A Cost-Directed Planner: Preliminary Report

Eithan Ephrati
AgentSoft Ltd. and
Department of Mathematics
and Computer Science
Bar Ilan University
Ramat Gan 52900, ISRAEL
tantush@sunlight.cs.biu.ac.il

Martha E. Pollack
Computer Science Department
and Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260, USA
pollack@cs.pitt.edu

Marina Milshtein
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260, USA
marisha@cs.pitt.edu

Abstract

We present a cost-directed heuristic planning algorithm, which uses an A^* strategy for node selection. The heuristic evaluation function is computed by a deep lookahead that calculates the cost of complete plans for a set of pre-defined top-level subgoals, under the (generally false) assumption that they do not interact. This approach leads to finding low-cost plans, and in many circumstances it also leads to a significant decrease in total planning time. This is due in part to the fact that generating plans for subgoals individually is often much less costly than generating a complete plan taking interactions into account, and in part to the fact that the heuristic can effectively focus the search. We provide both analytic and experimental results.

Introduction

Most of the work on search control for planning has been based on the assumption that all plans for a given goal are equal, and so has focused on improving planning efficiency. Of course, as has been recognized in the literature on decision-theoretic planning (Williamson & Hanks 1994; Haddawy & Suwandi 1994), the solutions to a given planning problem are not necessarily equal: some plans have lower execution cost, some are more likely to succeed, and so on.

In this paper, we present a cost-directed heuristic planner, which is capable of finding low-cost plans in domains in which actions have different costs associated with them. Our algorithm performs partial-order causal-link (POCL) planning, using an A^* strategy. The heuristic evaluation function is computed by a deep lookahead that calculates the cost of complete plans for a set of pre-defined top-level subgoals, under the (generally false) assumption that those subgoals do not interact. The essential idea is to treat a set of top-level subgoals as if they were independent, in the sense of (Korf 1987). For each of these subgoals, we independently find a subplan that is consistent with the current global plan, i.e., the partial plan in the current node. The sum of the costs of these subplans is then used as the heuristic component, h', of the A^* algorithm. The overall estimate f' ($= g + h'$, where g is the cost of the actions in the global partial plan) is used to determine which node to work on next. This contrasts with most POCL planners, which perform node selection using a shallow heuristic, typically a function of the number of steps and flaws in each node.

Our approach leads to finding low-cost plans, and in many circumstances it also leads to a significant decrease in total planning time. This is due in part to the fact that generating plans for subgoals individually is often much less costly than generating a complete plan, taking interactions into account(Korf 1987; Yang, Nau, & Hendler 1992). Moreover, while focusing on lower-cost plans, the heuristic function effectively prunes the search space. The use of the deep evaluation in node selection can outweigh the marginal additional complexity.

The Algorithm

We model plans and actions similarly to other POCL systems, except that we assume that each action has an associated cost. We also assume that the cost of a plan is equal to the sum of the costs of its consituent actions. At the beginning of the planning process, the global goal is partitioned into n exhaustive and mutually disjoint subgoal sets, which should be roughly equivalent in complexity. For simplicity in this paper we assume that each subgoal set is a singleton, and just speak about top-level subgoals, sg_i, for $1 \leq i \leq n$; we call the set of all these top-level subgoals SG . At a high level, our algorithm is simply the following:

Until a solution has been found do:

1. Select [1] a node p representing a partial global plan with minimal f' value.
2. Select a flaw in p to refine. For each successor node p_i generated:
   - Set the actual cost function $g(p_i)$ to be the actual cost of p plus the cost of any new action added

[1] The choose operator denotes non-deterministic choice, and hence, a possible backtrack point; the `select` operator denotes a heuristic choice at a point at which back-tracking is not needed.

in the refinement. (If the refinement is a threat resolution, then $g(p_i) = g(p)$).
- Independently generate a complete plan to achieve each of the *original* subgoals sg_i. Each such subplan must be consistent with the partial global plan that p_i represents, i.e., it must be possible to incorporate it into p_i without violating any ordering or binding constraints. Set $h'(p_i)$ to the sum of the costs of the complete subplans generated.

A formal definition of the algorithm is given in Figure 1. It relies on the following definitions, which are similar to those of other POCL algorithms, except for the inclusion of cost information for actions, and g and h' values, and subgoal partitions, for plans:

Definition 1 (Operator Schemata) *An* operator schema *is a tuple* $\prec T, V, P, E, B, c \succ$ *where* T *is an action type,* V *is the list of free variables,* P *is the set of preconditions for the action,* E *is the set of effects for the action,* B *is the set of binding constraints on the variables in* V*, and* c *is the cost.*[2] *A copy of some action* a *with fresh variables will be denoted by* $Fresh(a)$*. Given some action instance,* s*, we will refer to its components by* $T(s)$*,* $V(s)$*,* $P(s)$*,* $E(s)$*,* $B(s)$*, and* $c(s)$*.*

Definition 2 (Plan) *A* plan *is a tuple* $\prec \mathcal{S}, \mathcal{L}, \mathcal{O}, \mathcal{B}, \mathcal{A}, \mathcal{T}, SG, g, h' \succ$*, where* $\mathcal{S}$ *is a set of the plan steps,* $\mathcal{L}$ *is a set of causal links on* $\mathcal{S}$*,* $\mathcal{O}$ *is a set of ordering constraints on* $\mathcal{S}$*,* $\mathcal{B}$ *is a set of bindings of variables in* $\mathcal{S}$*,* $\mathcal{A}$ *is the set of open conditions,* $\mathcal{T}$ *is the set of unresolved threats,* SG *is the partition of* $\mathcal{A}$ *induced by the initial partition of top-level subgoals,* g *is the accumulated cost of the plan so far, and* h' *is the heuristic estimate of the remaining cost.*

The initial input to the planner is: $\{\prec \{s_0, s_\infty\}, \emptyset, \{s_0 < s_\infty\}, \emptyset, G, \emptyset, SG, 0, 0 \succ\}$, where s_0 is the dummy initial step, s_∞ is the dummy final step, G is the initial set of goals, and SG is a partition of G. The algorithm also accesses an operator library Λ.

As described above, the algorithm iteratively refines nodes in the plan space until a complete plan has been generated. Its main difference from other POCL planners is its computation of heuristic estimates of nodes. (See the boxed parts of the algorithm.) The algorithm maintains a queue of partial plans, sorted by f'; on each iteration, it selects a node with minimal f'. During refinement of that node, both the g and h' compnents must be updated. Updating g is straightforward: whenever a new step is added to the plan, its cost is added to the existing g value (Step 2(a)ii). Updating h' occurs in Step 4, in which subplans are generated for each of the top-level subgoals.

[2]To maintain an evaluation function that tends to underestimate, the cost of a step with uninstantiated variables is heuristically taken to be equivalent to the cost of its minimal-cost grounded instance.

CostDirectedPlanSearch (Q)

While $Q \neq \emptyset$ **Select** the first plan, $p = \prec \mathcal{S}, \mathcal{L}, \mathcal{O}, \mathcal{B}, \mathcal{A}, \mathcal{T}, SG, g, h' \succ$, in Q

1. [Termination] If $\mathcal{A} = \mathcal{T} = \emptyset$ return p.
2. [Generate Successors]
   Let $o' = l' = b' = s' = \emptyset$, and **Select** either:
   (a) An open condition $\langle d, s_d \rangle$ (in $\mathcal{A}$ of p), and **Choose** an establisher:
      i. [An Existing Global Step] $s' \in \mathcal{S}$ and b', s.t. $e \in E(s')$, $(e \equiv_{B \cup b'} d)$, and $(s' < s_d)$ is consistent with $\mathcal{O}$, or
      ii. [A New Step] $s' = Fresh(a)$ such that $a \in \Lambda$ and b', s.t. $e \in E(a)$, $(e \equiv_{B \cup b'} d)$. Let $g = g + c(s')$.
      Let $o' = (s' < s_d), l' = \{\langle s', d, s_d \rangle\}$.
   (b) An unresolved threat $\langle s_t, s_e, d, s_d \rangle \in \mathcal{T}$ of p, and Choose either:
      i. [Demotion] If $(s_t < s_e)$ is consistent with $\mathcal{O}$, let $o' = \{(s_t < s_e)\}$, or
      ii. [Promotion] If $(s_t > s_d)$ is consistent with $\mathcal{O}$, let $o' = \{(s_t > s_d)\}$.
   If there is no possible establisher or no way to resolve a threat, then fail.
3. [Update Plan] Let $\mathcal{S}' = \mathcal{S} \cup s'$, $\mathcal{L}' = \mathcal{L} \cup l'$, $\mathcal{O}' = \mathcal{O} \cup o'$, $\mathcal{B}' = \mathcal{B} \cup b'$, $\mathcal{A}' = (\mathcal{A} \setminus d) \cup P(s')$. Update $\mathcal{T}$ to include new threats.
4. [Update Heuristic Value]
   $h' = \sum_{sg_i^* \in SG} \mathsf{SubPlan}(\prec \mathcal{S}', \mathcal{L}', \mathcal{O}', \mathcal{B}', \emptyset, sg_i^*, 0 \succ)$.
5. [Update Queue]
   Merge the plan $\prec \mathcal{S}', \mathcal{L}', \mathcal{O}', \mathcal{B}', \mathcal{A}', SG, g, h' \succ$ back into Q sorted by its $g + h'$ value.

Figure 1: The Search for a Global Plan

There are two alternatives for subplanning. In this paper, we assume that subplanning is done by a fairly standard POCL algorithm, performing best first search. The subplanning process is invoked from the main program (in Step 4) with its steps, links, and ordering and binding constraints initialized to their equivalents in the global plan. The set of open conditions is initialized to sg_i^*, which consists of the open conditions associated with the ith original subgoal: any conditions from the original partition that remain open, plus any open conditions along paths to members of sg_i.[3] Essentially, when subplanning begins, it is as if it were already in the midst of planning, and had found the global partial plan; it then forms a plan for the set of open conditions associated with a top-level subgoal. As a result, the plan for the subgoal will be consistent with global plan. If a consistent subplan cannot be found, then the global plan is due to fail. Subplanning can thus detect dead-end paths.

[3]The main algorithm tags each establisher chosen in step 2a with the top-level goal(s) that it supports. We have omitted this detail from the Figure 1 to help improve readibility.

The subplanning process keeps track of the actual cost of the complete subplan found, and returns that value to the main algorithm.

An alternative method for subplanning would be to recursively call the global planning algorithm in Figure 1. This would be likely to further reduce the amount of time spent on each node, because it would amount to assuming independence not only among top-level goals, but also among their subgoals, and their subgoals' subgoals, and so on. On the other hand, it would lead to a less accurate heuristic estimate, and thus might reduce the amount of pruning. We are conducting further experiments, not reported in this paper, to analyze this trade-off.

Complexity

We next analyze the complexity of the cost-directed planning algorithm. Let b be the maximum branching factor (number of possible refinements for each node), and let d be the depth of search (number of refinements performed to generate a complete plan). Then, as is well known, the worst-case complexity of planning search is $O(b^d)$. During each iteration of the cost-directed algorithm, the most promising partial plan is refined, and, for each possible refinement, a complete subplan for each of the n elements of the original subgoal set (SG) is generated. Let b_i and d_i denote the breadth and depth of subplanning for the subgoal sg_i, and let $\hat{b}_i = \max_i b_i$, and let $\hat{d}_i = \max_i d_i$ $(1 < i < n)$. Then the complexity of each subplanning step in the algorithm (Step 4) is $O(n \times (\hat{b}_i)^{\hat{d}_i})$. So in the worst case, if there is no pruning, the overall complexity of the search is $O(b^d \times n \times (\hat{b}_i)^{\hat{d}_i})$. Because $\hat{b}_i \leq b$, and $\hat{d}_i \leq d$, the absolute worst case complexity is $O(n \times b^{2d})$.

However, for many planning domains, $\hat{b}_i$ and $\hat{d}_i$ are likely to be smaller than b and d. As noted by Korf (Korf 1987, p.68), and by Yang *et al.* (Yang, Nau, & Hendler 1992), planning separately for n subgoals will tend to reduce both b and d by a factor of n, i.e., $b_i \approx \frac{b}{n}$ and $d_i \approx \frac{d}{n}$. Note that reduction in search depth is due to the fact that, if there were no interactions among the subgoals, an overall plan would have length equal to the sum of the lengths of the plans for the subgoals. Of course, positive interactions will decrease the length of the overall plan, while negative interactions will increase it. We assume that the effects of negative interactions will be at least as great as the effects of positive interactions, which appears to be the case in many domains.

To obtain the maximum benefits of planning for individual subgoals separately, the subgoals must be of roughly equal "complexity" to one another. If virtually all of the work of the top-level planning problem is associated with a single subgoal, then planning separately for that subgoal will be almost as costly as planning for the entire set of subgoals. We therefore also assume planning domains in which it is possible to partition subgoals into sets of equal complexity. For domains for which this does not hold, it may still be possible to use the cost-directed planning algorithm, but it would require invoking it recursively for subplanning, as described earlier.

We next consider the effect of pruning. The heuristic function in the A^* search reduces the complexity of the search from $O(b^d)$ to $O(h^d)$ where h is the heuristic branching factor (Korf 1987). Thus, for planning problems with the properties mentioned above (balanced positive and negative interactions; capable of being paritioned into subgoals with roughly equal complexity) the overall complexity of the cost-directed algorithm is $O(h^d \times n \times (\frac{b}{n})^{\frac{d}{n}})$. Cost-directed search for these problems will thus consume less time than a full breadth-first planner as long as the following inequality holds:[4]

$$n/(n-1)\log h \leq \log b - \log n$$

Of course, no POCL planning algorithms actually use breadth-first node selection; this inequality simply provides a baseline for theoretical comparison. Later, we provide experimental comparison of our algorithm with best-first and branch-and-bound control strategies.

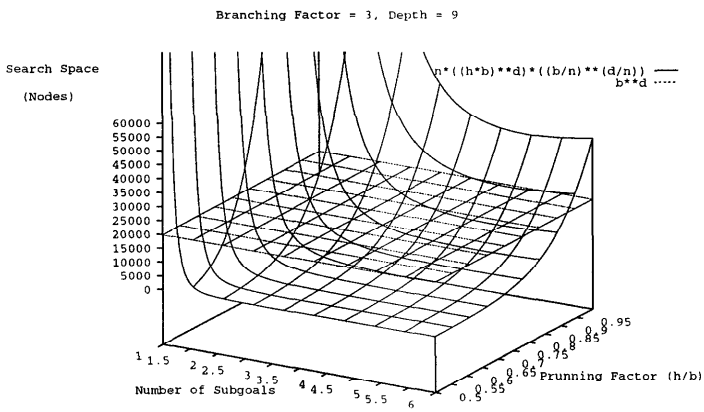

Figure 2: Typical Search Space Complexity

Figure 2 illustrates the inequality, comparing the complexity of cost-directed search with a breadth-first search (the level plane) as a function of the number of subgoals (n) and the pruning effect (h/b). We use a planning problem with a branching factor of 3 and a depth of 9 as an example. As the figure demonstrates, there exists a region in which the cost-directed planner performs better, namely the area in which the values of the curve fall below the level plane. In general, the effectiveness of the cost-directed planner increases with the pruning effect and the number of subgoals being used.

Note that this analysis assumes that the pruning factor is independent of the number of subgoal partitions.

[4]The complete derivation is given in (Ephrati, Pollack, & Milshtein 1996).

In reality, the pruning factor is inversely related to the number of subgoals—the higher the number of subgoals that are being used, the less accurate the heuristic evaluation will tend to become. Thus, for a specific problem, there exists some domain-dependent decomposition into subgoals which will optimize the performance of cost-directed search.

Caching Subplans

In the algorithm as so far presented, all subplans are recomputed on each iteration. Often, however, previously generated subplans may remain compatible with the current global partial plan, and it may thus make sense to cache subplans, and to check whether they are still consistent before attempting to regenerate them. An added advantage of caching the subplans is that the top-level planner can then attempt to re-use subplan steps; this helps maintain the accuracy of the heuristic evaluation function.

We therefore modify the original algorithm by attaching to each global plan a set of subplans ($\mathcal{P} = \{P_1, \ldots, P_n\}$) instead of just a partition of subgoals. Then, following each refinement of the global plan, the set of subplans is checked for consistency (with the newly refined global plan) and only the subplans that are incompatible are regenerated.

Although caching subplans may significantly reduce the number of subplans generated, it can also increase space complexity; details of this trade-off are given in (Ephrati, Pollack, & Milshtein 1996). In all the experiments reported below, subplans were cached.

Experimental Results

The complexity analysis above demonstrates the potential advantages of our approach. To verify the advantages in practice, we conducted a series of experiments comparing the cost-directed search with caching against the standard UCPOP algorithm with the built-in default search control, and againt UCPOP with a control mechanism that finds the minimal cost plan using branch-and-bound over action costs (henceforth called B&B).

Effects of Subgoal Independence

In the first experiment, our aim was to study the performance of the algorithms in domains in which the top-level subgoals had different levels of dependence. We therefore constructed three synthetic operator sets in the style of (Barrett & Weld 1994):

- Independent Subgoals: Barrett and Weld's $\theta_j D^0 S^n$ (p. 86). S^n means that it takes n steps to achieve each top-level subgoal. D^0 means that the delete set of each operators is empty, so all the top-level subgoals are independent. θ_j means that there are j different operators to achieve each precondition.
- Heavily Dependent Subgoals: Barrett and Weld's $\theta_j D^m S^n *$ (p. 93). As above, there are j different operators to achieve each precondition, and each top-level subgoal requires an n-step plan. $D^m *$ refers to a pattern of interaction among the operators. There is a single operator that must end up in the middle of the plan, because it deletes all the preconditions of the operators in stages 1 through k for some k, as well as all the effects of the operators in stages $k+1$ through n. In addition, for each stage, the operators for the ith subgoal delete all the preconditions of the same-stage operators for subgoals $j < i$. Consequently, there is only a single valid linearization of the overall plan; the dependency among top-level subgoals is heavy.
- Moderately Dependent Subplans: A variant of Barrett and Weld's $\theta_j D^m S^n$ (p. 91). Here again there are j different operators to achieve each precondition, and each top-level subgoal requires an n-step plan. In addition, the union of delete lists of the k + 1st stage operators for each subgoal sg_i delete *all* of the preconditions for the kth stage operators for all other $sg_j, j \neq i$. Because these preconditions are partitioned among the alternative kth stage operators, there are multiple valid linearizations of the overall plan. Although the top-level subgoals interact, the dependency isn't as tight, given the multiple possible solutions.

One thing to note is that the individual subplanning tasks in our experiments are relatively easy: the interactions occur among the subplans for *different* subgoals. Of course, this means that planning for UCPOP and B&B is comparatively easy as well. Further experiments are being conducted using operator sets with interactions within each subplan.

For the first experiment, we used problems with 4 top-level subgoals, and built the operator sets so that each top-level subgoal required a plan of length 3 (i.e., we used the S^3 version of each of the operator sets). The actions were randomly assigned costs between 1 and 10. Finally, we varied the number of operators achieving each goal (the j in θ_j) to achieve actual branching factors, as measured by UCPOP, of about 3 for the case of independent subgoals, about 2 for medium dependent subgoals, and about 1.5 for heavily dependent subgoals. The results are shown in Table 1, which lists

- the number of nodes generated and the number of nodes visited (these numbers are separated by an arrow in the tables);
- the CPU time spent in planning.[5]

[5]For certain problems that required a great deal of memory, the garbage collection process caused Lisp to crash, reporting an internal error. Schubert and Gerevini (Schubert & Gerevini 1995) encountered the same problem in their UCPOP-based experiments, with problems that required a

- the cost of the plans generated (sum of action costs).

As can be seen in the table, COST performed best in all cases, not only examining significantly fewer nodes, but taking an order of magnitude less time to find a solution than B&B. Not surprisingly, COST's advantage increases with the degree of independence among the subgoals: the greater the degree of independence among the subplans, the more accurate is COST's heuristic function, and thus the more effective its pruning. However, even in the case of heavy dependence among top-level subgoals, COST performs quite well.

Dependency	Planner	Nodes	Time	Cost
Independent	UCPOP	failure	39927.89	-
	B&B	35806 → 12900	2183.21	37
	COST	60 → 22	126.13	37
Medium	UCPOP	failure	40348.54	-
	B&B	2/4 failures 14827 → 5198 (succ. only)	9185.691	-
	COST	309 → 146	613.54	37
Heavy	UCPOP	failure	*	-
	B&B	2/4 failures 42978 → 17567 (succ. only)	12727.83	-
	COST	780 → 444	3834.517	37

Table 1: Varying dependency (4 medium uniform subgoals, 3 Steps each, $b \approx 3$ for independent, $b \approx 2$ for medium, $b \approx 1.5$ for heavy)

Effects of Uniformity

The next thing we varied was the uniformity of action cost: see Tables 2 and 3. For this experiment, we used the independent subgoal operator set described above, with 3 steps to achieve each subgoal, and an actual branching factor of approximately 3. The experiment in Table 2 involved 6 top-level subgoals, while the experiment in Table 3 involved 4. In both experiments, we varied the distribution of action costs: in the highly uniform environment, all actions were assigned a cost of 1; in the medium uniform environment, they were randomly assigned a cost between 1 and 10; in the the low uniform environments, they were randomly assigned a cost between 1 and 100.

Both UCPOP and B&B failed to find a solution for problems with 6, and even with 4 independent subgoals within the 150,000 nodes-generated limit. In highly uniform domains, i.e., those in which all actions had the same cost, our cost-directed algorithm fared no better: it also failed, although by exceeding memory limits. This is not surprising: if all actions have the same costs, then no pruning will result from a strategy based on cost comparison, and the space taken by caching subplans will be enormous.

lot of memory. We do not report results for these problems, but instead put an asterisk (*) in the time column. Also, in some cases, B&B succeeded on some operator sets, and failed on others. For those cases, we report the average time taken on *all* runs (which will be an underestimate, as the failed run were terminated after 150,000 generated nodes). We report the number of nodes only for the successful cases. Finally, we do not report the costs found in these cases, because the high-cost plans are typically the cases that fail. In no case did B&B find a lower-cost plan than COST.

Degree	Planner	Nodes	Time	Cost
Low	UCPOP	failure	28980.45	-
	B&B	failure	26165.37	-
	COST	70 → 26	210.44	2167
Medium	UCPOP	failure	36519.96	-
	B&B	failure	26952.57	-
	COST	134 → 44	407.84	54
High	UCPOP	failure	14986.58	-
	B&B	failure	27304.23	-
	COST	failure	*	-

Table 2: Varying Uniformity (6 independent subgoals, 3 Steps each, $b \approx 3$)

Degree	Planner	Nodes	Time	Cost
Low	UCPOP	failure	14617.54	-
	B&B	65000 → 23158	12708.66	1761
	COST	50 → 18	98.22	1761
Medium	UCPOP	failure	39927.89	-
	B&B	35806 → 12900	2183.21	37
	COST	60 → 22	126.13	37
High	UCPOP	failure	14528.38	-
	B&B	failure	26988.26	-
	COST	failure	*	-

Table 3: Varying Uniformity (4 independent subgoals, 3 Steps each, $b \approx 3$)

However, when the environments become less uniform, we see the payoff in the cost-directed approach. For low uniform environments and a planning problem with 6 subgoals (Table 2), the cost-directed planner finds plans by generating less than 70 nodes, taking about 3.5 minutes—while UCPOP and B&B failed, after generating 150,000 nodes, and taking over 7 hours. Even with medium uniformity, cost-directed planning succeeds quickly, while the other two approaches fail. Similar results are observed for the smaller (4 subgoal) problem (Table 3).

Note again that these are very easy problems, given the independence of the top-level subgoals. Indeed, the optimal strategy would have been to plan completely separately for each subgoal, and then simply concatenate the resulting plans. However, one may not know, in general, whether a set of subgoals is independent, prior to performing planning. The cost-directed search performs very well, while maintaining a general, POCL strategy that is applicable to interacting, as well as independent, goals.

Other Factors Effecting Planning

Several other key factors that are known to effect the efficiency of planning are the average branching factor, the length of the plan, and the number of top-level subgoals. We have conducted, and are continuing to conduct, experiments that vary each of these factors; so far, our results demonstrate that in a wide range of environments, the COST algorithm performs very well, finding low-cost plans in less time than either UCPOP or B&B (Ephrati, Pollack, & Milshtein 1996).

Related Research

Prior work on plan merging (Fousler, Li, & Yang 1992; Yang, Nau, & Hendler 1992) has studied the problem of forming plans for subgoals independently and then merging them back together. Although similarly motivated by the speed-up one gets from attending to subgoals individually, our work differs in performing complete planning using a POCL-style algorithm, as opposed to using separate plan merging procedures.

The idea of using information about plan quality during plan generation dates back to (Feldman & Sproull 1977); more recent work on this topic has involved the introduction of decision-theoretic notions (Haddawy & Suwandi 1994). Pérez studied the problem of enabling a system to learn ways to improve the quality of the plans it generates (Pérez 1995). Particularly relevant is Williamson's work on the PYRRHUS system (Williamson & Hanks 1994), which uses plan quality information to find an optimal plan. It performs standard POCL planning, but does not terminate with the first complete plan. Instead, it computes the plans's utility, prunes from the seach space any partial plans that are guaranteed to have lower utility, and then resumes execution. The process terminates when no partial plans remain, at which point PYRRHUS is guaranteed to have found an optimal plan. What is interesting about PYRRHUS from the perspective of the current paper is that, although one might expect that PYRRHUS would take significantly longer to find an optimal plan than to find an arbitrary plan, in fact, in many circumstances it does not. The information provided by the utility model results in enough pruning of the search space to outweigh the additional costs of seeking an optimal solution. This result, although obtained in a different framework from our own, bears a strong similarity to our main conclusion, which is that the pruning that results from attending to plan quality can outweigh the cost of computing plan quality.

Conclusions

We have presented an efficient cost-directed planning algorithm. The key idea underlying it is to replace a shallow, syntactic heuristic for node selection in POCL planning with an approximate, full-depth lookahead that computes complete subplans for a set of top-level subgoals, under the assumption that these subgoals are independent. Our analytical and experimental results demonstrate that in addition to finding low-cost plans, the algorithm can significantly improve planning time in many circumstances. The performance of the algorithm is dependent upon the possibility of decomposing the global goal into subgoal sets that are relatively (though not necessarily completely) independent, and that are roughly equivalent in complexity to one another.

The experiments we presented in this paper support the main hypothesis, but are by no means complete. We are continuing our experimentation, in particular, studying domains that involve a greater degree of interaction within each subplan. In addition, we are investigating the trade-off between using a complete POCL planner for subplanning, as in the experiments reported here, and recursively calling the main algorithm for subplanning. Finally, we are developing techniques to make the cost-directed search admissible.

Acknowledgments This work has been supported by the Rome Laboratory of the Air Force Material Command and the Advanced Research Projects Agency (Contract F30602-93-C-0038), by the Office of Naval Research (Contract N00014-95-1-1161) and by an NSF Young Investigator's Award (IRI-9258392).

References

Barrett, A., and Weld, D. 1994. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67(1):71–112.

Ephrati, E.; Pollack, M. E.; and Milshtein, M. 1996. A cost-directed planner. Technical Report, Dept. of Computer Science, Univ. of Pittsburgh, in preparation.

Feldman, J. A., and Sproull, R. F. 1977. Decision theory and artificial intelligence II: The hungry monkey. *Cognitive Science* 1:158–192.

Fousler, D.; Li, M.; and Yang, Q. 1992. Theory and algorithms for plan merging. *Artificial Intelligence* 57:143–181.

Haddawy, P., and Suwandi, M. 1994. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings of the Second International Conference on AI Planning Systems*, 266–271.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.

Pérez, M. A. 1995. Improving search control knoweldge to improve plan quality. Technical Report CMU-CS-95-175, Dept. of Computer Science, Carnegie Mellon University. Ph.D. Dissertation.

Schubert, L., and Gerevini, A. 1996. Accelerating partial order planners by improving plan and goal choices. Technical Report 96-607, Univ. of Rochester Dept. of Computer Science.

Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 176–181.

Yang, Q.; Nau, D. S.; and Hendler, J. 1992. Merging separately generated plans with restricted interactions. *Computational Intelligence* 8(2):648–676.