Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information

Bradley J. Clement and Edmund H. Durfee

Artificial Intelligence Laboratory, University of Michigan
1101 Beal Avenue, Ann Arbor, MI 48109-2110, USA
+1-734-764-2138

{bradc, durfee}@umich.edu

Abstract. Recent research has provided methods for coordinating the individually formed concurrent hierarchical plans (CHiPs) of a group of agents in a shared environment. 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 coordination problem to show that in spite of this exponential growth, coordinating CHiPs 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 greatly improve the search for the optimal global plan compared to a "fewest alternatives first" (FAF) heuristic that has been successful in Hierarchical Task Network (HTN) Planning.

1 Introduction

In a shared environment with limited resources, agents may have enough information about the environment to individually plan courses of action but may not be able to anticipate how the actions of others will interfere with accomplishing their goals. Prior techniques have enabled such agents to cooperatively seek merges of individual plans that will accomplish all of their goals if possible [7]. This is done by identifying conflicts and adding synchronization actions to the plans to avoid conflicts. Agents can also interleave planning and merging, such that they propose next-step extensions to their current plans and reconcile conflicts before considering extensions for subsequent steps. By formulating extensions in terms of constraints rather than specific actions, a "least commitment" policy can be retained [5]. In addition, recent research has provided these agents with tools to coordinate their hierarchical plans resulting in more flexible abstract solutions that allow the agents to choose refinements of their actions during execution that can withstand some amount of failure and uncertainty [3]. In addition to adding ordering constraints, agents may need to eliminate choices of subplans for accomplishing subgoals. In order to reason about abstract plans to identify and resolve conflicts, information about how the abstract plans must or may be refined into

This work was supported by DARPA (F30602-98-2-0142).

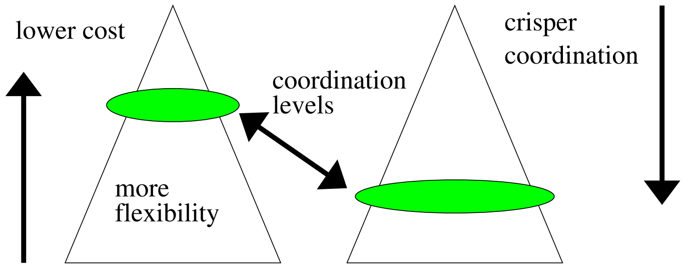


Fig. 1. Hierarchical plan coordination at multiple levels.

lower level actions must be available. This information can be summarized from the conditions of subplans in its potential refinements.

It was previously shown that using this strategy to find abstract solutions to the coordination problem can improve the overall performance of coordinating and executing plans [3]. As depicted in Figure 1, coordination is cheaper at higher levels in the hierarchy because there are fewer plan steps to reason about. Although anecdotal evidence was given to show this, in this paper we reinforce the result with a more rigorous complexity analysis. At lower levels in the hierarchy, however, more detailed solutions of potential greater quality can be found, but only after greater coordination effort. Depending on how costly computation time is compared to the cost of executing the coordinated plans, coordinating at levels in between the top and bottom could likely result in better overall performance. On the other hand, only coordinating at the lowest level can guarantee finding the optimal solution.

If the goal is to find the optimal solution, a reasonable criticism might be that using summary information to reason at abstract levels will be more costly than just coordinating at the lowest level of primitive actions because of the overhead of deriving and using summary information. The experimental results given here contradict this criticism and show how reasoning about plans at abstract levels can better focus the search to much more quickly find detailed solutions at the level of primitive actions.

This paper makes the following contributions:

– complexity analysis showing that finding global plans at higher levels can be exponentially less expensive than at lower levels;
– search techniques and heuristics, including Fewest Threats First (FTF) and Expand Most Threats First (EMTF), that take advantage of summary information;
– a description of a search algorithm that uses these heuristics for coordinating concurrent hierarchical plans; and
– preliminary experiments showing how these heurisitics can greatly save computation time in finding the optimal plan compared to a Fewest Alternatives First (FAF) heuristic [4] that has been successful in Hierarchical Task Network (HTN) Planning [8].

In addition, of potential interest to the planning community, we prove that resolving threats among a set of unordered STRIPS operators is NP-complete. This result is necessary our complexity analysis.

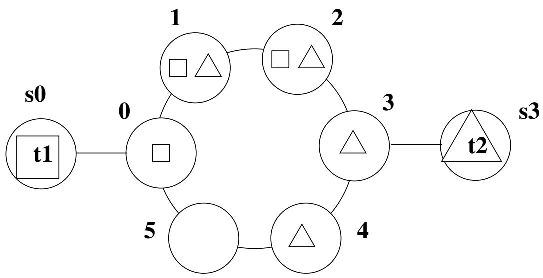


Fig. 2. Transports $t1$ and $t2$ must pick up square and triangle evacuees respectively.

Reasoning about abstract plans with conditions of lower-level subplans has also been used to efficiently guide the search through hierarchical plan spaces (HTN planning) for single agents [9]. This technique 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. We have detailed a procedure for deriving these summary conditions, proofs of their properties, and sound and complete mechanisms for determining legal interactions of abstract plans based on summary conditions elsewhere in [2].

1.1 A Simple Example

This example illustrates how agents can coordinate their actions using summary information to guide the search for a global plan that resolves conflicts and optimizes the total completion time of the agents' plans. In a non-combative evacuation operation (NEO) domain, transport agents 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 an enemy attack), the transports need to coordinate in order to avoid collisions along the single lane routes and must accomplish their goals as quickly as possible.

Suppose there are two transport agents, $t1$ and $t2$, located at safety points $s0$ and $s3$ respectively, and they are responsible for visiting the locations 0-2 and 1-4 respectively as shown in Figure 2. Because there is overlap in the locations they must visit, they must synchronize their individual plans in order to avoid a collision. The hierarchical plan of $t1$ at the highest level is to evacuate the locations for which it is responsible. This decomposes into a primitive action of moving to location 0 on the ring and then to traverse the ring. It can choose to adopt a plan to travel in one direction around the ring without switching directions, or it can choose to switch directions once. $t1$ can then choose to either go clockwise or counterclockwise and, if switching, can choose to 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 can go to the first safety point it finds. $t2$ has a similar plan.

Now let us say $t1$ collects summary information about $t2$'s plan and attempts to coordinate it with its plan. Looking just at the highest level, $t1$ can determine that if it finishes its plan before $t2$ even begins execution, then there will be no conflicts since the external postconditions of its $evacuate$ plan reveal that none of the routes are being traversed. $t1$ then tells $t2$ to add a plan step to the beginning of its plan to $wait$ for $t1$'s signal, and $t1$ can append a $signal$ subplan to the end of its plan. However, this coordinated global plan is inefficient since there is no parallel action—the length ranges from 12 to 26 steps depending on how the agents decompose their plans during execution. If the agents wish to get more concurrency, then they must expand the top-level plans into more detailed plans and resolve conflicts there. At a mid-level expansion where both agents move clockwise without switching directions, the algorithm finds a solution with a length of only eight steps. Now the search algorithm can eliminate the need to resolve threats for any global plan whose length can be no shorter than eight. To find the optimal solution, the agents must almost completely expand their hierarchies. This is a plan of length seven where $t1$ moves clockwise until it reaches location $s3$, and $t2$ starts out clockwise, switches at location 4, and then winds up at $s0$.

1.2 Overview

In the next section, we describe how concurrent hierarchical plans can be coordinated using 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 coordination algorithm. 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.

2 Top-Down Coordination of Concurrent Hierarchical Plans

Our approach to coordinating concurrent hierarchical plans (CHiPs) is to first try to coordinate the plans at the top-level of the hierarchies, then consider their subplans, and iteratively expand selected subplans until a "feasible" solution is found. A general algorithm for doing this is described in [3]. Here we briefly explain the basic mechanisms for deriving and using summary information and then describe a specific algorithm we use to evaluate the effectiveness of coordinating using summary information. All terms and mechanisms mentioned here are formalized in [2].

2.1 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 [1] or point relations over endpoints of plan execution time intervals. A primitive plan is only different in that it has an empty set of subplans. In the style of STRIPS planning

external preconditions the conditions that must be met external to the execution of an abstract plan for any decomposition of the plan in order for the execution to succeed

external postconditions the effects of a successful execution of an abstract plan for any decomposition of the plan that are not *undone* by its execution (which includes any execution of subplans in its decomposition)

summary preconditions the external preconditions computed for an abstract plan along with $existence$ ($must$ or may) information for each condition[2]

summary inconditions the internal conditions computed for an abstract plan that include any required conditions or effects that must hold within the interval of execution for some decomposition of the plan along with $existence$ ($must$ or may) and $timing$ ($always$ or $sometimes$) information for each condition

summary postconditions the external preconditions computed for an abstract plan along with $existence$ ($must$ or may) information for each condition[2]

must property of a summary condition where the condition must hold for the execution of the plan for *any* decomposition

may property of a summary condition where the condition must hold for the execution of the plan for *some* decomposition

always property of a summary incondition where the condition must hold throughout the execution of the plan for any decomposition

sometimes property of a summary incondition where the condition must hold at some point during the execution of the plan for some decomposition

clobber the effects of one plan's execution negates a condition required by another plan causing the plan to fail

achieve the effects of one plan's execution asserts a condition required for another plan to be successful

undo the effects of one plan's execution negates a condition asserted by another plan

CanAnyWay($plans$, $order$) every plan in $plans$ can be decomposed and executed in any way according to the ordering constraints in $order$, and all executions will succeed

MightSomeWay($plans$, $order$) there is some decomposition and execution of each plan in $plans$ according to the ordering constraints in $order$ such that all executions succeed

Fig. 3. A review of terminology formalized in [2].

operators [6], each of these plans has sets of preconditions and effects.[1] However, since we necessarily worry about agents 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.

An agent's plan library is a set of CHiPs, any of which could be part of the agent's current plan, and 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 Section 1.1, 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.

[1] These are not summary conditions.

2.2 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. As mentioned in the Section 1, 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.

2.3 Temporal Interactions of CHiPs

In conventional planning, we often speak of *clobbering* and *achieving* preconditions of plans [10]. 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 across the plan hierarchies of the agents. For example, plan p of one agent cannot clobber a condition c of plan q of another agent if there is another plan r ordered between p and q that achieves c for q. However, if plan r only may achieve c because c is a may postcondition of r, then p threatens q. Reasoning about these kinds of interactions, we can determine that a set of temporal relations can hold among plans no matter how they are decomposed ($CanAnyWay$) or that certain relations cannot hold for any decomposition ($\neg MightSomeWay$). As the procedure for determining these relations is similar to propagating summary information, its complexity is also $O(n^2c^2)$ for n plans with c conditions each [2].[3]

2.4 Top-Down Search Algorithm

Since an agent can determine whether a set of the agents' abstract plans $CanAnyWay$ or $MightSomeWay$ be executed successfully under particular ordering constraints, we can integrate this into an algorithm that smartly searches for a consistent global plan for a group of agents. The particular algorithm we describe here is sound and complete

[3] The algorithm for determining $\neg MightSomeWay$ looks at all pairs of plans to detect if one must clobber another. This is not a complete algorithm because it does not consider impossibilities of satisfying combinations of threatened conditions. In Section 3 we show that this is an intractable problem.

and returns the optimal global plan if it exists. The search starts out with the top-level plans of each agent, which together represent the global plan. The algorithm tries to find a solution at this level and then expands the hierarchies deeper and deeper until the optimal solution is found or the search space has been exhausted. A pseudocode description of the algorithm is given later in this section.

A state of the search is a partially elaborated global plan that we represent as a set of and plans (one for each agent), 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 in addition to those dictated by the agents' individual hierarchical plans. Blocked subplans keep track of pruned or subplans.

While one agent can be responsible for coordinating the plans of all agents in the system, the agents can use the summary information to determine which subsets of agents have locally conflicting plans and designate a coordinator for each localized group. In addition, the decisions made during the search could be made decentrally. The agents can negotiate over ordering constraints imposed, choices subplans to accomplish higher level plans, and which decompositions to explore first. While the algorithm described here does not comment specific negotiation techniques, it does provide the mechanics for identifying the choices over which the agents can negotiate.

The operators of the search are expanding non-primitive plans, blocking or subplans, and adding temporal constraints on pairs of plans. When the agents expand one of their plans, 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 the multiple applicable remaining subplans. The agents can take another approach by selecting subplans (effectively blocking the others) to investigate choices that are given greater preference or are more likely to resolve conflicts.

In the pseudocode below, the coordinating agent collects summary information about the other agents' plans as it decomposes them. The $queue$ keeps track of expanded search states. If the $CanAnyWay$ relation holds for the search state, the Dominates function determines if the current solutions are better for every agent than the solution represented by the current search state and keeps it if the solution is not dominated. If $MightSomeWay$ is false, then the search space represented by the current search state can be pruned; otherwise, the operators mentioned above are applied to generate new search states. Nondeterministic "Choose" functions determine how these operators are applied. Our implementation uses heuristics specified in Section 2.5 to determine what choices are made. When a plan is expanded or selected, the ordering constraints for that plan must be updated for the subplans that replace it. The Update-Order function accomplishes this.

Hierarchical Plan Coordination Algorithm

$plans = \emptyset$
for each agent a_i
    p_i = get summary information for top-level plan of a_i

```
    plans = plans ∪ {p_i}
end for
queue = {(plans, ∅, ∅)}
solutions = ∅
loop
    if queue == ∅
        return solutions
    end if
    (plans, order, blocked) = Pop(queue)
    if CanAnyWay(initial_state, plans, order, blocked)
        solution = (plans, order, blocked)
        if Dominates(solutions, solution) == false
            solutions = solutions ∪ {solution}
        end if
    end if
    if MightSomeWay(initial_state, plans, order, blocked)
        operator = Choose({expand, select, block, constrain})
        if operator == expand
            plan = ChooseAndPlan(plans)
            if Exists(plan)
                plan.subplans = get summary information for
                    subplans of plan
                plans = plans ∪ plan.subplans - plan
                UpdateOrder(order, plan, plan.subplans, plan.order)
            end if
        end if
        if operator == select
            plan = ChooseOrPlan(plans)
            if Exists(plan)
                plan.subplans = get summary information for
                    subplans of plan
                for each subplan ∈ plan.subplans
                    newblocked = blocked ∪ plan.subplans - {subplan}
                    newplans = plans ∪ {subplan} - plan
                    neworder = order
                    UpdateOrder(neworder, plan, {subplan}, ∅)
                    InsertStateInQueue(queue, newplans, neworder,
                        newblocked)
                end for
            end if
        end if
        if operator == block
            plan = ChooseOrPlan(plans)
            if Exists(plan)
                plan.subplans = get summary information for
                    subplans of plan
                for each subplan ∈ plan.subplans where subplan ∉ blocked
                    newblocked = blocked ∪ subplan
                    neworder = order
                    if ∃! subplan' ∈ plan.subplans, subplan' ∉ blocked
                        newplans = plans ∪ {subplan'} - plan
                        UpdateOrder(neworder, plan, {subplan'}, ∅)
                    else
                        newplans = plans
                    end if
```

```
            InsertStateInQueue(queue, newplans, neworder,
                newblocked)
        end for
    end if
end if
if operator == constrain
    plan = ChoosePlan(plans)
    plan' = ChoosePlan(plans - {plan})
    constraint = ChooseConstraint({Start, End} ×
        {<, ≤, =, ≥, >} × {Start, End})
    neworder = order ∪ constraint
    if Consistent(neworder)
        InsertStateInQueue(queue, plans, neworder, blocked)
    end if
end if
end if
end loop
```

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 performs the work of merging non-hierarchical plans since it is used to find a synchronization of the individual agents' plans that are one level deep. In the pseudocode above, the ChooseConstraint function nondeterministically investigates all orderings (represented by point algebra constraints over the "Start" and "End" points of action intervals), and inconsistent ordering constaints are pruned. However, in our implementation, we only investigate legal ordering constraints that resolve threats that are identified by algorithms determining must/may achieves and clobbers relations among CHiPs. 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 merges by resolving threats with ordering constraints.

The soundness and completeness of the coordination algorithm depends on the soundness and completeness of identifying solutions and the complete exploration of the search space. Each search state is tested by the $CanAnyWay$ procedure to determine whether it is a solution. The $CanAnyWay$ procedure is shown to be sound and complete in [2]. Although the algorithm for determining $\neg MightSomeWay$ is only complete for a total ordering of CHiPs, it is used to prune invalid branches in the search space, so it is enough that it is sound [2]. In order to explore the search space completely, the coordinator would need to consider all synchronizations of all possible decompositions of each of the agents' top-level plans. We assume that the plan hierarchy of each agent is finite in its decomposition, so when the coordinator nondeterministically expands abstract plans, eventually all abstract plans will be replaced with primitive decompositions. Likewise, eventually all or plans will be replaced with subplan choices, and since new search states are generated and added to the queue for each subplan of an or plan, all possible decompositions of the agents' top-level plans are explored. The Choose function for selecting operators nondeterministically explores any synchronization of the expanded plans in conjunction with the ChooseConstraint function, so the search is complete.

2.5 Heuristics Using Summary Information

As discussed in [3], summary information is valuable for finding coordinated plans 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 global plans. Inconsistent global plans can be pruned away at the abstract level by doing a quick check to see if $MightSomeWay$ is false. In terms of the number of states expanded during the search, employing this technique will always do at least as well as not using it. 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. So, EMTF is a heurisitic ordering plans to expand, and FTF orders subplan choices and, thus, search states to investigate. In addition, in trying to find optimal solutions in the style of a branch-and-bound search, 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. Again, pruning abstract plans can only help improve the search. We report experimental results in Section 4 that show that these techniques and heuristics can greatly improve coordination performance.

3 Complexity

In [3], 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 is expected to be exponentially cheaper than at lower levels.

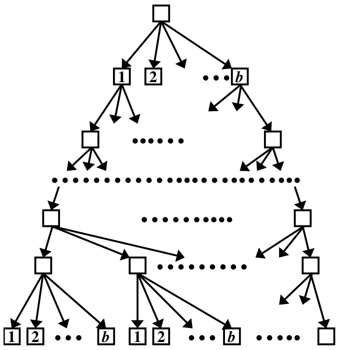
The procedure for deriving summary conditions works by basically propagating the conditions from the primitives up the hierarchy to the most abstract plans. Because the conditions of any non-primitive plan depend only on those of its immediate subplans, deriving summary conditions can be done quickly. In [2], it was reported that the complexity of this is $O(n(log^2 n)c^2)$ 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. If c' is the greatest number of literals in any plan's pre-, in-, and postconditions, then the complexity is $O(n^2 c'^2)$.

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. In this way, the maximum number of conditions are propagated up the hierarchy and all of the expanded plans must be synchronized to avoid conflicts. Consider a global hierarchy with n total plans, b subplans for each non-primitive plan, and depth d.[4] At each level, the procedure tests each condition in each summary condition set of the b subplans of each plan at that level to see if they are achieved/clobbered/undone by any other subplan attempting to assert that condition. 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 for each plan with b subplans each having $O(c)$ summary conditions. So, as shown in Figure 4, at the next-to-bottom depth level $d-1$, each of the b^{d-1} plans has b primitive subplans each with $O(c')$ conditions. Thus, $O(b^2c'^2)$ operations are performed for each of the b^{d-1} plans for a total of $O(b^{d-1}b^2c'^2)$ operations for that level. At level $d-2$, there are b^{d-2} plans, and the number of conditions that must be compared among their subplans at level $d-1$ additionally includes those propagated from the primitive level for a total of $3c' + b3c'$ conditions. Thus, $O(b^{d-2}b^2(c' + bc')^2)$ operations are performed at level $d-2$. This generalizes to $O(\sum_{i=0}^{d-1} b^i b^2 (b^{d-i-1}c')^2)$ operations for the entire hierarchy. We can reduce this to $O(b^2c'^2 \sum_{i=0}^{d-1} b^{2d-i-2}) = O(b^{2d}c'^2)$, 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 as shown in the second column of Figure 4. At the primitive level d, each plan has only $3c' = O(c')$ conditions, and there are $c' + bc' = O(bc')$ summary conditions for each plan at level $d-1$ and $O(b^2c')$. There are at most $3nc'$, 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 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, as will be shown, 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 yield 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 [2], the algorithm to check whether a particular ordering of n plan steps (each with c summary conditions) results in all plans executing successfully is similar to deriving their collective summary information and has a complexity of $O(n^2c^2)$. 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

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



level	#plans	#conds / plan	#operations to derive summ. info.	#test operations / solution candidate	solution space
0	1	$O(b^d c')$	$O(b^2(b^{d-1}c')^2) = O(b^{2d}c'^2)$	$O(1)$	1
1	b	$O(b^{d-1}c')$	$O(bb^2(b^{d-2}c')^2) = O(b^{2d-1}c'^2)$	$O(b^2(b^{(d-1)}c')^2) = O(b^{2d}c'^2)$	$O(b!)$
2	b^2	$O(b^{d-2}c')$	$O(b^2b^2(b^{d-3}c')^2) = O(b^{2d-2}c'^2)$	$O(b^4(b^{(d-2)}c')^2) = O(b^{2d}c'^2)$	$O(b^2!)$
d-2	b^{d-2}	$O(b^2c')$	$O(b^{d-2}b^2(bc')^2) = O(b^{d+2}c'^2)$	$O(b^{2(d-2)}(b^2c')^2) = O(b^{2d}c'^2)$	$O(b^{d-2}!)$
d-1	b^{d-1}	$3c'+b3c' = O(bc')$	$O(b^{d-1}b^2c'^2) = O(b^{d+1}c'^2)$	$O(b^{2(d-1)}(bc')^2) = O(b^{2d}c'^2)$	$O(b^{d-1}!)$
d	b^d	$3c'$	$O(1)$	$O(b^{2d}c'^2)$	$O(b^d!)$

Fig. 4. The table gives the number of plans and summary conditions for each plan at some level of expansion of a global plan hierarchy (with branching factor b) where each plan has c' conditions in each set of pre-, in-, and postconditions. The number of operations to derive summary information for all of the plans at a particular depth level is the product of the number of plans at that level, the square of the number of subplans per plan, and the square of the number of conditions per subplan. The number of operations to check if a candidate expansion under particular ordering constraints is a solution is on the order of the square of the product of the number of plans and the number of summary conditions per plan at that level of expansion. The solution space is the number of temporal orderings of the expanded plans (approximated by the factorial).

is *independent of the depth level*. In Figure 4, this is shown in the fifth column of the table where the number of operations is the same at each level. But, there is a huge space of $n! = O((b^i)!)$ sequential orderings[5] of the n plans at level i to potentially check to find a valid synchronization.[6] 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 for larger numbers of plan steps, so we show that it is actually NP-complete. We reduce HAMILTONIAN PATH to the THREAT RESOLUTION problem for STRIPS planning and claim that a similar reduction can be done for our problem that allows concurrent execution.

Theorem THREAT RESOLUTION is NP-complete. This is the problem of determining whether there is a set of ordering constraints that can be added to a partial order

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

[6] This is why Georgeff[7] chose to cluster multiple operators into "critical regions" and synchronize the (fewer) regions since there would be many fewer interleavings to check. By exploiting the hierarchical structure of plans, we use the "clusters" predefined in the hierarchy to this kind of advantage without needing to cluster from the bottom up.

STRIPS plan such that no operator's preconditions are threatened by another operator's effects.

Proof If there is a set of ordering constraints that will resolve all threats, then there is at least one corresponding total order where there are no threats. Thus, the problem is in NP since orderings of operators can be chosen non-deterministically, and threats can be identified in polynomial time.

Given a directed graph $G = (V, E)$ with nodes $v_1, v_2, \ldots, v_n \in V$ and edges $e_1, e_2, \ldots, e_m \in E$ (a set of ordered pairs of nodes), HAMILTONIAN PATH is the problem that asks if there is a path that visits each node exactly once. We build an instance of THREAT RESOLUTION (a partial order plan) by creating an operator for each node v_i. The only precondition of the operator is $A(i)$, representing the *accessibility* of the node. There is a postcondition $A(j)$ for each edge $e_k = (v_i, v_j)$, and a postcondition $\overline{A(l)}$ for all other nodes for which there is no edge from v_i. All operators are unordered and the initial state and goal state is empty.

If there is a Hamiltonian path for the graph, then the operators for the nodes can be ordered the same as the nodes in the path because the accessibility preconditions of each operator will be satisfied by the previous operator. If there is no Hamiltonian path for the graph, then there is no consistent ordering of the operators. We know this because there is a one-to-one mapping from an ordering of nodes to an ordering of operators. If the ordering of the nodes is such that there is no edge from one to a succeeding node, then the accessibility precondition of the corresponding operator will be clobbered. In addition, for any walk through the graph, there eventually will be an unvisited node for which there is no edge from the last node visited. In this case, the unvisited node will be clobbered because its accessibility precondition will not be met. Thus, THREAT RESOLUTION is NP-hard, and since it was shown to be in NP, it is NP-complete. □

In order to show that resolving threats among CHiPs is also NP-complete, we only need to add inconditions to each operator that prevent concurrent action. This can be done by adding $A(i)$ for v_i and $\overline{A(j)}$ for every other $v_j \in V$ to the inconditions of the operator corresponding to v_i for each $v_i \in V$. This ensures that the only temporal relations that can hold between any pair of operators are $before$, $after$, $meets$, or $imeets$, and the one-to-one mapping from paths in the graph to sequences of operators is preserved.

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.

4 Experiments

The experiments described here used the coordination algorithm described in Section 2.4 with all of the stated heuristics. It was compared to another top-down search algorithm that did not use summary information but used a FAF ("fewest alternatives first") heuristic [4] to decide the order in which or subplans are investigated. This simply means we chose to expand the or subplan that had the fewest number of subplan choices. Since no summary information was used, threats could only be resolved at primitive levels. The FAF heuristic has been shown to be effective in the HTN planning domain to get large improvements in search time [8], and a similar approach to ours shows how heuristics using external conditions can be used to get exponential improvements over FAF [9]. We show here that summary information can also be used to gain significant improvements over FAF. Certainly, a comparison of our approach with that in [9] could help shed light on the benefits and disadvantages of varying amounts of summary information. This is a future consideration of this work.

The problems were hand-crafted from the NEO domain, described in the example in the Introduction. Agents had plans to either visit their specified locations by traveling in one direction only or switching directions at some location. These choices expand into choices to begin traveling clockwise or counterclockwise. For the branch where the agent switches directions, it can choose to change directions at any location it is specified to visit. Primitive actions are to move between adjacent locations without running into another agent. Optimality is measured as the total completions time where each move has a uniform time cost. We chose problems with four, six, and eight locations; with two and three agents; and with no, some, and complete overlap in the locations the agents visited. Results of the experiments are given in Figure 5.

For problems with only four locations and two agents, both algorithms found the optimal solution quickly. For more complex algorithms, the heuristics using summary information appear to make great improvements over FAF, which could only solve six of the 21 problems within memory constraints. These results are by no means conclusive, but they do show promise for search based on summary information. For most of these problems, coordinating at the primitive level was intractable. In most cases, the algorithm using summary information was able to find an abstract solution quickly.

5 Conclusions and Future Work

We have shown that summary information can can find solutions at higher levels exponentially more quickly than at lower levels; and we have identified heuristics and search techniques that can take advantage of summary information in finding coordinated plans. In addition, we have characterized a coordination algorithm that takes advantage of these search techniques and experimentally shown how it can make large improvements over an FAF heuristic in finding optimal coordinated plans. More work is needed to show that these results translate to different domains, and future considerations include comparing this approach to other planning heuristics that capitalize on domain knowledge in order to better understand the relationship between plan structure and search performance. We expect the benefits of using summary information to

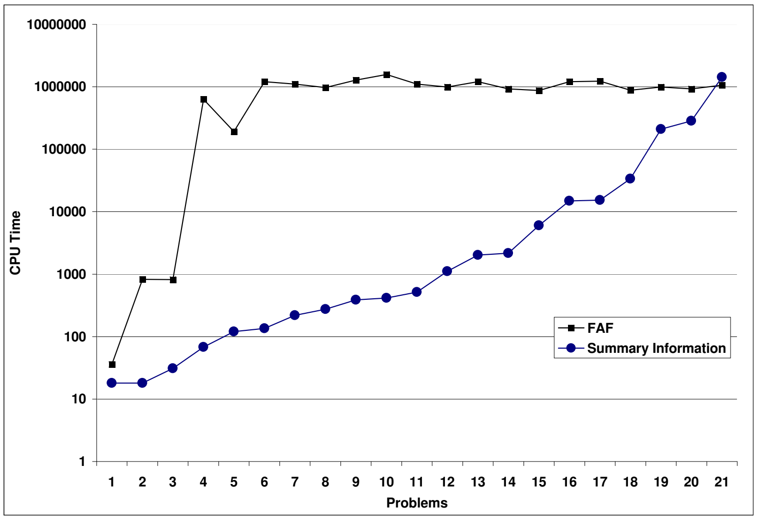


Fig. 5. CPU time measurements comparing summary information heuristics to FAF for finding optimal solutions. FAF only solved problems 1-5 and 7; others were killed when the search queue was too large to fit in memory.

also apply to hierarchical planning and wish to compare these techniques with current heuristics for concurrent hierarchical planning.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.
2. B. Clement and E. Durfee. Theory for coordinating concurrent hierarchical planning agents. In *Proc. AAAI*, 1999.
3. B. Clement and E. Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proc. Intl. Conf. Autonomous Agents*, 1999.
4. K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.
5. E. Ephrati and J. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI*, pages 375–380, July 1994.
6. R. E. Fikes and Nilsson N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
7. M. P. Georgeff. Communication and interaction in multiagent planning. In *Proc. AAAI*, pages 125–129, 1983.
8. R. Tsuneto, J. Hendler, and D. Nau. Space-size minimizationin refinement planning. In *Proc. Fourth European Conference on Planning*, 1997.
9. R. Tsuneto, J. Hendler, and D. Nau. Analyzing external conditions to improve the efficiency of htn planning. In *Proc. AAAI*, pages 913–920, 1998.
10. D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.