Priority-Based Playbook™ Tasking for Unmanned System Teams

David J. Musliner, Robert P. Goldman, Josh Hamell, Chris Miller
Smart Information Flow Technologies (SIFT)
Minneapolis, MN 55401
Email: {dmusliner,rpgoldman,jhamell,cmiller}@sift.net

ABSTRACT

We are developing real-time planning and control systems that allow a single human operator to control a team of unmanned aerial vehicles (UAVs). If the operator requests more tasks than can be immediately addressed by the available UAVs, our planning system must choose which goals to try to achieve, and which to postpone for later effort. To make this decision-making easily understandable and controllable, we allow the user to assign *strict priorities* to goals, ensuring that if a goal is assigned the highest priority, the system will use every resource available to try to build a successful plan to achieve that goal. In this paper we show how unique features of the SHOP2 hierarchical task network planner permit an elegant implementation of this priority queue behavior. Although this paper is primarily about the technique itself, rather than SHOP2's performance, we assess the scalability of this priority queue approach and discuss potential directions for improvement, as well as more general forms of meta-control within SHOP2 domains.

I. Introduction

We are developing real-time planning and control systems that allow a single human operator to control a team of unmanned aerial vehicles (UAVs). Our approach, based on the Playbook™ concept of high-level task delegation,[1,2] allows the operator to assign goals to a team of UAVs and then rely on the planning and control system to decide how different UAVs perform different tasks. If the operator requests more tasks than can be immediately addressed by the available UAVs, the system must choose which goals to try to achieve, and which to postpone for later effort. This sort of planning problem, involving choosing which goals to achieve in an over-constrained domain, has been called *over-subscription planning* and also *partial satisfaction planning*.

Partial satisfaction planning can be viewed as a form of meta-control— the challenge is to choose which goals to plan to achieve when resources are limited. The limited resources may be both domain-level resources that are used to actually achieve the goal (*e.g.*, assets, fuel, mission execution time) and deliberation resources that are used by the planning process itself (*e.g.*, memory, planning time). Many approaches to partial satisfaction planning are based on the idea that different goals will be assigned different reward values, primitive actions will have some cost (negative reward), and the objective is to maximize the net reward by carefully choosing the set of goals to be achieved and the overall plan to achieve them.[3]

While this problem formulation is a straightforward simplification of the more general probabilistic domain concept of maximizing expected utility, in some situations it can lead to non-intuitive results. Because the units of reward are interchangeable, it is possible for the planner to decide that achieving several lower-reward goals is preferable to achieving a single higher-reward goal. Thus if a human is assigning reward values to goals, it can be difficult to understand exactly what tradeoffs the planning system might make.[4]

In our domain, UAV control, we need a more clearly-understandable and controllable approach to planning, so that the human operator remains confident that he knows how the UAV team will treat his goals, and how he can adjust the goal specifications to achieve his true intent. So, rather than describing goals with interchangeable reward values, we allow the user to assign *strict priorities*, which have an easily-understood and highly controllable semantics. If a goal is assigned the highest priority, the system will use every resource available to try to build a successful plan to achieve that goal. Only once a plan is available, or it has been proven that no plan is possible, will the system move on to try to plan for the next-lower priority goal. The system will never "trade off" the higher-priority goal to instead achieve a set of lower-priority goals.

This strict priority behavior makes the system easily understood and controlled— if the system does not build a plan for a particular goal, then the user knows that there is either no way to achieve the goal at all, or some higher-priority goals have consumed the necessary resources. If the user decides the unplanned goal is required, he can adjust the priorities to raise its place in the priority queue.

Note that the strict priorities do not eliminate the potential for goal interactions and plan optimization. If planning time is available, the system may search to find the best plan for a higher-priority goal that also makes more lower-priority goals achievable (*e.g.*, by choosing resource bindings that leave resources available for the lower-priority goals).

In this paper we describe our most recent implementation of priority-based Playbook tasking and task execution. We discuss how the SHOP2 hierarchical task network (HTN) planner[5] reasons about priorities to generate a team plan, re-planning whenever the goals and priorities evolve or the situation changes. We also describe how the OpenPRS[6] procedural system executes the resulting plan, managing the changing environment and updated plans in real-time.

II. The Simple Hierarchical Ordered Planner (SHOP2)

SHOP2 is a modern HTN planner with a relatively clean implementation that has performed well in past planning competitions.[5]Another advantage of the SHOP2 planning system is that it is available under a generous open-source license, and is maintained at SourceForge[1]. We have discussed elsewhere the application-based reasons for our preference for HTN over first-principles planning,[2,7] in the context of work on control of autonomous aerial vehicles.

Like other HTN planners, and unlike first-principles planners, SHOP2 searches top-down from a task or set of tasks, rather than chaining together primitive actions. SHOP2 and other HTN planners *decompose* complex tasks into more primitive sub-tasks (*methods*), thus building a plan tree that terminates at leaves corresponding to primitive actions (*operators*). As in earlier action representation languages, SHOP2 operators have separate add and delete lists, rather than effects, and SHOP2's variables are untyped. SHOP2 can also use more standard PDDL action representations, but we do not use them in this paper.

In addition to actions, SHOP2's language provides *methods* for describing how to perform complex tasks. A method definition associates a task with a set of preconditions and a task network. When the preconditions are satisfied,a task that matches the task in the method definition can be decomposed to the given task network. Task networks are lists of tasks that may be constrained to be `:ordered`, or that can be executed in any order (`:unordered`).

```
(:operator (!lase-target ?uav ?target-id )
  ((has-laser ?uav))     ;; preconditions
  ((has-laser ?uav))     ;; delete-list
  ())                    ;; add-list

(:operator (!strike-target ?uav ?target-id )
  ((has-missile ?uav))   ;; preconditions
  ((has-missile ?uav))   ;; delete-list
  ())                    ;; add-list

(:method (lase-target ?uav ?target-id)
   ((has-laser ?uav))    ;; precondition
   (!lase-target ?uav ?target-id))

(:method (strike-target ?uav ?target-id)
   ((has-missile ?uav)) ;; precondition
   (!strike-target ?uav ?target-id))

(:method (prosecute-target ?lasing-uav
             ?striking-uav ?target-id ?priority)
   ()    ;; no preconditions
         ;; task network, :ordered by default
   ((lase-target ?lasing-uav ?target-id)
    (strike-target ?striking-uav ?target-id)))
```

Figure 1: Simple methods that implement the domain-level `prosecute-target` play.

Figure 1 shows several operator and method definitions that form part of our very simple example domain. The example shows that to prosecute a target, we must build a plan that first performs the `!lase-target` operator and then the `!strike-target` operator.

[1] `http://sourceforge.net/projects/shop`

In this encoding, we follow the convention that primitive *operator* names begin with '!' and, since SHOP2 does not search over possible parameter bindings for operators, we wrap each operator in a similarly-named *method* that checks preconditions and supports search (in this case, over which UAV will perform each task). Also, this very simple encoding indicates that using the `!lase-target` operator actually deletes the `has-laser` predicate for the chosen UAV. While this is not very realistic, it effectively stops the planner from planning to use that UAV's laser for a different task at the same time. When this task is completed (or fails), we expect the planner to be re-run with a new/revised initial state and planning problem (task priority queue), so the system will re-consider how to use the UAV's laser at that time. We have used the same simplification for the `has-missile` predicate, meaning that a single UAV can only shoot at a single target at a time; a more complex multi-missile model could be easily added.

The full SHOP2 language and planner are capable of representing and reasoning about many other more complicated domain aspects that have been omitted, such as action durations, more complex resource models, conditional action effects, *etc*. In this paper, we are concerned with a different aspect: how can we give the planner a prioritized list of `prosecute-target` tasks and cause it to search for optimized solutions that respect the strict priority semantics?

SHOP2 does include a very limited form of optimization, which minimizes the sum of a user-defined cost function applied to each of the steps of a plan. This optimization is performed using branch-and-bound, and can be time-limited. The key to our strict priority planning is to exploit this optimization behavior.

III. Making SHOP2 Respect Priorities

The simplest way to get SHOP2 to plan for tasks while respecting strict priority is to use a scaled cost function that will tell the planner that skipping a higher-priority task (not planning to achieve it) is more costly than skipping any number of lower-priority tasks. Since the priority queue is presented to the planner all at once, it is easy to compute cost values that enforce this criterion. We then tell SHOP2 that, for each top-level task (play) that can be on the priority queue there are one or more methods that really achieve the goal and one "skip" method that does not. Choosing the skip method will incur the priority-based cost.

```
  ;; an internal primitive that just costs ?cost
(:operator (!!charge-cost ?cost)
        ((numberp ?cost))
        ()
        ()
        ?cost

  ;; the real method, with new priority argument
(:method (prosecute-target ?lasing-uav
             ?striking-uav ?target-id ?priority)
  ()    ;; no preconditions
        ;; task network, :ordered by default
  ((lase-target ?lasing-uav ?target-id)
   (strike-target ?striking-uav ?target-id)))

  ;; the skip method
(:method (prosecute-target ?lasing-uav
             ?striking-uav ?target-id ?priority)
        ;; pre: compute penalty based on priority
  ((assign ?cost (compute-cost ?priority)))
        ;; task network: charge penalty
  ((!!charge-cost ?cost)))
```

Figure 2: The most trivial approach to enforcing strict priorities: each top-level method requires a "skip" method.

Figure 2 illustrates this approach in simple form for our running example. While workable, this approach has several significant disadvantages. Most importantly, it requires that every top-level task have a custom-built "skip" method which has the same invocation conditions as the method(s) that really accomplish the task. While the construction of the skip methods could be easily automated, they still make it somewhat more challenging to determine whether a plan really achieves all of the goals or not; one must look one level down in the plan decomposition to see whether the chosen method results in an immediate `!!charge-cost` call or not. Another minor annoyance is the need to include the priority parameter in every top-level method,

when that value has no real meaning to the task itself, just to the planner's consideration of the task (and hence the parameter is never referenced within the non-skip methods).

```
  ;; Really plan for a prioritized task
(:method (plan-for ?priority ?taskspec)
  ()    ;; no preconditions
  ((:task . ?taskspec)))

  ;; Skip planning for a prioritized task
(:method (plan-for ?priority ?taskspec)
  ((assign ?cost (compute-cost ?priority)))
  ((!!charge-cost ?cost)))

;; Example priority=3 task invocation:
;; (plan-for 3 (prosecute-target ?L ?S target-12))
```

Figure 3: This cleaner approach uses generalized meta-level methods to implement strict priorities without task-specific skip methods.

Figure 3 illustrates a more elegant approach to accomplishing the same strict-priority planning. Rather than replicating skip methods for every top-level method at the domain level, we instead have just two meta-level methods that apply strict-priority planning semantics to all methods. Instead of invoking a desired top-level task directly, the priority queue will include invocations of the `plan-for` method, as shown in the figure. Then the planner will either use the first method, which reduces to actually planning for the task specification, or the second, generic skip method, which charges the penalty cost as before. The key trick here is in the first method, which really plans for an arbitrary task specification by taking advantage of SHOP2's very flexible unification system and its Lisp underpinnings. The `?taskspec` parameter can unify to a full task invocation specification (*e.g.*, `(prosecute-target ?L ?S target-12)`) and the method uses the '.' operator to join that task specification into the method's task network.

As a result, there is no need to replicate skip methods or modify top-level tasks with unused priority arguments. Furthermore, detecting whether a prioritized task is actually planned-for is easier because the plan either will or will not include a method that corresponds to the task specification (*e.g.*, `prosecute-target`).

IV. Evaluation

While our investigation is fairly new, we have conducted early experiments to assess the scalability of this prioritized plan optimization approach. As in the illustrations shown here, our experimental domains omit many details of the more realistic UAV planning problem, including action durations and UAV-selection heuristics that depend on travel distances and fuel levels. We expect that those aspects are not the primary source of complexity; rather, the scale issues arise from the large number of potential asset allocation choices (*i.e.*, which UAV performs which task).

To explore the approach's scalability, we created a set of very simple planning domains that included different numbers of `prosecute-target` tasks (T) and different numbers of laser-equipped UAVs (L) and missile-equipped UAVs (M). Since each `prosecute-target` task requires one of each UAV type, the total number of possible assignments is $(L!M!/((L-T)!(M-T)!)$ when $L >= T$ and $M >= T$. When there are more tasks than can be satisfied by the available UAVs, the planner must choose a subset to satisfy $(T!/(R!(T-R))!$ choices, where $R = min(L, M)$) and then consider the alternative asset assignments, giving $(L!M!/((L-R)!(M-R)!)$ permutations for each subset choice. For our test domains, where each task had a different priority but all UAV assignments are of equal value, the number of optimal plans is not increased as T exceeds R, because the optimal plans all fail to achieve the lower $T-R$ goals.

Figure 4 shows the resulting planning time for different numbers of prioritized tasks when the domain includes six laser-equipped UAVs ($L = 6$) and five missile-equipped UAVs ($M = 5$). We tested SHOP2 in three different modes, returning a single optimal plan (Opt-1), all the optimal plans (Opt-all), or all the plans regardless of optimality (All). As expected, returning a single optimal plan is faster than returning all the optimal plans. However, we were surprised that returning all the plans, regardless of cost, is so much faster than returning all the optimal plans (at least for $T < 5$, when the All mode does not run out of memory). While some computation is clearly needed to calculate and compare costs, the Opt-all mode should be able to quickly prune many plans. For example, all the tasks can be accomplished whenever $T < 6$, so any

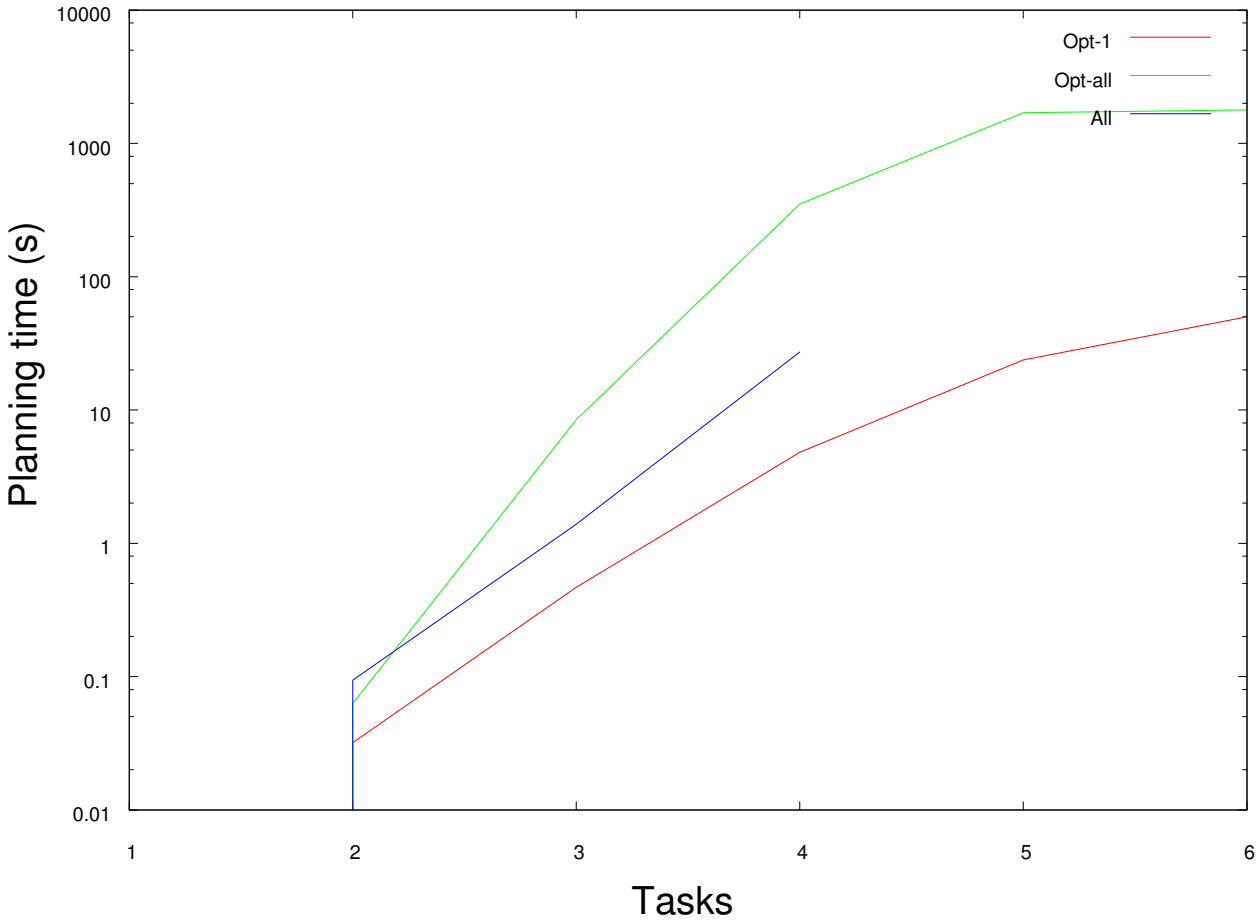

Figure 4: Planning time scales exponentially but still acceptably when SHOP2 is asked to only return a single optimal plan.

plans that involve skipping goals should be immediately pruned before being fleshed out fully. Nevertheless, Opt-all was a full order of magnitude slower than the non-optimizing algorithm. These early results may indicate that significant improvements may be identified by comparing the optimizing and non-optimizing algorithms.

V. Related Work

Early work in over-subscription scheduling emphasized domain-specific greedy algorithms that schedule higher priority tasks first and then perform local revisions/repairs.[8,9] A more recent survey found that simulated annealing worked best for scheduling oversubscribed satellite observations.[10] Over-subscription *planning* has received little attention until fairly recently. Some partial satisfaction planning approaches rely on using heuristics to select a subset of goals and then building plans to match.[3,11] Other techniques integrate goal-selection and plan-construction and use advanced heuristics to guide both.[12,13] However, all of these methods have been developed in the context of first-principles planners, rather than the more powerful HTN framework.

By integrating the cost of skipping a goal into the normal plan cost optimization framework, our approach allows a single unified HTN planning algorithm to address both goal selection and plan construction. However, we have not yet developed any heuristic guidance methods that can "understand" the representational trick of Figure 3. We may be able to build on recent preference-based planning extensions of SHOP2.[14]

VI. Conclusion and Future Directions

We have illustrated two domain modeling techniques that allow the SHOP2 HTN planner to control its own goals and thus, indirectly, its level of planning effort. The second, more elegant technique is both domain-independent and broadly generalizable to other forms of goal selection criterion, beyond the strict priority scheme we have used for our UAV tasking problem. In earlier work, we demonstrated a related technique in which SHOP2's optimization mode is used to address over-constrained problems by relaxing temporal constraints.[15] These techniques just scratch the surface of the full potential for meta-control and complex reasoning within SHOP2. For example, we could easily implement goal pruning based on planning time, so that as the planner's runtime grows or a deadline approaches, the planner sheds less-important goals.

Acknowledgments

This work was supported by the U.S. Army Aviation Applied Technology Directorate under SBIR Contract W911W6-08-C-0066. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. Government.

References

[1]Miller, C. A., Pelican, M. J. S., and Goldman, R. P., "Tasking Interfaces for Flexible Interaction with Automation: Keeping the User in Control," *Proceedings of the Conference on Human Interation with Complex Systems*, April - May 2000.

[2]Miller, C. A., Goldman, R. P., Funk, H. B., Wu, P., and Pate, B., "A Playbook Approach to Variable Autonomy Control: Application for Control of Multiple, Heterogeneous Unmanned Air Vehicles," *Proc. American Helicopter Society 60th Annual Forum*, June 2004, pp. 2146–2157.

[3]Nigenda, R. S. and Kambhampati, S., "Planning Graph Heuristics for Selecting Objectives in Over-subscription Planning Problems," *Proc. Int'l Conf. on Automated Planning & Scheduling*, 2005, pp. 192–201.

[4]von Winterfeldt, D. and Edwards, W., *Decision Analysis and Behavioral Research*, Cambridge University Press, Cambridge, ENGLAND, 1986.

[5]Nau, D., Au, T. C., et al., "SHOP2: An HTN Planning System," *Journal of Artificial Intelligence Research*, Vol. 20, 2003, pp. 379–404.

[6]Ingrand, F. F., Georgeff, M. P., and Rao, A. S., "An Architecture for Real-Time Reasoning and System Control," *IEEE Expert*, December 1992, pp. 34–44.

[7]Goldman, R. P., Haigh, K. Z., Musliner, D. J., et al., "MACBeth: A Multi-Agent Constraint-Based Planner," *AAAI Workshop on Constraints and AI Planning*, 2000, pp. 11–17.

[8]Kramer, L. and Giuliano, M., "Reasoning About and Scheduling Linked HST Observations with Spike," *Proc. Int'l Workshop on Planning and Scheduling for Space Exploration and Science*, 1997.

[9]Potter, W. and Gasch, J., "A photo album of earth: Scheduling Landsat 7 mission daily activities," *Proc. Int'l Symp. on Space Mission Operations and Ground Data Systems*, 1998.

[10]Globus, A., Crawford, J., Lohn, J., and Pryor, A., "A Comparison of Techniques for Scheduling Earth Observing Satellites," *Proc. Conf. on Innovative Applications of AI*, 2004.
[11]Smith, D. E., "Choosing objectives in over-subscription planning," *Proc. Int'l Conf. on Automated Planning & Scheduling*, 2004.
[12]van den Briel, M., Nigenda, R. S., Do, M. B., and Kambhampati, S., "Effective Approaches for Partial Satisfaction (Over-Subscription) Planning," *Proc. National Conf. on Artificial Intelligence*, 2004, pp. 562–569.
[13]Li, L. and Onder, N., "Generating Plans in Concurrent, Probabilistic, Over-Subscribed Domains," *Proc. National Conf. on Artificial Intelligence*, 2008, pp. 957–962.
[14]Sohrabi, S., Baier, J. A., and McIlraith, S. A., "HTN Planning with Preferences," *Proc. Int'l Joint Conf. on Artificial Intelligence*, July 2009.
[15]Goldman, R. P., Miller, C. A., Wu, P., Funk, H. B., and Meisner, J., "Optimizing to satisfice: Using optimization to guide users," *Proc. American Helicopter Societys Int'l Specialists Meeting on Unmanned Aerial Vehicles*, 2005.