Exploiting Domain Knowledge with a Concurrent Hierarchical Planner

Bradley J. Clement and Edmund H. Durfee
University of Michigan
Ann Arbor, MI 48109
{bradc, durfee}@umich.edu

Abstract

Based on recent research about coordinating concurrent hierarchical plans (CHiPs), we introduce a sound and complete hierarchical planner that can better reason about precomputed conditions (*summary information*) of abstract plans to potentially make better refinement decisions than previous approaches. A reasonable criticism of this technique is that the summary information can grow exponentially as it is propagated up a plan hierarchy. This paper analyzes the complexity of the problem to show that in spite of this exponential growth, finding solutions at higher levels is still exponentially cheaper than at lower levels. In addition, this paper offers heuristics, including "fewest threats first" (FTF) and "expand most threats first" (EMTF), that take advantage of summary information to smartly direct the search for a global plan. Experiments show that for a particular domain these heuristics could greatly improve the search for the optimal global plan compared to two other heuristics (FAF and ExCon) that have both been successful in Hierarchical Task Network (HTN) planning.

Introduction

In HTN planning, as formalized in (Erol, Hendler, & Nau 1994), domain knowledge is encoded in the form of abstract plans in an effort to quickly guide the refinement of an abstract network of tasks (representing goals) into a primitive-level plan. At any step in the refinement process, the planner may choose to either constrain the binding of a variable, decompose an abstract task, select or deselect a decomposition method to apply, or add an ordering constraint. The performance of such planners depends on how well the plans are structured and these refinement choices are made.

Most previous strategies for making these refinement choices take a "least commitment" approach to avoid unnecessary branching in search state expansion (Tsuneto *et al.* 1996). However, these heuristics cannot avoid backtracking due to task interaction conflicts because they only detect conflicts when the refinement reaches the primitive level. The approach described here aims to avoid such backtracking by reasoning about interactions of operators at abstract levels.

Recent research has shown that by deriving and reasoning about the *external conditions* of an abstract plan operator (the conditions that must be achieved by other plan steps before this abstract step), the planner can better select tasks to decompose in order to avoid conflicts and the backtracking that previous least commitment strategies do not avoid (Tsuneto, Hendler, & Nau 1998). Our approach differs in that we reason about all of the preconditions, postconditions, and intermediate conditions that may or must hold for an abstract plan operator based on its potential refinements. By summarizing this information for all abstract plans, the planner can reason more completely about abstract plan interactions to make more informed decisions about refinement and to even find solutions at abstract levels.

This idea originated in research directed at coordinating the individually formed hierarchical plans of a group of agents that share an environment with limited resources. In (Clement & Durfee 1999c) we describe a general coordination algorithm, and an example was used to demonstrate that using this strategy to find solutions at different levels of abstraction can improve the overall performance of coordinating and executing plans. In this paper we analyze the complexity of deriving and using summary information to show that this result also applies to the planning problem. Another advantage of finding abstract solutions is that the agent(s) retain flexibility in their plans by preserving further refinement choices. Robust plan execution systems such as PRS (Georgeff & Lansky 1986), RAPS (Firby 1989), and JAM (Huber 1999) can then be used to interleave refinement with execution and provide some amount of recovery from unexpected events.

Coordinating abstract plans involves reasoning about the concurrent execution of actions, but even for the planning problem, it is necessary to reason about conflicts that involve conditions that must hold during the execution of abstract plans. In (Clement & Durfee 1999a), we describe a model of concurrent hierarchical plans (CHiPs) and their execution that can be used in a hierarchical planner to find solutions that allow for parallel activity. In addition, we detailed a procedure for deriving summary conditions, proofs of their properties, and sound and complete mechanisms for determining different interactions of abstract plans based on summary conditions. This paper describes a sound and complete concurrent hierarchical planner built on these mechanisms that can use summary information to better exploit domain knowledge than previous heuristics and that can find consistent CHiPs instead of fully refined primitive-level plans.

This paper makes the following contributions:

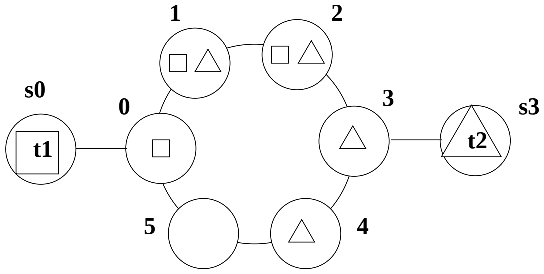

Figure 1: Transports $t1$ and $t2$ must pick up square and triangle evacuees respectively and return them to $s0$ or $s1$.

- complexity analysis showing that finding solutions at higher levels can be exponentially less expensive than at lower levels;
- heuristics, including "fewest threats first" (FTF) and "expand most threats first" (EMTF), that take advantage of summary information;
- a sound and complete concurrent hierarchical planner that uses these heuristics for finding consistent concurrent hierarchical plans; and
- preliminary experiments that show how these heurisitics may greatly save computation time in finding the optimal plan compared to "fewest alternatives first" (FAF) and *external condition* (ExCon) heuristics that have been shown to be successful in HTN planning.

A Simple Example

This example illustrates how a concurrent hierarchical planner can use summary information to resolve conflicts and optimize the total completion time of a plan. In a non-combative evacuation operation (NEO) domain, transports are responsible for visiting certain locations along restricted routes to pick up evacuees and bring them back to safety points. To avoid the risk of oncoming danger (from a typhoon or enemy attack), the planner must ensure that transports avoid collisions along the single lane routes and accomplish their goals as quickly as possible. Transports are allowed to be at the same location at the same time.

Suppose there are two transports, $t1$ and $t2$, located at safety points $s0$ and $s2$ respectively, and they must visit the locations 0, 1, and 2 and 2, 3, and 4 respectively and bring evacuees back to safe locations as shown in Figure 1. Because of overlap in the locations they must visit, the planner must synchronize their actions in order to avoid collision. The planner's goal network includes two unordered tasks, one for each transport to *evacuate* the locations for which it is responsible. The high-level task for $t1$ decomposes into a primitive action of moving to location 0 on the ring and an abstract plan to traverse the ring. $t1$ can travel in one direction around the ring without switching directions, or it can switch directions once. $t1$ can then either go clockwise or counterclockwise and, if switching, can switch directions at any location and travel to the farthest location it needs to visit from where it switched. Once it has visited all the locations, it continues around until it reaches the first safety point in its path. The task for $t2$ can be refined similarly.

Suppose the planner derives summary information for the plan hierarchy and attempts to resolve conflicts. Looking just at the summary information one level from the top, the planner can determine that if $t1$ finishes evacuating before $t2$ even begins, then there will be no conflicts since the external conditions of $t1$'s *evacuate* plan are that none of the routes are being traversed. However, this solution is inefficient (length of 16 steps) since there is no parallel action. If we wish to get more concurrency in the global plan, then the planner must expand the top-level plans into more detailed plans and resolve conflicts there. At a mid-level expansion where both transports move clockwise without switching directions, the algorithm finds a solution with a length of only eight steps. Now the planner can eliminate resolving threats for any global plan whose length can be no shorter than eight. To find the optimal solution, the planner must almost completely expand the hierarchy. This is a plan of length seven where $t1$ moves clockwise until it reaches location $s3$, and $t2$ starts out clockwise, switches directions at location 4, and then winds up at $s0$. For this solution $t1$ waits at location 2 for one time step to avoid a collision on the route from location 2 to location 3.

Overview

In the next section, we describe how concurrent hierarchical planner can use summary information. Then we explain why it is easier to compute abstract solutions at higher levels than at lower levels with a complexity analysis of the algorithms for deriving and using summary information. Next we show experimental results verifying that summary information can greatly improve the search for the optimal global plan even when it exists at the lowest level of primitive actions.

Concurrent Hierarchical Planner

Our approach to searching for concurrent hierarchical plans mainly differs from other HTN planners in that we first try to resolve conflicts among the tasks at the top-level of the hierarchy before decomposing them. Then, as those tasks are decomposed, we can again try to resolve conflicts, and iteratively expand subplans until a "feasible" solution is found. This follows the general coordination algorithm described in (Clement & Durfee 1999c). Here we explain the basic mechanisms for deriving and using summary information and then describe a specific algorithm that we use to evaluate the effectiveness of a concurrent hierarchical planner. All terms and mechanisms mentioned here are formalized in (Clement & Durfee 1999a).

CHiPs

As described here, hierarchical plans are non-primitive plans that each have their own sets of conditions, a set of subplans, and a set of ordering constraints over the subplans. These ordering constraints can be conjunctions of temporal interval relations (Allen 1983) or point relations over endpoints of plan execution time intervals. A primitive plan is only different in that it has no subplans. In the style of STRIPS planning operators (Fikes & J. 1971), each of these plans has sets of preconditions and effects.[1] However, since we wish to reason about performing tasks in parallel, we also associate a set of inconditions with each plan so that threats during the execution of a task can be represented.

Each plan in the hierarchy is either a primitive plan, an *and* plan, or an *or* plan. An *and* plan decomposes into a set of plans that each must be accomplished according to specified temporal constraints. An *or* plan decomposes into a set of plans of which only one must be accomplished. So, for the example given in the Introduction, there is an *or* plan that would have subplans for traveling clockwise or counterclockwise, and there are *and* plans for chaining primitive level movements between locations to get a transport around the ring. The decomposition of a CHiP is in the same style as that of an HTN as described in (Erol, Hendler, & Nau 1994). An *and* plan is a task network, and an *or* plan is an extra construct representing a set of all tasks that accomplish the same goal or compound task. Tasks in a network are subplans of the plan corresponding to the network. CHiPs were originally modeled to represent a variety of interesting procedures executable by PRSs and are designed to share aspects of PRS procedures as well as HTNs.

Plan Summary Information

We derive summary conditions for CHiPs by propagating the conditions from the primitive level up the hierarchy. The procedure is quick ($O(n^2c^2)$ for n plans in the hierarchy each with c conditions) because the summary conditions of a plan are derived only from its own conditions and the summary conditions of its immediate subplans.

A similar approach, (Tsuneto, Hendler, & Nau 1998), computes the *external conditions* of abstract plans, which are the preconditions required external to the abstract plans in order for them to be executed successfully. We redefine these as *external preconditions* and additionally employ *external postconditions*, the effects seen external to an abstract plan. Since the coordination problem requires reasoning about the concurrent execution of actions, we also derive *summary inconditions*, the intermediate, or internal, conditions that must or may be required to hold during an abstract plan step for the execution to be successful. Summary preconditions, inconditions, and postconditions are computed for each plan to represent the external preconditions, internal conditions, and external postconditions respectively. Modal information about whether these conditions *must* or *may* hold and whether they must hold throughout the plan's execution (*always* or *sometimes*) is kept to reason about whether certain plan interactions must or may occur.

Temporal Interactions of CHiPs

In conventional planning, we often speak of *clobbering* and *achieving* preconditions of plans (Weld 1994). With CHiPs these notions are slightly different since inconditions can clobber and be clobbered. We use these concepts to determine whether a summary precondition of a plan should be a summary condition of its parent. A summary precondition is an *external* precondition of its subplans, and what makes the precondition external is that it is not *achieved* by another subplan—it needs to be met outside the scope of the parent plan. A summary postcondition is external because it is a net effect of the execution of the subplans. Thus, we need to also determine when a postcondition is *undone* by another subplan since a postcondition is not external if it is undone.

Determining these relationships helps us derive summary information, but it also helps identify threats within the plan hierarchy. For example, a subplan p cannot clobber a condition c of a subplan q if there is another subplan r ordered between p and q that achieves c for q. However, if r only *may* achieve c because c is a *may* postcondition of r, then p threatens c in q. Reasoning about these kinds of interactions, we can determine that a set of temporal relations can hold among a set of expended subplans no matter how they are refined ($CanAnyWay$) or that certain relations cannot hold for any refinement ($\neg MightSomeWay$). These properties also depend on whether or not the subplans are consistent underneath. Local consistency for all abstract plans in hierarchy can be determined during the derivation of their summary information. As the procedure for determining $CanAnyWay$ and $MightSomeWay$ is similar to propagating summary information, its complexity is also $O(n^2c^2)$ for n plans with c conditions each (Clement & Durfee 1999a).

Planning as Searching for CHiPs

Since we can determine whether abstract plans $CanAnyWay$ or $MightSomeWay$ be executed successfully under particular ordering constraints, we can integrate this into an algorithm that searches for a consistent abstract plan (CHiP). As shown later in this section, the planner we describe here is sound and complete. Like other HTN planners, the search starts out with the goal task network and applies refinement operators to generate new search states.

A state of the search is a partially decomposed task network that we represent as an *and* plan, a set of temporal constraints, and a set of blocked plans. The subplans of the *and* plans are the leaves of the partially expanded hierarchies of the agents. The set of temporal constraints includes synchronization constraints added during the search to resolve conflicts in addition to those already specified in *and* plans that are decomposed. Blocked subplans keep track of pruned *or* subplans.

The operators of the search are expanding nonprimitive plans, blocking *or* subplans, and adding temporal constraints on pairs of plans. When a plan is expanded, it is replaced by its subplans, and the ordering information is updated in the global plan. *Or* plans are only replaced by a subplan when all other subplans are blocked.

Blocking an *or* subplan can be effective in resolving a constraint in which the other *or* subplans are not involved. This can lead to least commitment abstract solutions that leave the agents flexibility in selecting among multiple applicable subplans. Completeness is

[1] These are not summary conditions. A postcondition of a CHiP is equivalent to a high-level effect in an HTN (Erol, Hendler, & Nau 1994).

not sacrificed, however, if the *or* subplans are selected instead of blocked as long as search nodes are generated by selecting each subplan.[2]

Adding temporal constraints should only generate new search nodes when the ordering is consistent with the other global and local constraints. In essence, this operator is used to find a synchronization of the individual agents' plans at some level(s) of expansion. In our experiments, we separated the search for synchronizations from the expansion and selection of subplans. An outer search was used to explore the space of plans at different levels of abstraction. For each state in the outer search, an inner search explores the space of plan orderings by resolving threats with temporal constraints.

Variable binding for an abstract plan can be done by reasoning about the summary conditions of other plans that must or may clobber or achieve a condition with unbound variables. For example, such a condition may be threatened by some plan, but the threat may be resolved by binding the variable to a value such that another plan could be ordered to achieve the condition.

This refinement can be done to reach all potential decompositions in the same style as UMCP, a sound and complete HTN planner (Erol, Nau, & Hendler 1994). The soundness and completeness of the concurrent hierarchical planner then rests on the soundness and completeness of determining $CanAnyWay$ relations among abstract plans. At any point in the refinement, a partially or fully expanded plan hierarchy and corresponding ordering constraints may be determined to have the $CanAnyWay$ property and, thus, be returned as a solution. Any fully refined primitive plan is a solution if and only if $CanAnyWay$ is true. In addition, if $MightSomeWay$ is false, the search state can be pruned.

Heuristics Using Summary Information

As discussed in (Clement & Durfee 1999c), summary information is valuable for finding CHiPs at abstract levels. However, this information can also be valuable in directing the search to avoid branches in the search space that lead to inconsistent or suboptimal plans. Inconsistent plans can be pruned away at the abstract level by doing a quick check to see if $MightSomeWay$ is false. Another strategy that is employed is to first expand plans involved in the most threats. For the sake of completeness, the order of plan expansions does not matter as long as they are all expanded at some point when the search trail cannot be pruned. But, employing the "expand on most threats first" (EMTF) heuristic aims at driving the search down through the hierarchy to find the subplan(s) causing conflicts with others so that they can be resolved more quickly. This is similar to a most-constrained variable heuristic often employed in constraint satisfaction problems. Another heuristic used in parallel in our experiments is "fewest threats first" (FTF). Here the search orders nodes in the outer search queue by ascending numbers of threats to resolve. By trying to resolve the threats of global plans with fewer conflicts, it is hoped that solutions can be found more quickly. In addition, in trying to find optimal solutions in the style of a branch-and-bound, we use the cost of abstract solutions to prune away branches of the search space whose minimum cost is greater than the maximum cost of the current best solution. This technique can be used without summary information, but then only solutions at the primitive level can be used to prune the search space. We report experimental results in a later section that show that these heuristics can greatly improve coordination performance.

[2] "Selecting" a subplan just means blocking all others.

Complexity

In (Clement & Durfee 1999c), anecdotal evidence was given to show that coordinating at higher levels of abstraction is less costly because there are fewer plan steps. But, even though there are fewer plans at higher levels, those plans have greater numbers of summary conditions to reason about because they are collected from the much greater set of plans below. Here we argue that even in the worst case where summary conditions increase exponentially up the hierarchy, finding solutions at abstract levels (whether for the planning or coordination problem) is expected to be exponentially cheaper than at lower levels.

The procedure for deriving summary conditions works by basically propagating the conditions from the primitives up the hierarchy to the most abstract plans. Because the conditions of any non-primitive plan depend only on those of its immediate subplans, deriving summary conditions can be done quickly. If c' is the number of literals in each plan's pre-, in-, and postconditions, then the complexity is $O(n^2c'^2)$ as follows.[3] Here, the worst case is when all plans are *and* plans, and the conditions of each plan are completely different than those of any other plan. Consider such a hierarchy with n plans, b subplans for each non-primitive plan, and depth d.[4] At the next-to-bottom depth level $d-1$, the procedure tests each of c' conditions in each condition set of the b primitive subplans of each of the b^{d-1} plans to see if they are achieved/clobbered/undone by any other subplan attempting to assert c' inconditions and c' postconditions. Thus, a constant number of operations must be performed when comparing each condition in each subplan with every other condition in every other subplan resulting in $O(b^2c'^2)$ operations. So, for b^{d-1} plans, $O(b^{d-1}b^2c'^2)$ operations are performed to derive summary information at level $d-1$. At level $d-2$, there are b^{d-2} plans, and the number of conditions that must be compared among their subplans additionally includes those propagated from the primitive level for a total of $c' + bc'$ conditions. Thus, $O(b^{d-2}b^2(c' + bc')^2)$ operations are performed at level $d-2$. This generalizes to $O(\sum_{i=1}^{d} b^{d-i}b^2(b^{i-1}c')^2)$

[3] In (Clement & Durfee 1999a), it was shown that the complexity of this is $O(n(log^2n)c^2)$ for n non-primitive plans with c conditions in each plan's *summary* pre-, in-, and postconditions. This, however, does not tell us how the complexity grows as a result of summary conditions accumulating in greater and greater sets as they are propagated up the hierarchy.

[4] We consider the root at depth level 0 and the leaves at level d.

operations for the entire hierarchy. We can reduce this to $O(b^2c'^2\sum_{i=1}^{d} b^{d+i-2}) = O(b^{2d}c'^2)$, and since $n = O(b^d)$, the complexity can be simply stated as $O(n^2c'^2)$.

In this worst case, the number of summary conditions for an abstract plan grows exponentially as you go up the hierarchy. There are $3nc'$, or $O(b^dc')$, summary conditions at the root of the hierarchy—this is the total number of pre-, in-, and postconditions in the hierarchy. One level down, there are $O(b^{d-1}c')$ summary conditions, and there are $O(b^{d-2}c')$ at level 2 while there are only c' at level d. One might argue that in such cases deriving summary information only increases computation. But, actually, exponential computation time is saved when decisions based on summary information can be made at abstract levels because the complexity from exponential growth in the number of plans down the hierarchy outweighs the complexity of conditions growing exponentially up the hierarchy. This is because the only known algorithms for synchronizing plan steps to avoid conflicts are exponential with respect to the number of plans expanded in the hierarchy, which also grows exponentially with the depth. This exponential growth down the hierarchy outweighs the exponential growth of summary conditions in plans up the hierarchy. So the improvements made using summary information can make exponential savings while only incurring a small polynomial overhead in deriving and using summary information.

Let's make this more clear. At the ith depth level in the hierarchy, each of the $O(b^i)$ plans has $O(b^{d-i}c'^2)$ summary conditions in the worst case. As described in the next section, the algorithm to check whether a particular ordering of these plans results in all plans executing successfully has a complexity of $O(n^2c^2)$ for n plans and c summary conditions for each of those plans. Checking such a synchronization for the plans at any level i in a plan hierarchy is, thus, $O(b^{2i}b^{2d-2i}c'^2) = O(b^{2d}c'^2)$. So, since i drops out, the complexity of doing this check is *independent of the depth level*. But, there is a huge space of $n! = O((b^i)!)$ sequential orderings[5] of the n plans at level i to be checked to find a valid synchronization. Thus, the search space grows doubly exponentially down the hierarchy despite the worst case when the number of conditions grows exponentially up the hierarchy. This argument assumes that finding a valid synchronization is intractable, so we show that it is actually NP-complete. We reduce 3SAT to the THREAT RESOLUTION problem for STRIPS planning and claim that a similar reduction can be done for our problem that allows concurrent execution. The proof can be found in (Clement & Durfee 1999b).

There are only *and* plans in this worst case. In the case that there are *or* plans, by similar argument, being able to prune branches at higher levels based on summary information will greatly improve the search despite the overhead of deriving and using summary conditions. Obviously, the computational savings of using summary information will be even greater when there are conditions common to plans on the same level, and the number of summary conditions does not grow exponentially up the hierarchy. Still, surely there are cases where none of the details of the plan hierarchy can be ignored, and summary information would incur unnecessary overhead, but when the size of problem instances are scaled, dealing with these details will likely be infeasible anyway.

[5] There are more for other orderings allowing for concurrent execution.

Experiments

The experiments described here used the coordination algorithm (that we shall label FTF-EMTF) described in the Planning as Searching for CHiPs section with all of the strategies mentioned in the section, Heurisitics Using Summary Information. It is compared to three other top-down search algorithms that use other heuristics for plan expansion and *or* subplan selection. One algorithm (FAF-FAF) does not employ summary information but rather uses an FAF ("fewest alternatives first") heuristic (Currie & Tate 1991) to decide both the order in which subplans are expanded and the order in which *or* subplans (and search states) are investigated. This simply means it chooses to expand and select plans that have the fewest subplans. Since no summary information is used, threats are only resolved at primitive levels. A second algorithm (DFS-EXCON) uses depth first search to select *or* subplans and derives summary information for an ExCon heuristic to choose the order in which plans are expanded. The ExCon heuristic (Tsuneto, Hendler, & Nau 1998) first selects plans that can achieve an external precondition, or if there are no such plans, it selects one that threatens the external precondition. In the case that there are neither achieving or threatening plans, it chooses randomly. Note that EMTF will additionally choose to expand plans with only threatened external preconditions but has no preference as to whether the plan achieves, threatens, or is threatened. To try and isolate the tradeoffs of EMTF versus ExCon, a third algorithm (FTF-EXCON), that uses FTF for *or* subplan (and search state) selection and ExCon for plan expansion, is compared. While it has been shown that the FAF heuristic can be effectively used by an HTN planner (Tsuneto, Hendler, & Nau 1997), ExCon has been shown to make great improvements over FAF in some domains with more task interactions (Tsuneto, Hendler, & Nau 1998). We show in one such domain that the FTF and EMTF heuristics can together outperform FAF and ExCon.

The problems were hand-crafted from the NEO domain, described in the example in the Introduction. Optimality is measured as the total completion time where each move has a uniform time cost. We chose problems with four, six, eight, and twelve locations; with two, three and four transports; and with no, some, and complete overlap in the locations the transports visit. Performance results for 24 of these problems are given in Figure 2.

The number of search states expanded by each of three of the four algorithms to find solution of the same quality is plotted. Missing data points indicate that the algorithm could find no solution before some threshold of memory and time usage. Data for

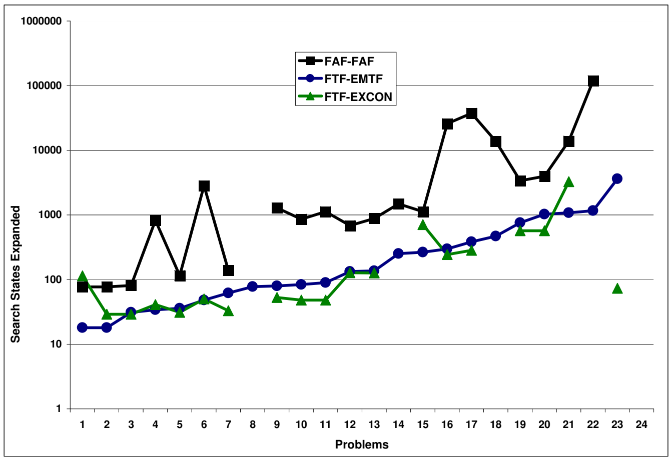

Figure 2: Numbers of search nodes expanded by algorithms searching for optimal solutions for 24 problems from a NEO domain.

DFS-EXCON is not shown because for the most of the problems that it returns solutions, the solutions were of significantly higher cost and would skew the comparison of the others. FTF-EMTF and FTF-EXCON consistently outperformed FAF-FAF by orders of magnitude in most cases. DFS-EXCON had varied performance for the problems for which it was able to find solutions. FTF-EMTF and FTF-EXCON perform similarly for most problems, but there are four problems where FTF-EXCON was unable to find a solution when FTF-EMTF could. FTF-EMTF and FAF-FAF found the optimal solutions for 14 of the problems while FTF-EXCON found 12, and DFS-EXCON found three. FAF-FAF explored many more states than the rest because finding the optimal solutions required near full expansion of the plan hierarchy, and the other algorithms that use summary information incurred significant overhead by recomputing threats to check for abstract solutions at each search state while there was no such overhead for FAF. In fact, FAF-FAF found solutions in less CPU time than the others for most problems. We believe that the algorithm can be significantly streamlined by tracking threats as abstract plans are expanded. So, we expect that as the algorithm is improved, the overhead for using summary information will be dwarfed by the complexity of finding valid orderings of plan steps at lower levels in the hierarchy, and avoiding backtracking through invalid search spaces will allow the algorithms using summary information to run in less time for the more difficult problems. This is true for some of the problems in this experiment.

This experiment shows (for this domain, at least) that FTF is clearly more effective at choosing the selection of *or* subplans and search states than DFS or FAF. The tradeoffs of using EMTF or ExCon are less clear even though EMTF performed well more consistently than the others. A hybrid ExCon-EMTF heuristic may be more effective at expanding plans for this and other domains. Without looking at other domains, conclusions about the performance of FTF and EMTF are difficult to make, but they do show promise for search based on summary information. In addition, the benefits of using higher-level solutions and to prune away more costly ones and ignoring spaces where inconsistent abstract plans are detected deserves further examination.

Conclusions and Future Work

We have shown that summary information can be used by a concurrent hierarchical planner to find solutions at higher levels exponentially more quickly than at lower levels; and heuristics using summary information could make large improvements over previous ones in finding optimal global solutions. Our future work includes further experiments with different domains and a study of the relationship of plan structure to search performance.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Clement, B., and Durfee, E. 1999a. Theory for coordinating concurrent hierarchical planning agents. In *Proc. AAAI*.

Clement, B., and Durfee, E. 1999b. Theory for coordinating concurrent hierarchical planning agents. http://www.eecs.umich.edu/˜bradc/papers/aaai99.

Clement, B., and Durfee, E. 1999c. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proc. Intl. Conf. Autonomous Agents*.

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

Erol, K.; Hendler, J.; and Nau, D. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland.

Erol, K.; Nau, D.; and Hendler, J. 1994. Umcp: A sound and complete planning procedure for hierarchical task-network planning. In *Proc. AIPS*.

Fikes, R. E., and J., N. N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Firby, J. 1989. *Adaptive Execution in Complex Dynamic Domains*. Ph.D. Dissertation, Yale University.

Georgeff, M. P., and Lansky, A. 1986. Procedural knowledge. *Proc. IEEE* 74(10):1383–1398.

Georgeff, M. P. 1983. Communication and interaction in multiagent planning. In *Proc. AAAI*, 125–129.

Huber, M. 1999. Jam: a bdi-theoretic mobile agent architecture. In *Proc. Intl. Conf. Autonomous Agents*, 236–243.

Tsuneto, R.; Erol, K.; Hendler, J.; and Nau, D. 1996. Commitment strategies in hierarchical task network planning. In *Proc. AAAI*.

Tsuneto, R.; Hendler, J.; and Nau, D. 1997. Space-size minimizationin refinement planning. In *Proc. Fourth European Conference on Planning*.

Tsuneto, R.; Hendler, J.; and Nau, D. 1998. Analyzing external conditions to improve the efficiency of htn planning. In *Proc. AAAI*, 913–920.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.