Identifying and Resolving Conflicts among Agents with Hierarchical Plans

Bradley J. Clement and Edmund H. Durfee

Artificial Intelligence Laboratory
University of Michigan
Ann Arbor, MI 48109-2110
bradc@umich.edu

Abstract

Agents can use negotiation techniques to resolve conflicts over limited resources so that they can achieve their goals. The individual plans of agents provide the necessary information for discovering and resolving such conflicts. Conflicts 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 identify conflicts and find ways to coordinate hierarchical plans from the top down to primitive actions. Agents can then negotiate over the choice of different coordination methods using any of several techniques.

Introduction

The development of negotiation techniques enables agents to accomplish their goals in environments with limited resources. Agents' plans and goals provide the information necessary to determine when conflicts may arise and, thus, when negotiation is needed. However, determining conflicts among planning agents requires identifying conflicting conditions or states required by their plans. At the same time, it is important to preserve the agents' options of accomplishing their subgoals when their plans are conditional.

Conflicts during plan execution can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents [11] and by merging/coordinating the individual plans of agents by introducing synchronization actions [9]. There may be several ways of coordinating the agents' plans, and deciding on one way is the purpose of negotiation.

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 [10], 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 other's abstract plans and how those plans could possibly be refined, then such potential conflicts 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 [11], and Georgeff [9] and Ephrati and Rosenschein [6] described methods for merging plans at the primitive level, the work reported here combines these strategies such that conflicts can be resolved flexibly at the levels in-between.

Because coordinating at abstract levels involves making

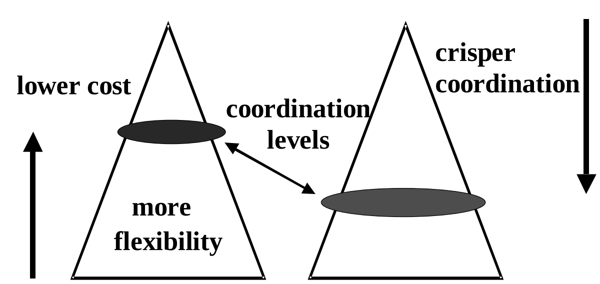


Figure 1: Hierarchical plan coordination at multiple levels

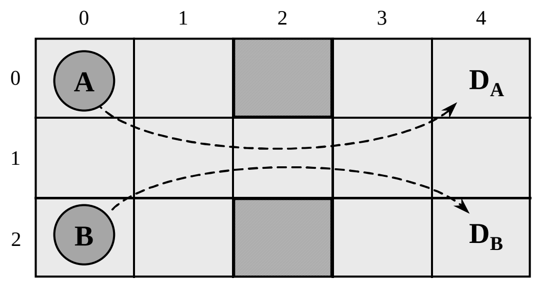


Figure 2: Two agents moving within a grid world.

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, I describe procedures for deriving summary information for non-primitive plans that capture the *external preconditions* [13] 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 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 could be ordered. The algorithm reports candidate synchronizations of subplans as it discovers them. At any point, the agents can employ one of various techniques to negotiate what properties the merged plan should have and then commit to one of the valid choices.

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 4) 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 one or more ways to synchronize the subplans for more fine-grained solutions. Then, B can negotiate with A over which solution should be chosen.

Overview

This paper summarizes work described previously in [2] and [3]. I describe how to derive summary conditions of hierarchical plans. This information is used to describe *CanAnyWay* and *MightSomeWay* rules that are used to efficiently identify conflicts and reveal how plans can or might be temporally related without looking at how they could be refined. Then, I show 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. I also give examples of how an implemented instance of the general algorithm coordinates plans. This approach is evaluated by comparing it to replanning for the combined goals of agents and by showing the tradeoffs of coordinating at different levels of abstraction under different cost scenarios.

Using Summary Information

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 [9], 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. A thorough formalization of these hierarchical plans and their concurrent execution is given by [3].

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.

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 the introduction. 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. Rules for determining how conditions are propagated from the primitives up a hierarchy are formalized in [3]. This work also proves properties of summary conditions. For example, summary preconditions and postconditions are exactly the plan's sets of *external preconditions* and *postconditions*, respectively.

External precondition are 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.* [13]. Similarly, *external postconditions* are the effects of subplans that are not undone by other subplans, 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.

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 illustrated in the introduction, 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 $\neg v$. This means p has *may* summary postconditions v and $\neg v$. 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 whether the *overlaps* relation can or might hold for two plans, p and q.

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 $q_{\text{pre}} \cup q_{\text{in}}$, 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;

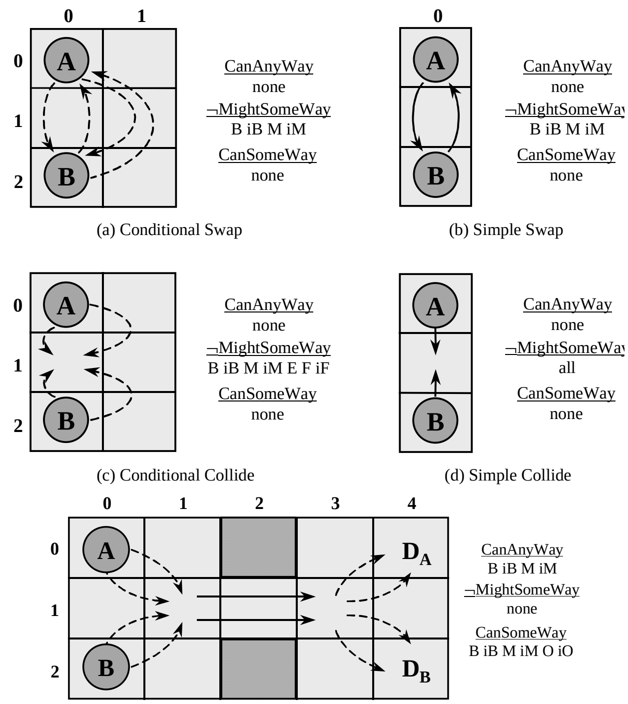
q_{pre} is the set of *must* summary preconditions of q that are not achieved by a summary in- or postcondition of p; and
q_{in} is the set of *must*, *always* summary inconditions of q that are not achieved by a summary in- or postcondition of p.

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 solution property. 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. A proof of the soundness and completeness of such rules in determining the temporal relations that can or might hold between plans based on their summary information are described in [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 resolve conflicts during execution. This centralized approach is being explored in parallel with the decentralized approach about which this paper focuses.

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.



(a) Conditional Swap (b) Simple Swap (c) Conditional Collide (d) Simple Collide (e) Cross Room

Figure 3: Relations computed using the decentralized and centralized approaches.

We can use a table of temporal relations built this way to validate the conclusions derived by our decentralized approach. Figure 3 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 3e is the hierarchical plan detailed in Figure 4. In Figure 3a 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 3b, c and d as verified by the centralized method's computation of no *CSW* relations. In Figure 3c the *MSW* rules cannot determine that the plans cannot overlap with only their top-level summary information. In Figure 3d the agents can immediately determine that it is impossible to coordinate the plans because *MSW* is false for all relations. Figure 3e shows how the *CAW* rules can directly find ways to coordinate.

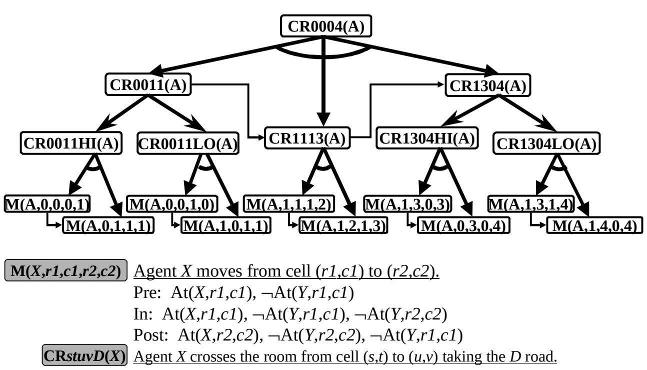


Figure 4: Agent A's hierarchical plan

Coordination Using Temporal Relations

These *CAW* and *MSW* rules identify temporal relations that potentially or definitely cause conflicts and determine whether they can or might be resolved. 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 implemented versions perform for the simple problems of the 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 detailed in [2].

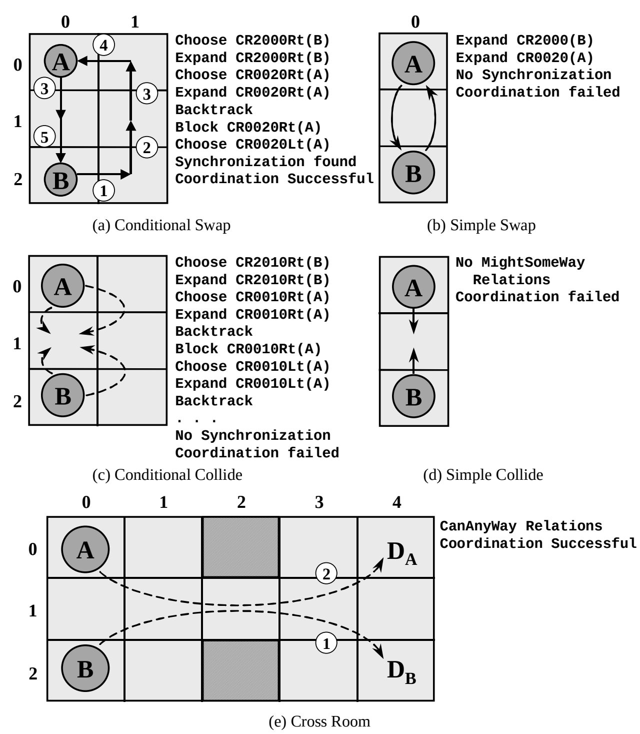
The search begins with the global plan consisting of only the summary information for the top-level plans of the agents to be coordinated, an empty set of temporal constraints, and an empty set of blocked plans. The operators of the search are non-primitive plan expansions of the immediate subplans, *or* subplan blocking, and imposing temporal constraints on pairs of subplans. New global plans are obtained by applying an operator to the global plan of the current search state to create new search nodes to insert into the queue. 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.

Specific search techniques (e.g., breadth-first) are implemented by deciding how the nodes are ordered in the queue. 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.

We implemented a simple depth-first search (DFS) algorithm to validate the decentralized preprocessing approach. To avoid visiting the same search states, s*elect* 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 5 shows how this algorithm worked for the previous examples from Figure 3.

A trace of the execution of the DFS algorithm is given for each example in Figure 5 with plan steps numbered for the plans that were successfully coordinated. Using the *CAW* and *MSW* relations, synchronizations were found for Figure 5a and e without unnecessary plan expansion. In Figure 5c, because ¬*MSW* cannot be determined for some relations at abstract levels (as described for the same example in Figure 3c), 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 5c) where interactions at the lowest level are 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 [9]. In the cases where the plans can be coordinated at abstract levels, the *or* branches underneath in the individual plans would be



(a) Conditional Swap (b) Simple Swap (c) Conditional Collide (d) Simple Collide (e) Cross Room

Figure 5: Sample executions of a top-down DFS algorithm

preserved, allowing agents to retain decomposition options and robustness in their plans that can be taken advantage of by procedural reasoning systems.

Negotiation

The top-down search algorithm may return several global plans that each describe how subplan choices at different levels can be synchronized to accomplish all of the agents' goals. Agents may have preferences over subplan choices and time constraints on reaching certain goals, so an agent may prefer some global plans to others, and it is unlikely that they will all prefer the same global plan. The agents can employ any of various techniques to negotiate over which features the global plan should have. For example, the agents may be willing to pay different amounts of money for different properties of the global plan, and a market mechanism could be used to select a Pareto optimal global plan. Among many other techniques, voting or distributed constraint-based search could also be employed.

Conclusion

I have described a method for identifying and resolving conflicts among multiple planning agents based on summary information derived for hierarchical plans. This information is used to merge the plans of multiple agents at different abstract levels to preserve robust execution and to avoid unnecessary computation by abstracting details of lower level subplans. The concurrent execution of hierarchical plans and the properties of coordination mechanisms have been formalized in order to avoid costly, irreversible search and coordination decisions. The general algorithm and specific implementations are shown to provide a set of solutions from which agents can choose using negotiation techniques. Future considerations include relaxing assumptions to allow for more expressive plan representations and finding other ways to summarize the information of plan refinements.

Acknowledgements

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

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] Clement, B., and Durfee, E. Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents. To appear in the *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[3] Clement, B., and Durfee, E. Theory for Coordinating Concurrent Hierarchical Planning Agents using Summary Information. To appear in the *Proceedings of the 16th National Conference on AI*, 1999.

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

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

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

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

[13] 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.