Durative Planning in HTNs*

Robert P. Goldman
SIFT, LLC
rpgoldman@sift.info

Abstract

This paper provides techniques for hierarchical task network (HTN) planning with durative actions. HTNs can provide useful heuristic guidance to planners, express goals that cannot be expressed in simple first principles planners, and allow plan generation to be limited by external constraints. The decomposition information in HTN plans can also help guide plan execution. This paper provides a method for encoding PDDL Level 3 durative actions in an HTN formalism compatible with the SHOP2 planner. Efficient planning with these actions requires additional search control. We illustrate the utility of the technique with experimental results on the IPC 2004 AIRPORT domain, and explain how SHOP2 methods were written for this domain.

Introduction

Much of the effort of the planning research community is aimed at the development of "first-principles" planners that assemble action sequences by means-ends reasoning about action pre- and post-conditions. However, for many applications of planning technology, hierarchical task network (HTN) planners have significant advantages. In work on planner-based control of multiple uninhabited (or "unmanned") aerial vehicles (UAVs) (Miller *et al.* 2004), we were led to choose HTN planning for a number of reasons. HTNs permit the programmer of the planning system to constrain the set of plans that can be built, and to introduce steps "just because." These constraints are useful for generating plans that follow existing procedures or policies. Erol, Hendler, and Nau (1994) have shown that HTN planners have more expressive power than first principles planners, because they can capture constraints on the trajectory a plan follows. The hierarchical structure of the plan can give valuable insights, if it is to be understood or (partially) executed by humans, and can be used by interpreters when a plan is to be executed by an autonomous system.

*Thanks to Håkan Younes for supplying with the VHPOP planner, and to Derek Long for the VAL PDDL plan validator. For advice on the airport domain, thanks to Sebastian Trüg, Jörg Hoffman, and Wolfgang Hatzack. For other discussions thanks to Ugur Kuter, J. William Murdock, David J. Musliner, Dana Nau, and the anonymous reviewers.

The UAV control application is like many other systems that use planning: it must reason about temporally-extended ("durative") actions. In work on planner-based control of multiple uninhabited aerial vehicles (UAVs), we have developed a variant of the SHOP2 HTN planner (Nau *et al.* 2003) that plans with durative actions. In this paper, we describe how to model durative actions for use in SHOP2. We illustrate the behavior of the resulting model with tests on the Airport domain from the 2004 International Planning Competition (IPC).

Durative Actions

Durative actions were added to PDDL for the 2002 IPC to create "Level 3 PDDL" (Fox & Long 2003). PDDL durative actions have conditions, rather than preconditions. The domain modeler may specify conditions that must hold at the start of the action (`at start`), at the end of the action (`at end`), and over the duration of the action (`over all`). Similarly, effects may occur either `at start` or `at end` of the action. A key limitation of PDDL durative actions is that their duration is a fixed function of the parameters of the action, thus only a function of the state at the start of the action. The semantics of a PDDL Level 3 plan can be described in terms of a sequence of time points ("happenings"): the start and end points of the durative actions.

Note that there is no "now" variable, or t, in PDDL domains. This limits the kinds of cost functions that can be used to evaluate the quality of PDDL level 3 plans. PDDL metrics include makespan, but that had to be specially added, and it is not possible to evaluate plans on metrics that require integration over time.[1]

SHOP2: Simple Hierarchical Ordered Planner

SHOP2 is a modern HTN planner with a simple, clean implementation, that is easy to adapt for applications, and that has performed well in the IPC (Nau *et al.* 2003). SHOP2 permits easy integration of external problem-solvers and is available under a generous open source license.

SHOP2 and other HTN planners *decompose* complex tasks into primitive tasks, building a plan tree, which terminates at leaves that correspond to primitive actions (*oper-*

[1] Hoffmann, *et al.* (2005) discuss why such metrics would be useful in the Airports domain.

ators). SHOP2 operators provide conditional effects, quantification, and disjunction.

In addition to operators, SHOP2's language provides *methods*, which are complex tasks. A method definition associates a task with a set of preconditions and a task network. When the preconditions are satisfied, a matching task can be decomposed to the given task network. Task networks are sets of tasks that can be executed in any order (`:unordered`) or that may be constrained to be `:ordered` (these may be nested).

SHOP2 also has *axioms*, and *protections*. Axioms allow some literals to be deduced from the sets of primitive literals that make up the states. The programmer of a SHOP2 domain can specify operators that add protections for literals. If an operator application would negate a protected literal, that operator application will fail.

SHOP2 is a forward state-exploring planner; at all times SHOP2 has a notion of the current state. The forward planning nature of SHOP2 makes it amenable to integration with complex external reasoners which may be able to compute state progressions, but not invert state changes.

Modeling durative actions in SHOP2

We now describe how to encode PDDL durative actions as SHOP2 HTNs. Fox and Long (2003) describe the semantics of durative actions in terms of pairs of non-temporal actions and additional constraints. These additional constraints are beyond what atemporal first-principles planners can handle, and must be treated specially. In contrast, because of the additional expressive power offered by methods and protections, HTN planners can directly encode almost the full semantics of durative actions.

Every PDDL durative action will correspond to a SHOP2 method, with the same parameters. That method will decompose into a task network with two or three operators, one for the start of the durative action, and one for its end, and for IPC applications, an additional pseudo-operator will be necessary. The procedure is as follows:

1. Create a method definition with the same signature as the PDDL action, *act*.
2. Add all `at start` conditions as preconditions to the method definition.[2]
3. **Handle action duration:** Add three additional variables, `?start`, `?end`, and `?duration` to the method definition, and additional preconditions binding them:
   (a) `(time ?start)`
   (b) `(assign ?duration <duration-expr>)` — `<duration-expr>` will be supplied with the PDDL action.
   (c) `(assign ?end (+ ?duration ?start)`
4. Create a `!start-`*act* operator, adding the `?start` variable to its arguments.
5. Create an `!end-`*act* operator, adding the `?end` variable to its arguments
6. The task network for the new method will be `(:ordered <!start-op> <!end-op>)`.
7. Add all `at end` conditions as preconditions to the end operator.
8. **`over all` conditions:** `over all` are enforced as follows:
   (a) Add the `over all` conditions as preconditions to the method definition, so that they will hold at the start of the interval.
   (b) For each `over all` condition, the start operator must add a protection.
   (c) Each of the protections added by the start operator must be deleted by the end operator.
9. Manage the `time` fluent:
   (a) Add a `(time ?start)` precondition to the start operator.
   (b) Update the time fluent at the end operator. To do this, we must first check that the time for the end operator has not passed, so we add the following preconditions: `(time ?t)` and `(<= ?t ?end)`. Finally, we add the postcondition: `(time ?end)`.

The above encoding was sufficient for successful temporal planning in our UAV control application. However, some additional steps were necessary for PDDL compliance, because PDDL has somewhat odd continuous time semantics. In particular, the endpoints of producers must fall *strictly* before the start points of consumers. In order to meet this requirement, we add "spacer" pseudo-actions that introduce epsilon advances in time after the end operators of durative actions. This leaves the possibility of pathological interleavings of two start operators, or two end operators, one of which spuriously seems to establish a condition consumed by the other. This is case is so rare and pathological that we have handled it by post-checking with the VAL validator (Howey, Long, & Fox 2004). We have not actually encountered a case of the phenomenon; all plans generated have been validated.

The above method can readily be translated into an algorithm, but we have not performed these translations programmatically, for several reasons. The first is that these translations are not onerous to perform, so the tradeoff in effort for automation is not especially favorable. A second reason is that SHOP2's input language, unlike PDDL, is not typed, so some additional preconditions must be added. Finally, the SHOP2 planner does not automatically order its precondition sets, and poor ordering choices could, in some domains, lead to very poor performance.

MTP: Nau, *et al.* (2003) have proposed an alternative temporal encoding that they call Multi-timeline preprocessing (MTP). In MTP, durative actions are *not* split into begin and end pairs, and protections are not used.Instead, additional read- and write-time fluents are associated with base fluents, and these are manipulated, instead of manipulating a time fluent. MTP gets concurrency through "left packing"

[2]This is more efficient than attaching the preconditions to the start operator (see below) because it allows earlier detection of failure.

operators; our approach gets it by interleaving start and end operators. The discussion of MTP is incomplete; it doesn't handle `at start` effects, axioms, nor does it separate consumers from producers. Without additional precautions it will permit `:ordered` constraints on method task networks to be violated. An advantage of our approach is that the time fluent supports plan metrics that integrate over time.

Search with durative actions

In addition to translating durative actions into HTN representations, one must modify the planner for planning to be feasible with these new actions. Without such modification, SHOP2 will be perfectly happy to choose tasks for expansion without concern for their time of occurrence, and the planner will get lost.

We must prune the set of choices of task to expand. Because of the constraints on durative actions in PDDL, by the time an end-operator is to be added to a plan, that operator's time of occurrence is known. There are three types of tasks to choose among: The first type of task are those tasks whose time of occurrence is rigidly fixed; for Level 3 domains without timed initial literals, these are exclusively end operators.[3] The second type of tasks are those whose time of occurrence floats. These include the start operators, and methods. The third type of task, a sort of hybrid case, is the "spacer" actions, which are not locked to a specific time, so they must be allowed to float, but they must not be allowed to float past the start of their consumers.

We prune by sorting the tasks as follows:

1. Check the list of tasks for feasibility. If there are any tasks with a time of execution that is greater than the current time, then fail and backtrack.
2. If there are any spacer tasks in the list, remove all of the second (unscheduled) operators from the list.
3. Sort the remaining operators by their scheduled time for end tasks, and the end time + ϵ for spacers.

When there are no spacer tasks, but there are both scheduled and unscheduled tasks, one can attempt either the unscheduled or scheduled tasks first. The former may produce plans with lower makespan, but is also more likely to cause backtracking. The latter is more likely to yield a plan quickly, but the plan is likely to be of poorer quality (compare "timed first" and "untimed first" in result plots).

Experiments

To illustrate the performance possible with durative actions in an HTN, we have translated the Airport domain from IPC 2004 (Hoffmann & Edelkamp 2005; Trüg, Hoffmann, & Nebel 2004; Hoffmann *et al.* 2005) using the scheme we have described earlier. The airport domain requires that systems plan ground movements of aircraft at an airport. A number of aircraft must be moved either from a runway to a parking position, or from a parking position to a runway to take off. The problems in this domain vary on two primary axes: the topology of the airport and the number of aircraft to be managed. Airports range from a quite simple airport 1 ("minimal"), through 3 ("toy") and then 4 (half of the Munich Airport) and 5 (the full Munich airport).

We have used the "ADL Temporal" version of the Airport domain, which allows for quantification and conditional effects in the actions, leading to a relatively terse formalization of the domain (approximately 200 lines for the operators). Unfortunately, none of the planners entered in the competition was able to handle the ADL Temporal version of the problem. Several were able to do the STRIPS Temporal version, in which there were only ground operators, albeit at the expense of an explosion in the size of the domain description (e.g., the operators for problem 6, with airport 2, and only two aircraft is 3145 lines; the largest problem, P50, has 42538).

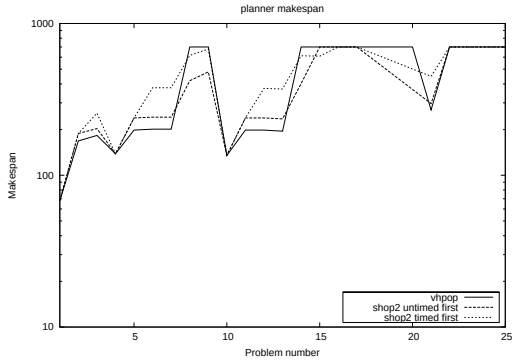

Figure 1: Makespan results for airport domain, airports 1-4.

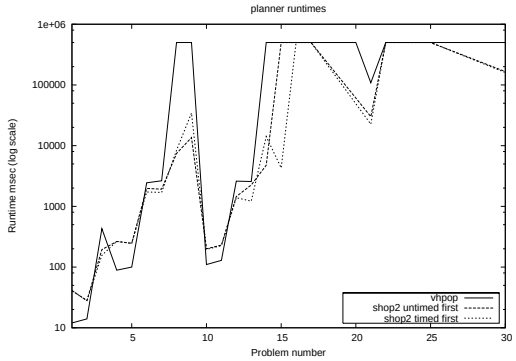

Figure 2: Runtime results for airport domain, airports 1-4.

Results

Figures 1 and 2 show comparative results between SHOP2 and VHPOP and on the domains for airports 1 through 4.[4]

[3] Additional cases arose in our UAV application

[4] Maximum values are pseudo-values added to indicate planner failure/timeout.

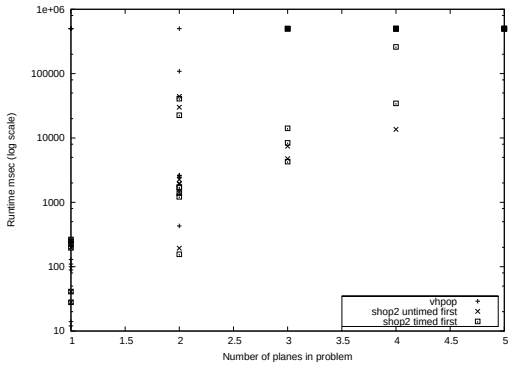

Figure 3: Runtime by number of planes.

These experiments were run on Linux, on a Dell Dimension 8300 with 2 3.0 GHz processors and 2 GB of RAM. VHPOP was compiled using gcc 3.4.3, and SHOP2 using Franz Allegro Common Lisp 7.0. The plans generated were checked for correctness using the VAL program (Howey, Long, & Fox 2004). SHOP2 runs were set to time-out at 300 seconds; VHPOP runs timed out based on the same time limit or search node or memory limitations. SHOP2 provides results that are broadly comparable with VHPOP's; it is somewhat slower, but there are some domains where it finds solutions that VHPOP does not. The dominating factor was the number of planes, rather than the airport, up until Airport 4, when the topology became very complex. We replot the runtimes in Figure 3 sorted by number of planes. This shows that with a moderate number of planes, SHOP2's heuristic methods are very helpful. Makespan results are for the first plan that SHOP2 finds. These are not in general optimal (nor are VHPOP's). SHOP2 can gradually optimize plan quality through branch and bound search; we do not have room to report on our optimization experiments here.

Methods for SHOP2

Our first methods for the Airport domain simply sequenced key phases of the motion. Somewhat to our surprise, these were sufficient for the single aircraft problems. On multiple-aircraft problems, though, SHOP2 simply bogged down.

In these problems, the planner would get into a configuration where one aircraft would be moving into a taxiway from the runway, while another was moving to the runway along the same taxiway. When the planner encounters one of these deadlocks, there is a very large space of bad configurations, reachable by backtracking, in each of which the deadlock would occur at a different point in the taxiway.

To obtain acceptable performance we added methods for moving down a taxiway towards the runway, and moving the other way from the pinch points to the parking location. The first step of these methods is to check the entire path to ensure it is clear, and then use protections to keep other aircraft from entering the path until the first is done. These methods gave the performance illustrated in these graphs. Of course, the deconfliction method outlined above can give suboptimal makespan results. This problem is not acute in the smaller airports, where the bottlenecks are relatively short.

We have also propagated preconditions from the leftmost task(s) of methods upwards towards their parents.[5] This avoids pointless work decomposing tasks to primitives only to find that the first operator cannot be executed.

Conclusions

One interesting challenge in the Airports domain is that, while the HTN planner is good at directing the actions of what is conceptually a single agent, the HTN scheme does not have any means of "looking at" what other agents are doing concurrently. Of course this is an imprecise use of language where there are only operators and fluents, and no real agents. Nevertheless, the intuition should be clear. Dealing with the temporal variables in larger problems is somewhat cumbersome; we expect to add more support for constrained plan variables, and incorporate a simple temporal network solver (Dechter, Meiri, & Pearl 1991) like the one in VHPOP. We have made use of more expressive power in our UAV domain, notably more complex scheduling (including timed initial literals), and time-integrated cost functions.

References

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1–3):61–95.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings AAAI*, 1123–1128. Menlo Park, CA: AAAI Press/MIT Press.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *JAIR* 24:519–579.

Hoffmann, J.; Edelkamp, S.; Englert, R.; dos Santos Liporace, F.; Thiébaux, S.; and Trüg, S. 2005. Towards realistic benchmarks for planning: the domains used in the classical part of IPC-4. Unpublished draft.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings ICTAI*, 294–301. IEEE Computer Society.

Miller, C. A.; Goldman, R. P.; Funk, H. B.; Wu, P.; and Pate, B. 2004. A playbook approach to variable autonomy control: Application for control of multiple, heterogeneous UAVs. In *AHS Forum Proceedings*, 2146–2157. Alexandria, VA: American Helicopter Society.

Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR* 20:379–404.

Trüg, S.; Hoffmann, J.; and Nebel, B. 2004. Applying automatic planning systems to airport ground-traffic control — a feasibility study. Technical Report 199, Insitut für Informatik, Universität Freiburg.

[5] This propagation could be automated, but is not at present.