and this language has very limited expressive power.

It is a dirty secret that although planners are supposed to allow us to separate the goal from the means, and provide easy ways to command intelligent systems, plan libraries are very difficult to engineer. At least some of this difficulty can be chalked up to the absence of support for debugging and correctness-checking. For that matter, based on my experience, it is not clear that even larger, more ambitious, planning systems have the domain engineering problem licked.

In the best of all worlds, I would like a planner that provided static and run-time correctness checking, perhaps along the lines of the very sophisticated type-checking offered by the ML family of languages. I would *not* suggest the development of elaborate GUI-based tools for engineering domains; those would actually make such planners more difficult to fit into larger domains, rather than easier. McCluskey's group has done some interesting work in this area, albeit with an IDE-style approach (Simpson, McCluskey, & Liu 2003). I would like to see how adaptable this work is to other planning systems.

I am conscious that developing domain-engineering tools is not an appropriate job for the community that developed planners for the planning competition. For the programmers/users of those competition systems, the effort expended to develop domain engineering tools is extremely unlikely ever to be successfully amortized across the limited number of domains they build — especially since their most important domains may be received from the competition organizers, rather than engineered!

There is some hope that ontologies and their tool support might be able to help with domain engineering. However, right now the modeling approaches of ontology researchers and planners do not harmonize terribly well. Furthermore, tool support for ontology languages like OWL is not yet good enough to encourage planning researchers to adopt it. This may change in the relatively near future.

I will not have much to say about the disadvantages of PDDL from the standpoint of planning applications. Rather, I direct the interested reader to last year's workshop paper by Frank, et. al.(Frank, Golden, & Jónsson 2003). They discuss a number of difficulties that PDDL presents to work like their own, which is on the boundary of planning and scheduling. Significant work is being done to extend PDDL to permit systems to reason about domains that evolve in time and even systems that evolve continuously (Fox & Long 2002; McDermott 2003), but it is still difficult to build plans for application domains out of PDDL operators. For our purposes, one of the greatest difficulties is that PDDL is a language only for first-principles planners, rather than hierarchical planners.

Theoretical desiderata

Based on problems I have encountered in interactive planning for complex control systems, here are some places where planning research could be of direct assistance. First, the execution semantics of existing plan languages does not provide enough guidance when translating plans into executing procedures. This is particularly true of the semantics of HTN planners, which despite their longevity in applied work, and a renaissance in research, are still mysterious beasts. With such open semantic issues, it should be no surprise that plan execution is problematic. Particularly troublesome is the execution of plans in dynamic environments. The forward planning approach that has dominated the planning competition may not be appropriate for more ambitious problems encountered in applications. Finally, we would like to use planning systems for more than just plan generation, so theories of plan management would be helpful.

HTN semantics

Researchers at the University of Maryland have done a great deal to clarify the meaning of plans generated by HTN planners(Erol, Hendler, & Nau 1994). However, we still have a distance to go to come up with theories that can elucidate the power of such systems. Existing semantic theories seem to miss some of the key strengths of HTN plans. When they do attempt to elucidate the meanings of method definitions, they seem tautological.

One problem with the existing semantics is that they seem to have been led by efforts like the planning competition to avoid key features of HTNs. The existing semantics primarily view HTNs as a way to control the search in the space of sets of STRIPS-like primitive operators. However, part of the reason for the appeal of HTN planners is that they allow us to specify tasks in ways that go beyond the semantics of STRIPS-like operators.

HTNs are useful for handling situations where primitive operators miss important features of the domain. One might use HTNs to capture procedures in domains where we do not have good causal theories. In most human organizations there are standard protocols to be followed for complex tasks. In complex domains like medicine there are actions which are known to be useful on empirical grounds, but for which we do not have good causal theories. HTNS can also be used to build robust plans without explicit use of conditionals. The HTNs can contain extraneous actions that are not useful if the plan proceeds smoothly, but that can help forestall bad eventualities or prepare the agent to meet them. It may be much easier to simply require an agent to bring spare parts than to generate a conditional plan that elucidates the reason for doing so.

One of the most interesting uses of HTNs, which we see in systems like Sipe(Wilkins 1988), is to plan in multiple abstraction spaces. For example, the planner could generate a plan to move between locations first by planning at a gross city-to-city level, then within a city, and then within a neighborhood(McDermott & Davis 1984). This use of decomposition planning is beyond existing semantical approaches, since it relies on refinement in the state representation as well as in task decomposition.

Execution

Our work on UAV control assumes a relatively standard three-level architecture, with a planner at the top level, a reactive system in the middle, and conventional control software at the bottom. This style of architecture is a de facto standard because the intermediate, reactive layer, can serve to keep the plan on track by handling divergences not handled by the control system. Typical control software is not intelligent enough to keep a plan on track, and often is not well enough integrated across subsystems. A key problem with this architectural structure is that it is difficult to be assured that the combination of the reactive and control layers will successfully reject disturbances and keep the plan on track. There are some relevant theoretical tools available in the field of discrete control, but typically the reactive control layer is more complex than can easily be modeled for such analyses. Certainly more could be done in this area.

Replanning, plan change metrics, and plan stability

We know that the plans we generate will not always be executed successfully. In most cases the executing agent will encounter unexpected contingencies. Some of these disturbances can be successfully rejected by the control system or the reactive layer. For example, a UAV might diverge from its flight schedule because of adverse winds, but an autopilot that provided 4D waypoint flying might be able to correct.[1] One could also imagine the unexpected appearance of another aircraft, that might be handled by the reactive layer's invoking an evasive maneuver package and then returning to the normal course. However, there will also be cases that must be handled by replanning. Some adverse winds will be sufficiently strong to make lateness unavoidable, and some obstacles may require radical rerouting. In such cases we must replan.

If our systems are to replan, their users should be able to be confident that they will get new plans that are as similar as possible to the original ones. This is closely related to the control theoretic notion of stability: relatively small disturbances should not lead to arbitrarily large changes in system behavior. In control theory for continuous systems, especially linear systems, such properties can readily be established. However, it is difficult to determine what sort of distance metric one should impose over the largely discrete structures that planners work with. For that matter, it is also difficult to establish a good measure of the size of a disturbance from the environment.

Forward Planning

Forward, "linear" planning by heuristic search has been a dominant approach in the planning competitions. As mentioned elsewhere, SHOP2 takes a similar approach, but with the heuristic guidance provided by hierarchical decomposition. The SHOP planners take this approach not simply for reasons of speed, but also because forward planning permits planning with the aid of very rich ancillary problem solvers. For example, a forward planner could ask a CAD system to project the consequence of some machining action from a known start point, but it would be very difficult to describe machining actions in conventional STRIPS-like terms.[2]

The forward planning approach is likely to present growing difficulties in domains where scheduling and resource management are of the essence. Consider a very simple example: There is an area to be continuously "staffed" by a set of robots for some period of time. Let us call this the ROBOT-STAFF task. To staff the area a robot must move to it, perform some service actions while remaining there, and return home. All of these actions consume some "zorch," and the robot must return home before its zorch is exhausted. Consider what might happen with a naive forward planner: it sends a first robot to the area, which arrives there and performs some service actions. In some way the planner identifies that the first robot must leave the area and return to base before it exhausts its zorch. Now a forward planner faces a difficulty: by the time it has projected far enough forward to identify the need to dispatch a second robot to replace the first one, it is beyond the point where the second robot can be dispatched.[3]

Note that this problem presents a grave challenge even to a forward HTN planner like SHOP2. An obvious approach to planning for this problem would be to provide two alternative ROBOT-STAFF methods. The simple method would be one that puts a robot on duty for the entire period, let's call it ONE-ROBOT-STAFF. The alternative method, MULTI-ROBOT-STAFF, would do a ONE-ROBOT-STAFF, and then a recursive ROBOT-STAFF that takes care of the staffing for the remainder of the period.

In our case, a SHOP-like planner would first try ONE-ROBOT-STAFF, and then, when it failed, try the MULTI-ROBOT-STAFF. Unfortunately, when backtracking, it would lose the information about the failed ONE-ROBOT-STAFF it could use in doing the task decomposition of MULTI-ROBOT-STAFF. In particular it would lose the knowledge of how long the first robot could stay in the area to be staffed before running out of zorch. The only solution, seemingly, is to provide problem-solving information in the preconditions to MULTI-ROBOT-STAFF. It is hard to avoid the uncomfortable feeling that the rules, functions, etc. that make up these preconditions to MULTI-ROBOT-STAFF end up doing most of the work, with little left over for the HTN proper.

There are two approaches that previous HTN planners have taken to problems like this. One has been to provide some plan critic facility that allows the system to use information from early failures in later backtracking. In this case, a plan critic could use information from the first failed expansion of ONE-ROBOT-STAFF when planning MULTI-ROBOT-STAFF. The danger of this approach is that we end up with an unstructured expert system embodied in the plan critics, with the planning algorithm itself providing us little value.

The second approach is HTN planning in abstraction spaces. A planner would first develop a very sketchy plan at a high level of abstraction. This high-level plan would either establish that ONE-ROBOT-STAFF was adequate to the demands of the task, or decompose the task adequately to ensure that a MULTI-ROBOT-STAFF would be able to dispatch the second robot in time to take over from the first. Challenges to this approach include the inverse of the advantage of SHOP's forward-planning: how do we reconcile abstraction planning with the requirements of very intricate domains, such as the part machining example we gave above? Also, what sort of guarantees can we provide that the abstraction approach will not make unacceptable completeness sacrifices?

Plan management

Drew McDermott has written that:

> Most applications of practical planning algorithms such as Sipe and Oplan [*sic*] have made good use of the plan management capacities of these systems but not much use of their plan search capacities. (McDermott 1999)

Indeed, our users would most like our planner to fill in "uninteresting details" of their plans, and check their plans to

[1]By 4D waypoint flying we mean the ability to direct an aircraft to fly a series of waypoints that are specified in terms of spatial (3D) location and time. Such a system would control the vehicle's airspeed to compensate for disturbances such as adverse winds.

[2]Dana Nau, personal communication.

[3]This problem here is closely related to the problem of "prepositioning," which we have discussed in a previous paper (Musliner & Goldman 1997).

verify that they are correct. For example, they would like to see the planner generate appropriate waypoint sequences for the UAVs' autopilots. Similarly, they would like the planner to check that a planned mission will not cause controlled flight into terrain, fly into forbidden airspace, or run the aircraft out of fuel. It would be very helpful if one could develop a theory that would help us answer the question: "what is a good extension of this user's partial plan?"

Similarly, we would like to have good algorithms for generating terse, understandable explanations of planning failures. If a user specifies a high-level task that is overconstrained from the standpoint of the planner, we would like to be able to guide him/her to a better solution. Indeed, such techniques would also be valuable to purely automated plan generation, since they could guide variable choice in backtracking.

Summary

In this position paper, I have discussed issues in the overlap between planning research and practice arising in planning for complex autonomous systems. I have talked about the pragmatic tradeoffs in using existing research systems, and discussed some possible advances in planning theory and methods.

Acknowledgments

Thanks to the anonymous reviewer for several helpful suggestions.

References

Bacchus, F. 2001. AIPS'00 planning competition. *AI Magazine* 22(1):47–56.

Currie, K., and Tate, A. 1991. O-Plan: the open planning architecture. *Artificial Intelligence* 52:49–86.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1123–1128. Menlo Park, CA: AAAI Press/MIT Press.

Fox, M., and Long, D. 2002. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*. `http://www.dur.ac.uk/computer.science/research/stanstuff/html/dpgpublic%ations.html`.

Frank, J.; Golden, K.; and Jónsson, A. 2003. The Loyal Opposition Comments on Plan Domain Description Languages . In *Proceedings of ICAPS'03 Workshop on PDDL*.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language.

Goldman, R. P.; Haigh, K. Z.; Musliner, D. J.; and Pelican, M. J. S. 2000. MACBeth: A multi-agent constraint-based planner. In Nareyek, A., ed., *Constraints and Artificial Intelligence Planning: Papers from the AAAI Workshop*, number WS-00-02 in AAAI Technical Report, 11–17. Proc. National Conf. on Artificial Intelligence.

McDermott, D. V., and Davis, E. 1984. Planning routes through uncertain territory. *Artificial Intelligence* 22:107–156.

McDermott, D. V. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence* 109(1 – 2):111–159.

McDermott, D. V. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

McDermott, D. V. 2003. The Formal Semantics of Processes in PDDL. In *Proceedings of ICAPS'03 Workshop on PDDL*.

Miller, C., and Goldman, R. P. 1997. "Tasking" interfaces; associates that know who's the boss. In *Proceedings of the Fourth USAF/RAF/GAF Conference on Human/Electronic Crewmembers*.

Musliner, D. J., and Goldman, R. P. 1997. CIRCA and the Cassini Saturn orbit insertion: Solving a prepositioning problem. In *Working Notes of the NASA Workshop on Planning and Scheduling for Space*.

Nau, D.; Cao, Y.; Lotem, A.; and noz Avila, H. M. 2001a. The Shop planning system. *AI Magazine* 22(1):91–94.

Nau, D.; Muñoz-Avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001b. Total-Order planning with partially ordered subtasks. In Nebel, B., ed., *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 425–430. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Simpson, R. M.; McCluskey, T. L.; and Liu, D. 2003. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *Printed Notes of ICAPS'03 System Demos*. Trento, Italy.

Wilkins, D. 1988. *Practical Planning*. Morgan Kaufmann Publishers, Inc.