Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents

Bradley J. Clement and Edmund H. Durfee
Artificial Intelligence Laboratory
University of Michigan
Ann Arbor, MI 48109
+1-734-764-4317
{bradc,durfee}@umich.edu

ABSTRACT

Uncertain and complex environments demand that an agent be able to anticipate the actions of others in order to avoid resource conflicts with them and to realize its goals. Conflicts during plan execution can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents and by merging/coordinating the individual plans of agents by introducing synchronization actions. We describe a method for coordinating plans at abstract levels that takes advantage of hierarchical representations of plan information and that retains the flexibility of plans used in robust plan execution systems such as procedural reasoning systems (PRS). In order to coordinate at abstract levels in plan hierarchies, information about how abstract plans can be refined must be available in order to identify and avoid potential conflicts. We address this by providing procedures for deriving summary information for non-primitive plans that capture the external preconditions and effects of their refinements. We also describe a general search algorithm and an implementation to show how to use this information to coordinate hierarchical plans from the top down to primitive actions.

1. INTRODUCTION

Uncertain and complex environments demand that an agent be able to anticipate the actions of others in order to avoid resource conflicts with them and to realize its goals. Conflicts during plan execution can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents [12] and by merging/coordinating the individual plans of agents by introducing synchronization actions [10].

Just as agents can enhance planning efficiency by exploiting the hierarchical structure of planning operations, they can similarly enhance the efficiency and quality of coordinating their plans. Flexible plan execution systems such as PRS [11], RAPS [9], etc. interleave plan execution with the refinement of abstract plans into specific actions. By postponing refinement until absolutely necessary, such systems leave themselves flexibility to choose refinements that best match the current circumstances. However, when sharing resources with other agents, it may also be necessary to look ahead and avoid refining abstract plans into immediate actions that ultimately lead to irreversible deadlock or failure. If agents know about each others abstract plans and how those plans could possibly be refined, then such potential coordination errors could be avoided by, for example synchronizing (merging) abstract plans.

As illustrated in Figure 1, coordinating plans at abstract levels can cut computation costs because such plans are typically smaller. If done properly, coordinating at abstract levels also means that each agent can refine its abstract plan independently, since any dependencies have been resolved at the abstract level. However, coordination at abstract levels can impose commitments on agents that unnecessarily restrict and sequentialize their activities, as we shall see. Thus, there is a tradeoff between cost and flexibility against the efficiency of a coordinated outcome. Whereas Lansky offered techniques to coordinate the plans at the most abstract level [12], and Georgeff [10] and Ephrati and Rosenschein [6] described methods for merging plans at the primitive level, the work we describe in this paper combines these strategies such that plan coordination can be done flexibly at the levels in-between.

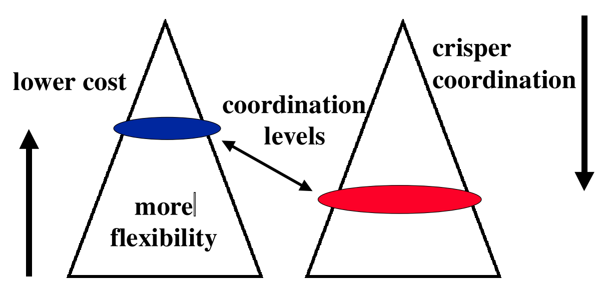

Figure 1: Hierarchical plan coordination at multiple levels

Because coordinating at abstract levels involves making decisions that cover the underlying plan steps, some information about how abstract plans can be refined must be available in order to identify and avoid potential conflicts. In this paper, we describe procedures for deriving summary information for non-primitive plans that capture the *external preconditions* [14] and *effects* of their refinements and show how to use this information to coordinate hierarchical plans from the top down to primitive actions. The idea is that agents can summarize the conditions of subplans for all possible refinements of a non-primitive plan offline. Then, when coordination is needed during execution, the agents can take a snapshot of their current plans, and agents

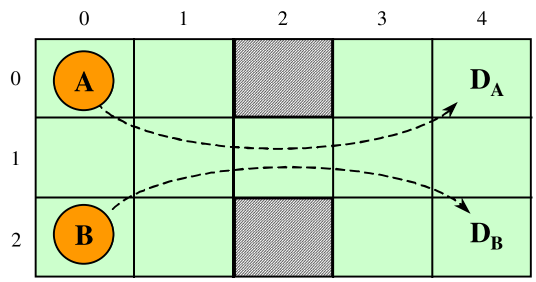

Figure 2: Two agents moving within a grid world.

responsible for coordination can request the summary information of interacting agents to determine how non-primitive plans can definitely or might possibly "safely" interact with others. This capability, in turn, is integrated into a general search algorithm for coordinating hierarchical plans by determining what refinement choices should be eliminated and how abstract or primitive actions should be ordered.

1.1 A Simple Example

Suppose that two agents wish to go through a doorway into another room as shown in Figure 2. Agent A has a hierarchical plan *p* (shown later in Figure 3) to move from (0,0) to (0,4), and B also has a plan *q* to move from (2,0) to (2,4), but they need to coordinate their plans to avoid collision. Agent A could have preprocessed plan *p* to derive its summary information. The set of *summary preconditions* of *p* includes all its preconditions and those of its subplans that must be met external to *p* in order for *p* to execute successfully: ({*At*(A,0,0), ¬*At*(*?*,0,1), ¬*At*(*?*,1,0), . . ., ¬*At*(*?*,0,4)}). *At*(A,0,0) is called a *must* condition because no matter how *p* is executed, the condition must hold. ¬*At*(*?*,1,0) is *may* because it may be required depending on the path A takes. Likewise, the *summary postconditions* of *p* are its effects and those of its subplans that are seen externally: ({*At*(A,0,4), ¬*At*(A,0,0), ¬*At*(A,1,0), . . .}). The *summary inconditions* are any conditions that must hold within the interval of time that the plan is executing and can be *must* or *may* and *always* or *sometimes*. An *always* condition is required to hold throughout the duration of any execution of the plan. These conditions and descriptors, such as *must* and *always*, provide the necessary information to reason about what conditions must or may be achieved or clobbered when ordering the executions of a set of plans.

Now suppose A sends B the summary information for *p*. Agent B can now reason about the interactions of their plans based on their combined summary information. B can then determine that if *p* were restricted to execute before *q*, then the plans *can* be refined in *any way*, or *CanAnyWay*(b, p_{sum}, q_{sum}). (Here, b is an abbreviation for the before relation, one of thirteen temporal relations described by Allen in [1], and p_{sum} and q_{sum} are the sets of summary conditions for *p* and *q*.) So, B could tell A to go ahead and start execution and send back a message when *p* is finished executing. However, B may wish to overlap their plan executions for better efficiency. Although ¬*CanAnyWay*(o, p_{sum}, q_{sum}), the summary conditions could be used to determine that there *might* be *some way* to overlap them, or *MightSomeWay*(o, p_{sum}, q_{sum}). Then, B could ask A for the summary information of *p*'s subplans, reason about the interactions of lower level subplans in the same way, and find a way to synchronize the subplans for a more fine-grained solution.

This approach builds on ideas of using abstract information in hierarchical planning. Corkill recognized a need for such abstract plan information in his study of interleaved planning and merging in a distributed version of an HTN planner [3]. Tsuneto *et. al.* also discovered the value of identifying such *external conditions* for HTN planning and showed that they can help prune the search space and avoid backtracking to improve planning efficiency [].

A similar approach is a hierarchical behavior-space search where agents represent their planned behaviors at multiple levels of abstraction [3]. Distributed protocols are used to decide at what level of abstraction coordination is needed and to resolve conflicts there. This approach capitalizes on domains where resources can be abstracted naturally, but we wish to deal with resource conflicts generally through popular plan representations.

1.2 Overview

In Section 2.1 we present a procedure that derives summary conditions. This information is used in Section 2.2 to describe *CanAnyWay* and *MightSomeWay* rules that are used to determine how plans can or might be temporally related without looking at how they could be refined. Section 3 shows how hierarchical plan coordination algorithms can exploit these rules to search through the space of global hierarchical plans to resolve conflicts and retain the robustness of the individual plans of the agents. We also give examples of how an implemented instance of the general algorithm coordinates plans. We evaluate this approach by comparing it to replanning for the combined goals of agents in Section 4 and by showing the tradeoffs of coordinating at different levels of abstraction under different cost scenarios.

2. USING SUMMARY INFORMATION FOR HIERARCHICAL PLANS

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. A primitive plan is only different in that it has an empty set of subplans. In the style of STRIPS planning operators [8], each of these plans has sets of preconditions and effects. However, since we necessarily worry about agents performing tasks in parallel, we also associate a set of *inconditions*, also called *during* conditions [10], with each plan so that threats during the execution of a task can be represented. Preconditions must hold at the beginning of execution; postconditions are effects that must hold at the endpoint of execution; and inconditions must hold throughout execution.

An agent's plan library is a set of plans, any of which could be part of the agent's current plan, where each plan can play a role in a plan hierarchy as 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.

This decomposition is in the same style as an HTN, as described by Erol *et. al.* [7], but 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 non-primitive task. So, a "hierarchical plan" is an HTN transformed into an *and/or* tree of subplans with primitives at the leaves. Although the plan coordination approach taken here is intended to work for HTNs, we assume that the plans have the downward refinement property [2], and HTN planning is only necessary for plans without this property.

Procedures used by procedural reasoning systems (PRSs [11]) specify contexts under which certain goals are satisfied. They can also introduce subgoals to be matched by other plans. The context of the procedure is equivalent to the preconditions of hierarchical plans since the plan execution is not attempted unless the context satisfied. Some PRSs also keep track of the effects of procedures (the postconditions), but for those that do not, capturing the post- and inconditions of a procedure is potentially difficult given looping constructs; handling this is beyond the scope of this paper. Conditional branches and subgoals can be captured by the and/or structure of hierarchical plans.

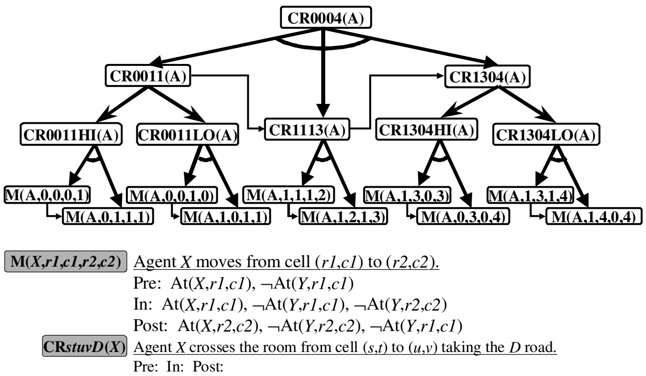

Figure 3: Agent A's hierarchical plan

2.1 Deriving Summary Conditions

With this specification of hierarchical plans, it is obvious that the needs and net effects of a non-primitive plan depend on the conditions of the subplans in its decomposition, in addition to its own specified conditions. If we want to understand how a plan can or might be coordinated with another, we need to know what preconditions and effects must or may hold *external* to the plan's execution as well as those conditions that must or may hold *internally*. We now describe a method for deriving this information in the form of what we call *summary conditions* as described in the example in Section 0. In addition to the *must*, *always* descriptors, pre- and postconditions are labeled *first* and *last* if they must hold at the beginning/end of execution, respectively, to help distinguish summary pre- and postconditions from summary inconditions in the parent plan. The following rules explain how conditions are propagated from the primitives up a hierarchy:

- For each plan, its sets of summary conditions include the respective pre-, in-, and postconditions described for the plan. All those conditions are *must*; those [pre, post]conditions are [*first*, *last*]; and those inconditions are *always*.

<u>Summary [pre, post] conditions of *and* plans</u>

- The set of summary [pre, post]conditions of *and* plan *p* additionally includes all summary [pre, post]conditions in *p*'s subplans that cannot be [achieved, clobbered] by a *must* summary condition of another subplan step. (These are also the conditions in the first column of a STRIPS triangle table [13].)
- A summary [pre, post]condition, *c*, is *must* if any of the subplans' corresponding summary [pre, post]conditions is *must*, and no other subplan has a summary condition that may [achieve, clobber] *c*; otherwise, it is *may*.
- Condition *c* is [*first*, *last*] iff a [least, greatest] temporally ordered plan (no others are constrained to [begin before, end after] it) has an equivalent [*first*, *last*] summary precondition. (This corresponds to the conditions in the first column and first row of a triangle table.)

<u>Summary [pre, post]conditions of *or* plans</u>

- The set of summary [pre, post]conditions of *or* plan *p* additionally includes the union of the [pre, post]conditions of *p*'s subplans.
- If a summary [pre, post]condition, *c*, of *p* is a *must* summary [pre, post]condition of all of *p*'s subplans, then it is *must*; otherwise, it is *may*.
- *c* is *first* iff it is *first* in a subplan.

<u>Summary inconditions of *and* plans</u>

- The set of summary inconditions of *and* plan *p* additionally includes the union of the summary [pre, post]conditions of all of the subplans except the [*first*, *last*] in the [least, greatest] temporally ordered subplan and the summary inconditions of all of the subplans.
- A summary incondition, *c*, of *p* is *must* if there is a corresponding *must* condition in a subplan of *p*; otherwise, it is *may*.
- *c* is *always* if it is *always* in every subplan; otherwise, it is *sometimes*.

<u>Summary inconditions of *or* plans</u>

- The set of summary inconditions of *or* plan *p* is the union of the inconditions of the subplans.
- If a summary incondition, *c*, of *p* is a *must* summary incondition of all of the subplans, then it is *must*; otherwise, it is *may*.
- *c* is *always* if it is *always* in every subplan; otherwise it is *sometimes*.

Note that the set of summary preconditions derived for a non-primitive plan is the set of *external preconditions*—those that must be achieved outside of the plan. They do not necessarily need to hold before execution of the plan begins; only the *first* preconditions need to hold at the beginning. These external preconditions are similar to the *external conditions* described for HTNs by Tsuneto *et. al.* [14]. Similarly, the summary postconditions are external effects in that effects of subplans that are undone by other subplans are not included, and only *last* postconditions need to hold at the end of the plan.

When propagating conditions, information about which condition went with which subplan is lost. This means that, without looking at subplans, there is no way to tell which *may* conditions are part of the same choice of subplans or when a *sometimes* condition holds. However, as shown in the next section, the information is still valuable in determining what temporal relations the plan can definitely or might potentially have with another.

2.2 Determining Temporal Relations

Using the summary information of two plans, we can define rules that indicate how the plans can or might be related. As introduced in Section 0, these *CanAnyWay* and *MightSomeWay* predicates can be defined for each of Allen's thirteen temporal relations for two time intervals: before, meets, overlaps, equals, during, starts, finishes, and the inverses of all but equals [1]. We say a temporal relation *holds* for two plans if the ordering does not cause any condition of a plan to be clobbered, which we assume causes the plan to fail. *CanAnyWay*(*rel*, p_{sum}, q_{sum}) is the relation where the temporal relation *rel can* hold for any plans *p* and *q* whose summary information is p_{sum} and q_{sum} for *any way* that they may be refined and their subplans interleaved. *MightSomeWay*(*rel*, p_{sum}, q_{sum}) is the relation where *rel might* hold for *some way* of executing *p* and *q*. More specifically, there is some *p* and *q* whose summary information is p_{sum} and q_{sum}, and there is some way of decomposing and synchronizing them such that they both execute successfully. (For convenience, we will abbreviate these relations as *CAW* and *MSW*.)

So, given only the summary information of a pair of plans, the goal is to determine either that the plans can definitely be related in certain ways (*CAW*) or that there is no way to relate them in certain ways (¬*MSW*). If *CAW* is true, a "safe" coordination decision can be made. If *MSW* is false, searching for a way to coordinate the plans at a deeper level should be avoided since no such coordination method exists. If we can determine neither, then the summary information is of no value because without *any* information we could say that there might be some way of relating two plans. For example, an *or* plan *p* may have two primitive subplans, p_1 and p_2. Suppose p_1 has a postcondition *v*, and p_2 has a postcondition ¬*v*. This means *p* has *may* summary postconditions *v* and ¬*v*. For a plan *q* with a *must* summary incondition *v*, there is no way that *p* could be executed during *q* since there would have to be a conflict with *q*'s summary inconditions. But, we cannot determine that from the summary information because there are plans that have the same summary information as *p* that can some way be executed during *q*. In other words, *MightSomeWay*(during, p_{sum}, q_{sum}) is true, so the summary information says nothing about whether *p* can be executed during *q*.

So, we want to determine when *CAW* is true and *MSW* is false. Here is a sample of rules that can be used to determine this:

CanAnyWay(before, *p*, *q*) is true if the summary pre- and postconditions of *p* do not conflict with (threaten) the summary preconditions of *q* that are not achieved by *must* postconditions of *p*.

MightSomeWay(before, *p*, *q*) is false if any conflict between the summary pre- and postconditions of *p* and the summary preconditions of *q* (that are not achieved by any summary postconditions of *p*) involves only *must* conditions.

CanAnyWay(overlaps, *p*, *q*) is true if the summary preconditions of *p* and *q* do not conflict; the summary inconditions of *p* do not conflict with the summary pre- or inconditions of *q*; and the summary postconditions of *p* do not conflict with the summary inconditions of *q*.

MightSomeWay(overlaps, p_{sum}, q_{sum}) is false if any of the following pairs of condition sets have a conflict: p_{pre} and q_{pre}, p_{in} and $q_{\mathrm{pre}} \cup q_{\mathrm{in}}$, and p_{post} and q_{in}, where
p_{pre} is the set of *must* summary preconditions of *p* that are not achieved by a summary incondition of *q*;
p_{in} is the set of *must*, *always* summary inconditions of *p* that are not achieved by a summary incondition of *q*;
p_{post} is the set of summary postconditions of *p*;
q_{pre} is the set of *must* summary preconditions of *q* that are not achieved by a summary in- or postcondition of *p*; and
q_{in} is the set of *must*, *always* summary inconditions of *q* that are not achieved by a summary in- or postcondition of *p*.

These rules assume that the plans individually would execute successfully. Specifically, the initial conditions ensure that there is a refinement under which all subplans in the decomposition succeed. This is known as the downward refinement property [2]. Without this assumption we would have to verify that the agents' plans were individually consistent given the current situation—this is the problem addressed by HTN planning.

These rules can be used to efficiently find safe ways to coordinate a set of plans. However, to get the most benefit, these rules should be written to take advantage of as much of the summary information as possible so that coordination opportunities are not missed, and searching for impossible methods is avoided. In other words, we would like them to be *complete*—the rules should determine [*CAW*, *MSW*] to be [true, false] whenever that is the case. However, the rules must also be *sound*—the rules should determine [*CAW*, *MSW*] to be [true, false] only if that is the case. Otherwise, the plans could fail due to decision-making errors using incorrect rules. For example, suppose agents A and B both wanted to move to the location between them as shown in Figure 4c and described in Section 2.3. This is not possible[1]; but, as written, the rules for the *overlaps* relation do *not* determine that *MSW* is false for the top-level plans. However, one might conclude that because *must* postconditions conflict (*At*(A,1,0) and ¬*At*(B,1,0) conflict with *At*(B,1,0) and ¬*At*(A,1,0)), the rules should have been written to account for this and to determine that *MSW* is false. Although this seems like a step towards finding a complete algorithm, we find this to be unsound based on the given specification of hierarchical plans. Because postconditions are not solely interpreted as either necessary conditions or effects, there is nothing that says that when B arrives at (1,0), it does not force A out of the cell because B has a postcondition ¬*At*(A,1,0). *MSW* is actually true since there are plans with the same summary information and can have the *overlaps* relation. The real reason these particular plans cannot be *overlapped* is that the preconditions of the primitive plan moving B into (1,0) are violated by the postconditions of A's plan, and this could only be detected by considering the conditions of the primitives. Therefore, designing complete rules can be dangerous, and proving that certain rules are both sound and complete would be a valuable result, but this is difficult in that it requires formalization of the semantics of hierarchical plan execution and summary information. This is beyond the scope of this paper; instead, we adopt a conservative stance of using sound rules that avoid such dangers because such rules are still effective for coordinating hierarchical plans.

2.3 Comparison Against a Centralized Approach

In the approach described in this paper for coordinating the hierarchical plans of multiple agents, the agents independently preprocess their plan libraries by summarizing conditions for non-primitive plans, and then plans among agents can be coordinated by sending summarized information instead of the entire current plan hierarchy. Another approach is to have the agents send their entire plan libraries to a central agent that off-line derives a table of all temporal relations that *can somehow* work for each pair of plans. Then, as long as agents execute plans within their libraries, this table can be used to quickly coordinate them during execution. This centralized approach is being explored in parallel with the decentralized approach about which this paper focuses.

---

[1] We assume the plan libraries do not include subplans for moving out of the "goal" cell; so, in effect, they cannot leave the cell.

Although we describe these approaches as "centralized" and "decentralized", these terms refer to how the mechanism for coordination is built instead of the coordination itself. For instance, the temporal relation table that is built in a centralized manner could certainly be copied to all of the agents so that coordination decisions are distributed. The main difference is that in the centralized approach complete plan library information is centralized in the preprocessing stage for quick coordination of plans during execution while for the decentralized approach preprocessing is done by each agent alone, and search is used to find a global plan either before or during execution.

An implementation of the centralized approach builds the temporal relation table so that the relations for a pair of plans can work for all subplan choice combinations with possible temporal constraints in addition to those imposed by the relation in the table. In this sense, the table reports *CanSomeWay* relations that are complete in telling when plans can have a certain temporal relation for any choice of subplan decompositions in "some way." Thus, all *CanAnyWay* relations determined for a pair of plans by the decentralized method will be a subset of those in the temporal relation table built using the centralized approach.

We can use a table of temporal relations built this way to validate the conclusions derived by our decentralized approach. Figure 4 shows the temporal relations determined at the top level to be *CAW*, ¬*MSW*, or *CSW* as a result of running implementations of the two approaches on simple examples that allow different kinds of interactions between two agents. The problems shown have arrows describing the hierarchical plans for each agent to reach a destination. A dotted arrow signifies a subplan choice while there are no choices along solid arrows. Movements across adjacent cells are primitives. The plan for agent A in the example shown in Figure 4e is the hierarchical plan detailed in Figure 3. In Figure

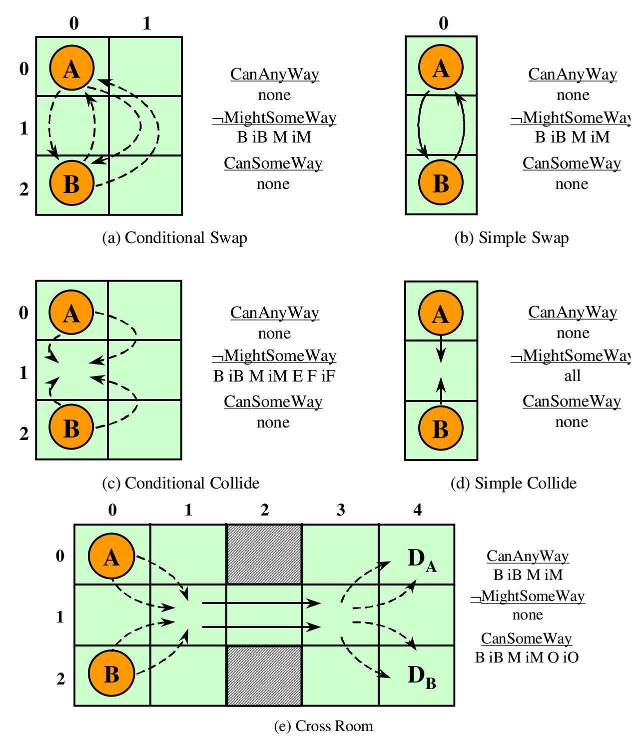

Figure 4: Relations computed using the decentralized and centralized approaches

4a the agents can accomplish their goals by choosing different paths, but the *CSW* computation does not recognize this because there is no coordination strategy for every choice of subplans. The *MSW* rules, however, correctly determine that they can only coordinate if they execute in parallel. As expected, the rules correctly find no *CAW* relations for the examples in Figure 4b, c and d as verified by the centralized method's computation of no *CSW* relations. In Figure 4c the *MSW* rules, as discussed the previous section, cannot determine that plans cannot overlap with only top-level summary information. In Figure 4d the agents can immediately determine that it is impossible to coordinate the plans because *MSW* is false for all relations. Figure 4e shows how the *CAW* rules can directly find ways to coordinate.

3. COORDINATION USING TEMPORAL RELATIONS

With rules that use summary information to determine the ways abstract plans can or might interact, we now describe how plans can be coordinated, or merged into a global plan, by searching for orderings of the plans/subplans that avoid execution failure. This search through the space of global plans is called "top-down" because it tries to first coordinate the plans at the top-level of the hierarchies, then considers their subplans, and iteratively expands subplans until a "feasible" solution is found. We describe a general algorithm for the top-down search and show how an implemented version performs for the simple problems of the

```
Fcn TopDownGlobPlanSearch(topLevelSummInfoList)
  nodes = MakeQueue(MakeNode(InitGlobalPlan(
            topLevelSummInfoList)))
  loop
    if nodes is empty then return failure
    node = PopQueue(nodes)
    if Goal(node.globalPlan) then
      return node.globalPlan
    end if
    if Operator(node)==Expand then
      ExpandPlan(node.planToExpand,
                 node.globalPlan)
    end if
    newnodes = {}
    for each agentPlan in node.globalPlan
    for each subplan sp1 of agentPlan
      if sp1 is an and or an unexpanded
                          or plan then
        newnodes = newnodes ∪
                   Apply(Expand,sp1,node)
      else if sp1 is an or plan then
        for each unblocked subplan sp2 of sp1
          newnodes = newnodes ∪
                     Apply(Block,sp1,sp2,node)
        end for
      end if
      for each agentPlan2 in node.globalPlan
      for each subplan sp2 of agentPlan
        newnodes = newnodes ∪
                 Apply(Constrain,sp1,sp2,node)
      end for
      end for
    end for
    end for
    InsertNodesIntoQueue(newnodes, nodes)
  end loop
end func
```

Figure 5: General top-down global plan search algorithm

previous section.

A state of the coordination search is a 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 *and* plans are the partially expanded plans of the agents, have a depth of one, and contain only the summary information for the immediate subplans. The set of temporal constraints includes orderings dictated by the agents' individual hierarchical plans as well as any synchronization constraints added during the search. Blocked subplans keep track of pruned *or* subplans. The general search algorithm is shown in Figure 5.

The search begins with the global plan consisting of only the summary information for the top-level plans of the agents to be coordinated, an empty set of temporal constraints, and an empty set of blocked plans. The operators of the search are non-primitive plan expansions of the immediate subplans, *or* subplan blocking, and imposing temporal constraints on pairs of subplans. Apply() creates search nodes for new global plans that are obtained by applying an operator to the global plan of the current search state. Expanding requires agents to send summary and ordering information at deeper levels within their hierarchical plans. When expanded, *and* plans are replaced by their subplans' summary information, and the ordering information is updated in the global plan. *Or* subplan summary information is added on expansion, but *or* plans are only replaced by a subplan when all other subplans are blocked.

The Constrain operator should only add temporal constraints that are consistent with those of the global plan. In essence, this operator performs the work of merging non-hierarchical plans since it is used to find a synchronization of the individual agents *and* plans that are one level deep. Thus, plan-merging algorithms such as Georgeff's [10] or Ephrati and Rosenschein's [6] could be employed to find efficient, conflict-free orderings of the subplans based on their summary conditions. These algorithms avoid some complexity by only considering a subset of possible plan interactions (*before*, *after*, and *any*). However, for non-primitive plans, at least, it is important to consider other interactions. For example, it could be determined that high-level plans can or might have the *during* relation but not *overlaps*. Thus, using prior plan-merging techniques without employing rules for determining *MSW* and *CAW* relations could overlook efficient coordination opportunities for these abstract plans.

Specific search techniques (e.g., breadth-first) are implemented by deciding how the nodes are ordered in the queue by InsertNode(). The insertion order may be based on the cost of communicating summary information between agents, the reliability of subplan choices, the expected utility of subplan choices, the expected computation cost, *etc*. The type of search is also decided by what is considered a "feasible" solution in Goal(). For instance, the goal criterion might be that all threats of clobbering conditions are resolved or that a consistent global plan has a cost within some threshold. In the Goal() method, summary conditions are derived for the agents' *and* plans, and *CAW* and *MSW* rules are used to identify relations at the top level that can or cannot hold. Thus, a search method could resolve threats by blocking subplans and could find a valid global plan without looking deeper in the hierarchy.

Another possible algorithm could combine aspects of this approach and those of the centralized approach described in Section 2.3. Instead of searching for synchronizations among all of the expanded plans using a potentially exponential constraint satisfaction algorithm, a temporal relations table could be built for the expanded plans using the centralized approach. If there are *CanSomeWay* relations for the top-level plans, then the rest of the coordination can be done on-the-fly as in the centralized approach.

We implemented a simple depth-first search (DFS) algorithm to validate the decentralized preprocessing approach. To avoid visiting the same search states, Select operators were used to block all but one subplan, and operators were ordered first by the agent of the plan being expanded or chosen, then with the expansion of *and* plans preceding that of *or* plans, and finally by temporal ordering of plans (first executed, first expanded). *CAW* rules were used to identify safe plan relations and restrict the search through the space of plan synchronizations. The first consistent solution found was returned. If communication costs were the overriding concern, this might be an appropriate algorithm since agents with smaller costs could be expanded first. Figure 6 shows how this algorithm worked for the previous examples from Figure 4.

A trace of the execution of the DFS algorithm is given for each example in Figure 6 with plan steps numbered for the plans that were successfully coordinated. Using the *CAW* and *MSW* relations, synchronizations were found for Figure 6a and e without unnecessary plan expansion. In Figure 6c, because ¬*MSW* cannot be determined for some relations at abstract levels (as described for the same example in Figure 4c), the algorithm had to expand all subplans to find that no solution was possible.

Thus, in many cases the top-down approach can avoid synchronizing plans at the primitive level. However, in other cases (like Figure 6c) where interactions at the lowest level are

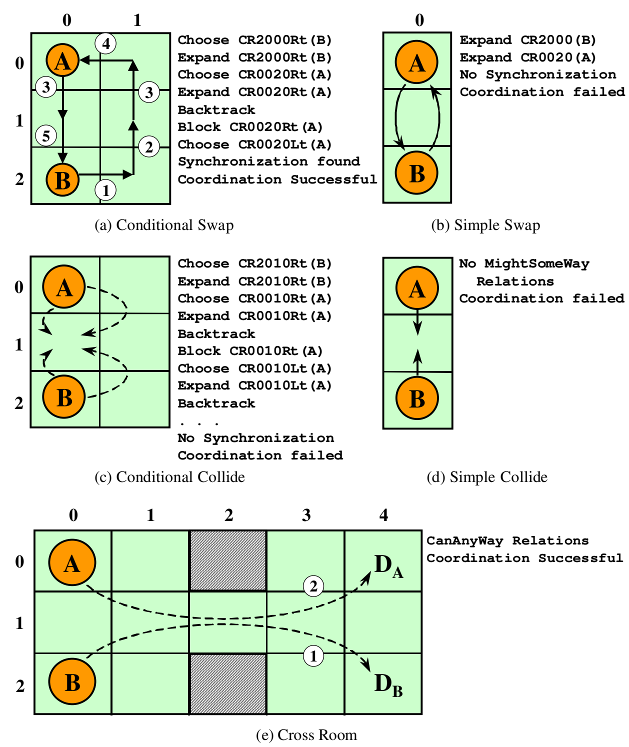

Figure 6: Sample executions of a top-down DFS algorithm

key in determining potential solutions, and full plan information is needed, a top-down algorithm may incur overhead that could be avoided by merging primitives from the start, as described Georgeff [10]. In the cases where the plans can be coordinated at abstract levels, the *or* branches underneath in the individual plans would be preserved, allowing agents to retain decomposition options and robustness in their plans that can be taken advantage of by procedural reasoning systems.

4. DISCUSSION AND EVALUATION

A more obvious approach to coordinating individually formed plans of multiple agents is to globally replan for the conjunction of their goals. Centrally constructing a flat (as opposed to a conditional or hierarchical) distributed plan from scratch could be done with a partial order planner or graphplan, and the tasks could be distributed back to the agents with synchronization actions inserted for parallel execution. This assumes the coordinating agent has all specialized knowledge needed to build a plan to accomplish each of the agents' goals. On the other hand, if the agents have hierarchical libraries of plans, a good solution would be one that left the agents multiple ways of decomposing their plans. Global replanning would result in a single flat plan that provides none of the options and robustness that are part of hierarchical plans. Global replanning could be performed repeatedly to build a library of global plans, but there would be no choice points since the library is not hierarchical, and a single agent's plan hierarchy represents b^n plan choices for *n or* plans with *b* subplans. Ideally, we would like to keep ~b^n plan choices without planning b^n times.

We might expect that plan-merging techniques could provide a jump-start in finding a global plan that satisfies the combined goals of a group of agents because the techniques take advantage of effort already spent on reaching the goals. However, replanning from scratch is complete in that it could find a plan that plan-merging techniques could not, assuming that the planning agent has access to all primitive operators. The advantage of the top-down method is that it provides a heuristic for looking for simpler coordination solutions at higher levels in the hierarchy in order to avoid the intractability of synchronizing the primitive actions of fully decomposed plans.

But, it would be desirable to merge the plans at the primitive level if the primary goal is to minimize the completion time of the agents' plans that closely interact at the primitive level. In some cases, it may be necessary to merge at the primitive level to find any solution. For problems with solutions at different levels in the hierarchy, we would expect that there is a tradeoff between the detail of the coordination and the computation cost.

Figure 7 illustrates this tradeoff for the example in Figure 6e by comparing the total costs of coordinating at different levels under varying computation/execution time cost ratios. Suppose we used a plan-merging algorithm, like Georgeff's [10], within a top-down breadth first search algorithm to synchronize the plans at three different levels corresponding to the hierarchical plan given in Figure 3: the top-level where the agents A and B have only the summary information for the top-level, a mid-level where A and B have three-step plans, and the primitive level where they each have six steps.

At the highest level, we could only discover before/after solutions resulting in a completion time of 12 primitive steps, but the computation time would be minimal. At the primitive level, one agent needs to wait one step at best to avoid a collision, so the agents can complete their plans in 7 steps. At the middle level, an agent must wait two steps for a best completion time of 8. So, there is a great improvement in the plan by coordinating at the mid-level, while there is only a minor improvement (one step) by coordinating the primitives. If we assume that the expected time of the algorithm is proportional to the space of interleavings of the plans, computation time grows exponentially with the total steps of the plans of agents A and B. Figure 7 shows how the total cost of coordination (y-axis) varies for each level of coordination for different computation/completion time cost ratios (x-axis). For a small ratio, where computation is cheap, the lowest cost strategy is to coordinate the primitive plans. But if computation is more costly, coordination at the mid- or top-level would be best, depending on the ratio. So, depending on how costly computation time is with respect to the average completion time, different levels of coordination may return lower cost solutions.

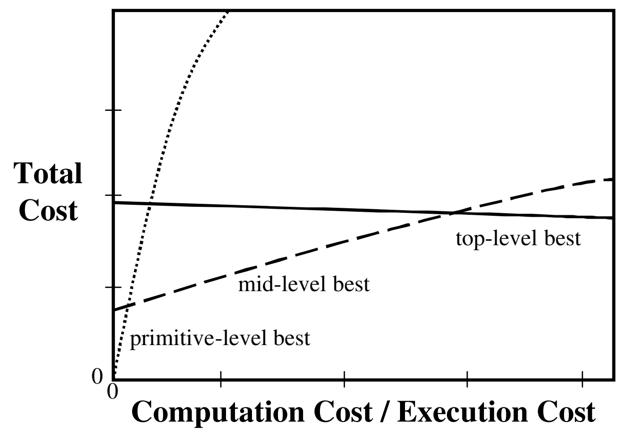

Figure 7: Tradeoffs of coordinating at different levels

5. CONCLUSION

We have described a method for extracting summary information for hierarchical plans and offered methods to coordinate the plans of multiple agents at different abstract levels based on this information. Our simulations show how algorithms can be developed to avoid unnecessary computation by abstracting details of lower level subplans. We have motivated future work in formalizing the rules for deriving summary conditions and determining safe plan interactions because incomplete rules cannot identify all opportunities for coordination at higher levels. However, plan interactions can be complex, and many complete specifications can lead to unsound rules that can cause agents to make costly irreversible decisions. Providing sound and complete mechanisms are left to future work. Other future considerations include relaxing assumptions to allow for more expressive plan representations and finding other ways to summarize the information of plan refinements.

6. ACKNOWLEDGEMENTS

This work benefited from discussions with Pradeep Pappachan and from code he has written. NSF (IRI-9158473) and DARPA (F30602-98-2-0142) supported this research.

7. REFERENCES

[1] Allen, J.F., Kautz, H.A., Pelavin, R.P., and Tenenberg, J.D. *Reasoning about Plans*, Morgan Kaufmann, San Mateo, CA, 1991.

[2] Bacchus, F., and Yang, Q. The downward refinement property. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence.* pp.286-292, 1991.

[3] Corkill, D. Hierarchical planning in a distributed environment. *Proceedings of the International Joint Conference on Artificial Intelligence.* pp.168-175, 1979.

[4] Decker, K., and Lesser, V. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4), 1994.

[5] Durfee, E.H., and Montgomery, T.A. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Distributed AI, SMC-21(6):1363-1378, November 1991.

[6] Ephrati, E., and Rosenschein, J. Divide and conquer in multi-agent planning, *Proceedings of the Twelfth National Conference on AI*, pp. 375-380, 1994.

[7] Erol, K., Hendler, J., and Nau, D. Semantics for hierarchical task network planning, Technical report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland, March 1994.

[8] Fikes, R.E., and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, 2(3-4):189-208, 1971.

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

[10] Georgeff, M.P. Communication and interaction in multiagent planning, *Proceedings of the Third National Conference on AI*, pp. 125-129, 1983.

[11] Georgeff, M.P. and Lansky, A. Procedural knowledge, *Proceedings of IEEE* 74(10):1383-1398.

[12] Lansky, A. Localized search for controlling automated reasoning, *Proceedings of DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*. pp.115-125, 1990.

[13] Nilsson, N.J. *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA, pp. 282-287, 1980.

[14] Tsuneto, R., Hendler, J., and Nau, D. Analyzing external conditions to improve the efficiency of HTN planning, *Proceedings of the 15th National Conference on AI*, pp. 913-920, 1998.