An Efficient Algorithm for Multiagent Plan Coordination

Jeffrey S. Cox and Edmund H. Durfee
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109
{jeffcox, durfee}@umich.edu

ABSTRACT

The *multiagent plan coordination problem* arises whenever multiple agents plan to achieve their individual goals independently, but might mutually benefit by coordinating their plans to avoid working at cross purposes or duplicating effort. Although variations of this problem have been studied in the literature, there is as yet no agreement over a general characterization of the problem. In this paper, we describe a general framework that extends the partial-order, causal-link plan representation to the multiagent case, and that treats coordination as a form of iterative repair of plan flaws that cross agents. We show, analytically and empirically, that this algorithmic formulation can scale to the multiagent case better than can a straightforward application of the most advanced single-agent plan coordination technique, highlighting fundamental differences between single-agent and multiagent planning.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

Coordination of multiple agents, Multiagent planning and merging

1. INTRODUCTION

The phrase *multiagent planning* has acquired a variety of meanings over the years. In part, this may be due to the ambiguity of exactly what someone considers to be "multiagent" about the planning. In some work [13, 21], it is the planning process that is multiagent; for example, multiple agents, each with specialized expertise in certain aspects of planning, might collaborate to formulate a complex plan that none of them could have generated alone. In other work [3, 11], it is the product of planning—the plan itself—that is multiagent, in the sense that it specifies the activities of multiple actors in the environment such that they collectively achieve their individual and/or common goals. And sometimes, it is both [8, 10, 4, 18, 7],where multiple agents interact to arrive at plans that will be carried out by multiple (and often, it is implicitly assumed, the same) agents.

This paper concentrates on a restricted but important class of problems of this third type, which we call *Multiagent Plan Coordination Problems* (MPCPs), in which multiple agents each plan their own individual activities but might mutually benefit by coordinating their plans to avoid interfering with each other unnecessarily duplicating effort. Multiagent plan coordination differs from "team planning," in which agents must work together more tightly as a team in order to achieve their joint goals. Instead, multiagent plan coordination is suited to agents that are *loosely-coupled* (nearly independent), where each agent can achieve its own goals by itself, but the presence of other agents who are also asynchronously operating in the same environment leads to potential conflicts and cooperative opportunities. This concept of coupling is similar to that described by Pecora et.al. in [15], where the problem of finding plans with minimal makespans is considered. In both their and our definition, the degree of coupling measures the degree of interaction between different plans (or "threads" in [15]) and thus affects the inherent difficulty of the planning problem.

In general, the multiagent plan coordination problem is known to be NP-Hard [22]. Despite this complexity, many researchers [10, 22] have explored the problem in detail. In particular, Yang [22] has developed a rigorous computational theory of single-agent plan coordination, and implemented an efficient and optimal algorithm that, under assumed characteristics, is polynomial with respect to the size of the plan coordination problem.

However, our investigations have indicated that there are fundamental differences between the *single-agent* plan coordination problem and the problem of coordinating the plans of *multiple agents*, that make Yang's methods less appropriate to the multiagent plan coordination problem. In this paper, we describe key differences between the single-agent and multiagent plan coordination problem, analyze why these differences are important, and show how previous work based on the single-agent problem is not well suited to address the multiagent problem. We then describe new methods for plan coordination that account for these differences and thus, under certain assumptions, *are* well-suited for the multiagent plan coordination problem. We conclude with empirical and analytical comparisons of our algorithms with Yang's work demonstrating the superiority of our techniques on multiagent coordination problems, and close with a discussion of our future research directions.

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

2. RELATED WORK

In general, past approaches to plan coordination in the single-agent case have relied on greedy algorithms, and thus were neither optimal nor complete, nor was their complexity well characterized [22]. However, Yang [22] was one of the first to give a formal characterization of both the coordination problem and its solution, although he considers a restricted formulation of the plan coordination problem that we consider in this paper. In Section 4, we conduct a detailed analytical and empirical comparison of Yang's method and our own, illustrating the advantages of our approach for loosely-coupled multiagent systems.

Ephrati [10] has applied the divide-and-conquer approach to multiagent planning that farms out subgoals of a single goal to different agents to solve and then integrate their sub-plans into a joint, multi-agent plan. His algorithm performs an A* search over possible interleavings of the plan steps in the sub-plans to arrive at a near-globally optimal solution. Although Ephrati's approach seems well suited for the multiagent plan coordination problem, it suffers from drawbacks of incompleteness, i.e., it is not guaranteed to return a solution to a given coordination problem, and thus does not make for a good solution should agents want optimally-coordinated plans.

More recently, Tonino et. al. [18] have described algorithms for coordinating the plans of multiple agents based on an underutilization of free resources. Unfortunately, in their approach, once a free resource (i.e., an effect) of a plan step has been used to replace the effect of another plan step, it is no longer available to replace the effects of other plan steps. Thus, their system will not work for the problem of plan coordination we consider, where a single step can produce a single effect used by many other steps of other agents in the multiagent plan.

Our own past work [7] described an algorithm for performing plan coordination between the different plans of multiple agents, where the algorithm exploited a *hierarchical* plan representation to reduce the complexity of the plan coordination problem. That work was itself based on work by Clement [4] that successfully identified and resolved conflicts between independent hierarchical plans. Others who have studied the multiagent plan coordination problem include [19, 17, 9, 12].

Despite the large body of work on multiagent plan coordination, there has been little attention paid to the problem of developing *computationally-efficient* algorithms to solve the multiagent plan coordination problem that still produce optimally-coordinated plans. It is this gap that our research is intended to fill.

3. MULTIAGENT PLAN COORDINATION PROBLEMS

The *Multiagent Plan Coordination Problem* (MPCP) is the problem of identifying and resolving *interactions* between the plans of different agents. To define and characterize the multiagent plan coordination problem, we first characterize what a plan is from the single-agent perspective, and then expand the concept to the multiagent case. The definitions given in this section are variations of well-established plan formalisms [3, 20, 1], given here for completeness.

3.1 Planning Concepts and Problems

Our definition of a multiagent plan extends the partial-order, causal-link (POCL) definition of a plan that has been well-established in the planning community [20], restricted to ground (or variable-free) operators described by propositional conditions.

DEFINITION 3.1. *A **POCL plan** is a tuple $P = \langle O, S, \prec_T, \prec_C \rangle$ where O is a set of plan operators, S is a set of plan steps (instantiated operators), $\prec_T$ and $\prec_C$ are (respectively) the temporal and causal partial orders on S, where $e \in \prec_T$ is a tuple $\langle s_i, s_j \rangle$ with $s_i, s_j \in S$, and where $e \in \prec_C$ is a tuple $\langle s_i, s_j, c \rangle$ with $s_i, s_j \in S$ and $c \in \Sigma$. A POCL plan has an* init *step, init $\in S$, and one or more* goal *steps, goal$_i \in S$ where the preconditions of the* goal *steps represent the conjunctive goal that the plan achieves, and the postconditions of the* init *step represent features of the initial state.*

Elements of S are *instances* of elements in O, and there may be multiple unique instances of a single operator from O in S. We will use $op(s)$ to refer to the operator that step s was instantiated from. Elements of $\prec_T$ are commonly called **ordering constraints** on the steps in the plan, and elements of $\prec_C$ are commonly called **causal links**, the latter representing causal relations between steps, where causal link $\langle s_i, s_j, c \rangle$ represents the fact that step s_i achieves condition c for step s_j. Temporal orderings are transitive and required to be irreflexive (so there are no cycles in the plan).

The *planning problem* is the problem of transforming an *inconsistent* plan into a *consistent* plan. A plan is inconsistent when it has plan *flaws*. Most research in planning has focused on the problem of *plan generation* (creating a plan from a starting state, a goal state, and a set of operators). For the plan generation problem, a plan flaw is either a *causal link threat flaw* or an *open condition flaw*.

DEFINITION 3.2. *A **causal-link threat flaw** in a POCL plan exists when there is some step s_k and some causal link $e \in \prec_C$ of form $\langle s_i, s_j, c \rangle$, s.t. not$(c) \in post(s_k)$, $\langle s_k, s_i \rangle \notin \prec_T$ and $\langle s_j, s_k \rangle \notin \prec_T$.*

Intuitively, the presence of a causal-link threat flaw in a plan indicates that there exist executions (linearizations) of the plan where the step s_k undoes (or "clobbers") a condition after it is asserted by s_i but before it can be used by step s_j. Given a threat between a step s_k and a causal link $\langle s_i, s_j, c \rangle$, standard plan-space methodologies add either $\langle s_k, s_i \rangle$ or $\langle s_j, s_k \rangle$ to $\prec_T$.

In addition to causal-link threat flaws, other causes of plan inconsistency include *open precondition flaws*.

DEFINITION 3.3. *An **open precondition flaw** exists when there is some step s_j with precondition c but there is no causal link $\langle s_i, s_j, c \rangle \in \prec_C$.*

An open precondition c of a step s_j can be satisfied by adding a causal link $\langle s_i, s_j, c \rangle$ where $c \in post(s_i)$ and either $s_i \in S$ and $\langle s_j, s_i \rangle \notin \prec_t$ (s_i is already in the plan and not ordered after s_j), or $op(s_i) \in O$ (s_i can be instantiated from the operator set of the plan).

In general, for any given flawed single-agent POCL plan, there may be many possible *consistent* plans one could create by repairing the various flaws. However, not all *consistent* plans will be *optimal*. Although there are many ways of measuring plan optimality, including the number of plan steps, as well as their makespan (as in Pecora's work [15]), we will adopt the step-minimization metric for this paper.

3.2 Plan Coordination Concepts and Problems

Just as individual agents want to create valid plans to achieve their goals, multiple agents executing their plans in a shared environment will want to coordinate their individual plans to ensure that all agents can achieve their goals as well. We define a *multiagent plan coordination problem* (MPCP) as the problem, given a set of agents **A** and the set of their associated POCL plans **P**, of determining if there is some *subset* of plan steps from the plans of the agents that can form a consistent multiagent parallel plan that

results in the establishment of all agents' goals, given the initial state of the agents (recall that the POCL representation represents the agent's initial and goal states as steps in the plan).

It is worth noting that any solution to the MPCP will only contain plan steps from the plans of the individual agents. Thus, our definition of the multiagent plan coordination problem essentially makes the problem a *bounded-length* planning problem, which in its most general form is the problem of trying to determine if a consistent plan exists that can have at most k steps (where k is the bound). Just as optimal single-agent planning can be viewed as the problem of trying to solve a series of increasing bounded-length problems until a valid plan is found [2, 14], we could view multiagent planning the same way. For example, agents could iteratively submit larger (and thus less-optimal) plans for plan coordination, until a solution is found. While formulating such approaches to the more general multiagent planning problem is outside the scope of this paper, we note that the problem of plan coordination is likely to be a fundamental component of the more general multiagent planning problem.

Both the single-agent planning problem and the Multiagent Plan Coordination Problem can be seen as the problem of transforming an *inconsistent* plan into a *consistent* plan. That is, we can think of the union of the individual agent plans as implicitly representing a single, multiagent plan P with its own set of flaws that need to be resolved. However, to ensure that we address all possible plan flaws, including those unique to the multiagent context, we must first extend our planning model to handle the *concurrency* that can arise in multiagent planning and execution. That is, unlike the single-agent planning case, multiple agents executing plans in the same domain may possibly execute steps in parallel, and thus we will want to extend our plan semantics to reason about action concurrency. First, we extend our operator model to include *inconditions* (or "during" conditions) [4], which describe the state of the world that holds during the execution of a plan step (we also relax our implicit assumption that actions are instantaneous). Operators (and thus steps) are now described by their preconditions, inconditions ($in(s_i)$), and postconditions. With this extended action model, we are ready to extend our model of a multiagent POCL plan:

DEFINITION 3.4. *A **multiagent parallel POCL plan** is a tuple $P = \langle A, O, S, \prec_T, \prec_C, \#, =, X \rangle$ where $\langle O, S, \prec_T, \prec_C \rangle$ is the embedded POCL plan, A is the set of agents, X is a set of tuples of form $\langle s, a \rangle$, representing that the agent $a \in A$ is* assigned *to executing step s[1], $=$ is the symmetric* concurrency *relation over the steps in S, and # is a symmetric* non-concurrency relation *over the steps in S.*

The relation $\langle s_i, s_j \rangle \in \#$ is the same as the statement $(\langle s_j, s_i \rangle \in \prec_T) \vee (\langle s_i, s_j \rangle \in \prec_T)$. The relation $\langle s_i, s_j \rangle \in =$ means that s_i and s_j are required to be executed simultaneously. For example, if a plan has multiple goal steps and is intended to reach a state where all goals are satisfied simultaneously, then all pairs of goals steps would be elements of $=$.

Given this definition, it is clear that a POCL plan P is a specialization of a multiagent parallel plan P' in which either all pairs of steps in P' are in $\#'$ (if it is assumed an agent can do only one action at a time) or none of them are (if all unordered actions are assumed to be concurrently executable). Likewise, a POCL plan implicitly requires that $=$ be empty, unless a single agent can execute multiple steps concurrently.

Given our multiagent plan definition, we can now introduce a new kind of plan flaw:

[1]Although this element raises the possibility of step reassignment, in this paper we do not consider reassigning actions among agents.

DEFINITION 3.5. *A **parallel step threat flaw** exists in a multiagent parallel plan when there are steps belonging to different agents[2] s_j and s_i where $post(s_i)$ or $in(s_i)$ is inconsistent with $post(s_j)$ or $in(s_j)$, $\langle s_j, s_i \rangle \notin \prec_T$, $\langle s_i, s_j \rangle \notin \prec_T$ and $\langle s_i, s_j \rangle \notin \#$.*

This definition is an extension of the *post-exclusion principle* [1], stating that actions cannot take place simultaneously when their postconditions are not consistent. Parallel step threat flaws can always be resolved no matter what other flaw resolution choices are made, and thus these flaws do not have to be considered using backtracking search for valid flaw resolutions. Thus, parallel step conflicts between two steps s_i and s_j can be resolved by adding $\langle s_i, s_j \rangle$ to #, leaving the enforcement of this constraint either to a simple postprocessing step or to the plan execution platform.

Although our definition of a multiagent parallel plan extends the standard partial-order plan definition, we note that in fact our definition is itself somewhat of a restriction of Boutilier's concurrent interacting actions STRIPS planning model [3]. Specifically, while we worry about some concurrent step interaction (via the post-exclusion principle), in his work, Boutilier provides semantics for reasoning about the effects of concurrent actions whose effects may be different than the simple combination of the effects of the individual steps (such as two agents picking up either end of a table). Boutilier's extension is straightforward, and in principle, it would be possible to extend our model to support such extended interaction semantics. Indeed, our representation is perhaps better adapted to the multiagent context, as the step assignment element of our plan representation is treated as a first-class variable, whereas in Boutilier and Brafman's work, the assignment of an agent is embedded in the instantiation of a step. However, extending our model to handle the kinds of interactions characterized in [3] would needlessly complicate the representation described in this paper, and detract from our central claim of the efficiency of plan-space search for loosely-coupled multiagent plan coordination.

The MPCP has other flaws that are not so easy to resolve. Since the individual plans of the agents are complete, there are no open condition flaws. However, causal link threat flaws are still present, as well as possible inter-agent *plan step merge flaws*.

DEFINITION 3.6. *A **plan step merge flaw** exists in a (potentially inconsistent) multiagent plan P when there exists in P two steps, s_i and s_k and, for each causal link $e \in \prec_C$ of form $\langle s_i, s_j, c \rangle$, it is also the case that $c \in post(s_k)$ and $\langle s_j, s_k \rangle \notin \prec_T$.*

That is, there is some step whose postconditions subsume all of the *necessary postconditions* (those postconditions associated with the outgoing causal links) of another step. This definition is based on the general form of Yang's plan step merging criteria from [22]. An important difference between plan step merge flaws and threat flaws is that threat flaws *must* be resolved for a multiagent plan to be consistent, whereas a step merge flaw can be ignored if agents are willing to tolerate the redundancy in the plan. The process by which steps s_i and s_k are merged (as defined in [22]) is as follows:

1. For each causal link $l \in \prec_C$ of form $\langle s_i, s_j, c \rangle$ add new link of form $\langle s_k, s_j, c \rangle$ to $\prec_C$.
2. For each causal link $l \in \prec_C$ of form $\langle s_i, s_j, c \rangle$ remove link from $\prec_C$.
3. For each causal link $l \in \prec_C$ of form $\langle s_p, s_i, c \rangle$ remove link from $\prec_C$.

[2]We assume that a single agent cannot execute actions in parallel. Obviously, if this restriction is relaxed we can also consider parallel step threats between steps of the same agent.

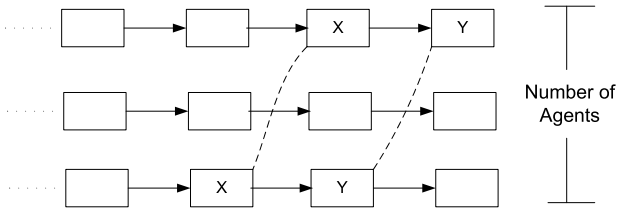

Figure 1: Visualizing Our Plan Coordination Algorithm

4. Remove s_i from S.

3.3 A Multiagent Plan Coordination Algorithm

To solve the MPCP, we have developed a general plan-space search algorithm that searches through the space of possible flaw resolutions between a set of agent plans to produce a coordinated solution. Our algorithm optimizes with respect to the total number of steps shared by all agents, which is a global optimality measure.

Algorithm 1: A Multiagent Plan Coordination Algorithm

```
Input  : an inconsistent multiagent plan
Output : an optimal and consistent multiagent plan
Initialize Solution to null;
Add input plan to search queue;
while queue not empty do
    Select and remove plan P from search queue;
    if P not bounded by Solution then
        if (P has no threat flaws) and (P is acyclic)
        and(cost(P) < cost(Solution)) then
            Solution = P;
        end
        Select and repair a flaw in P;
        Enqueue all repaired plans in search queue;
    end
end
return Solution;
```

The search algorithm (shown in Algorithm 1) begins by initializing the search queue with whatever current (flawed) multiagent plan it is to operate on, and by initializing the currently best solution, *Solution*, to *null*. Then, while the queue is not empty, it selects and removes a plan-state from the queue (the order of which is determined by the search's *heuristic function*). If the plan-state passes the *bounding test*, the algorithm then determines if the plan is a consistent plan that is better than the best consistent plan seen so far. If so, it becomes *Solution*. New plans are generated by choosing a plan flaw and generating new plans by repairing it (as even a consistent plan can still have optimality flaws in it). All possible plan-states generated by the repair are then added to the search queue. In this way, their algorithm converges to a globally minimal solution.

Figure 1 illustrates our plan-space search algorithm graphically. Intuitively, our algorithm first identifies and tries to repair flaws within the plan (in the case of the figure, the pairs of steps with the same letter can merge) and then tests the resultant plan for cycles in the partial order of steps, using the Floyd-Warshall algorithm [5]. This cycle check is carried out by performing a depth-first search on the partial order of steps, to determine if steps can be revisited by the search.

3.4 Using A Branch-and-Bound Algorithm

In generative POCL planning, the presence of an admissible search heuristic guarantees that the first *consistent* plan will also be the optimal one (given that the optimality measure is the number of steps in the plan), as the planner will start with small plans and build up. In contrast, given that our algorithm's starting point of a set of individually consistent POCL plans, there is no guarantee that the first *consistent* coordinated plan that our algorithm finds will be an *optimal* one.[3] This is why our algorithm, when searching for an optimal solution, cannot simply return the first consistent plan that is found, but must keep track of the *best* solution found until it can ensure that there is no other solution that is better.

Besides keeping track of the best solution seen so far, the algorithm can use *Solution* to "prune" the search space by preventing refinement of plan-states that can be determined to never lead to plans better than those already discovered. This pruning is accomplished using a *branch-and-bound algorithm*, where *Solution* is the effective upper bound. If a plan-state's lower bound (a best-case estimate of how good the plan-state is) is higher than the current upper bound, the state can be discarded. By using a good bounding mechanism, we can often dramatically reduce the size of the search space. Of course, if a consistent solution is all that is required, then the algorithm can terminate as soon as any consistent solution is found.

In our algorithm, we prune any plan state whose cost (number of plan steps) is higher than the current best solution, even if we assume *all* plan step merges could be performed in the state (ignoring unresolved threats and cycles in the network of steps). This is done by counting all steps that have outgoing causal links where some subset of the outgoing links could be replaced by other steps in the plan, and the remaining outgoing links point to steps that could also be removed by plan step merging. This bound is guaranteed never to underestimate the potential value of a state, and as we will see later, this bound gives the algorithm very good expected-case performance.

3.5 Computational Complexity of Our Coordination Algorithm

We can see the complexity of our algorithm by considering the number of search states that it can generate. For n plans, each of length d, the largest number of flaws f will be quadratic in d and n in the worst case. For each flaw, the algorithm will branch on all possible repaired plans b, resulting in an overall worst-case complexity of b^f, where f is worst-case $(nd)^2$. Thus, in the worst case, our algorithm will have to generate an exponential number of states in the number of flaws in the problem before terminating. Past work [22] has shown that the MPCP is NP-Hard,[4] and thus a worst-case exponential result is unavoidable.

4. EVALUATING OUR MULTIAGENT PLAN COORDINATION ALGORITHM

Given the worst-case exponential complexity of our algorithm, natural questions to ask are what classes of problems does it perform tractably on, and how does it compare to existing state-of-the-art plan coordination mechanisms? In this section, we examine how our algorithm performs on classes of coordination problems

[3] In fact, the initial multiagent plan formed as the union of the individual agent plans might already be consistent, but still have redundant steps.

[4] To be precise, Yang established that a restricted formulation of the MPCP was NP-Hard, which trivially proves the MPCP is also NP-Hard.

in which the agents are *loosely-coupled*. That is, we assume that agents who plan independently and then coordinate their plans will produce plans that are mostly independent of each other (recall that this assumption was one of the reasons motivating this plan coordination work). To provide some measure of performance comparison, we also examine how our algorithm compares to prior work on the plan coordination problem, particularly that of Yang [22].

Since Yang's algorithm considers a restricted form of the MPCP, for the purposes of making an empirical and analytical comparison between his algorithm and our flaw-repair-based algorithm, we adapt our algorithm to the particulars of the restricted problem he considers. Here we review Yang's dynamic programming algorithm, and then show analytically and empirically the advantage our algorithm has over his algorithm, particularly on loosely-coupled multiagent plan coordination problems.

4.1 Yang's Plan Merging Work

Yang [22] considers a general form of the single-agent plan coordination problem (which he calls Plan Merging) that is easily extensible to the multiagent case we consider, and then he imposes limitations on the problem's form in order to use a particular dynamic programming method to coordinate the plans. Yang's particular restrictions are as follows:

I *Operator-Type Plan Step Merging* - Two steps s_i and s_k can merge only when they have the same **operator type**, and when the two steps are unordered in the plan. Yang assumes that, given a plan coordination problem, there exists a step-typing function that maps any given plan step to a type. [5]

II *Conflict-Free Assumption* - Individual subplans are assumed to be free of steps that conflict with other steps or causal links between other steps.

Note that unlike our earlier definition of a plan step merge flaw, this modified definition is inherently *symmetric* in nature, and thus either of the pair of steps can be dropped.[6] One other difference between the problem Yang considers and the one we do is that when two steps of the same type merge, the replacing step inherits *all* of the ordering constraints from the two steps being merged. This difference makes merging in the restricted case result in a more tightly-constrained plan, which in turn can prevent further merges from being performed. Yang thus considers a restricted form of the *multiagent plan coordination problem* in which the initial plan is assumed to be consistent, and the challenge is to optimize the plan by removing as many redundant steps as possible given the ordering restrictions on the plans. Despite these restrictions, the problem remains NP-Hard [22].

To adapt our plan-space coordination algorithm to Yang's restricted problem, we need to tell it to ignore conflicts, and then adjust how step merge flaws are repaired. To repair plan step merge flaws, such that a step s_i can be replaced by step s_k, we now do the following:

1. For each temporal constraint $t \in \prec_T$ of form $\langle s_i, s_j \rangle$ add new constraint of form $\langle s_k, s_j \rangle$ to $\prec_T$.
2. For each temporal constraint $t \in \prec_T$ of form $\langle s_j, s_i \rangle$ add new constraint of form $\langle s_j, s_k \rangle$ to $\prec_T$.

[5] Yang speaks in fact of an arbitrary *set* of steps as being redundant, but redundancies between more than two steps can be repaired in an iterative manner, so we present a pairwise description.

[6] More generally, Yang assumes that the two steps can be replaced by a single step, which he also assumes can be easily and uniquely determined.

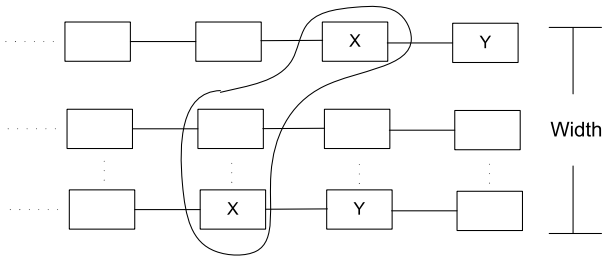

Figure 2: Visualizing Yang's Plan Coordination Algorithm

3. For each temporal constraint $t \in \prec_T$ of form $\langle s_i, s_j \rangle$ remove link from $\prec_T$.
4. For each temporal constraint $t \in \prec_T$ of form $\langle s_j, s_i \rangle$ remove link from $\prec_T$.
5. Remove s_i from S.

All other aspects of our algorithm remain the same, save the bounding mechanism. Significantly, because plan step merging in this problem formulation only tighten the temporal structure of the plan, we need only count the number of plan steps that can merge with other steps that remain unordered with respect to these steps. Although this still is often an overestimate of the quality of the plan, it serves as an extremely good bound for evaluating the potential of a given plan, as we show empirically, later in this paper.

4.2 Yang's Dynamic Programming Algorithm

To solve his restricted plan coordination problem, Yang [22] developed an optimal plan coordination algorithm based on a standard dynamic programming approach to the Shortest Common Supersequence problem [5]. In essence, the dynamic programming algorithm identifies the optimal way to merge steps in the POCL plan by recursively determining the optimal way to merge steps in the various plan "prefixes" of the POCL plan. Here, a plan prefix is a plan formed by performing vertical "cuts" through the partial order of plan steps and only keeping steps on one side of the cut. Optimal solutions to the subproblems are stored in a multidimensional table, and when the table has been completely constructed, the algorithm performs a polynomial search through the table to find the optimal set of steps to merge.

Figure 2 illustrates the dynamic programming algorithm graphically. Intuitively, Yang's algorithm first identifies possible cuts through the set of plans, where a cut groups together plan steps that are unordered with respect to each other. Within a cut, his algorithm then identifies sets of steps that can be merged, such as the steps marked with an "*X*" in the slice indicated in the figure. Yang's algorithm recognizes that plan steps that must be ordered to occur at strictly different times cannot possibly be merged together, and so it first groups together only steps that could co-occur, and then only looks for combinations within those sets for merging. Because the NP-Hard nature of the problem arises from the search for all possible combinations of identically-redundant steps in a set, Yang's algorithm bounds the size of the set to only the steps that could co-occur. His assumption (below) that the number of goals (plans) to merge remains constant means that the "width" of a slice (the size of any co-occurring set of plan steps) is bounded by a constant. As the problem scales, the plans get longer, but this only leads to a polynomial increase in computation because the number of slices grows but the maximum size of each slice does not.

We can see the complexity of Yang's algorithm by considering the number of cuts through a set of n plans to be coordinated. For the n plans, each of length d, the number of cuts is d^n, and for each cut of width n, Yang's algorithm has to consider merging all possible subsets of steps in the slice, which is 2^n. This means the complexity of Yang's dynamic programming algorithm is worst-case $O(2^n d^n)$. Thus, Yang's algorithm has time complexity that grows polynomially with the length of the plans to coordinate, but exponentially in the number of plans n.

4.3 An Analytical Comparison

If we compare the worst-case computational complexity of Yang's algorithm and our algorithm, it is clear that our algorithm is no better than Yang's algorithm, as Yang's algorithm is exponential with respect to the number of agents n to be coordinated, whereas ours is exponential with respect to the number of flaws in the plan (which in the restricted problem Yang considers, corresponds to the number of pairs of steps m to merge), which in the worst-case will grow quadratically with n and d.

However, unlike Yang's dynamic programming algorithm, our plan coordination algorithm is sensitive to the degree of interaction between the agents, or their *coupling*. In a multiagent system, the degree to which agents interact can range from being *uncoupled*, meaning that no action an agent takes can materially impact another agent (the agents are completely independent), to being *fully coupled*, meaning that every action an agent takes will materially impact all other agents (the agents are completely interdependent). We can characterize the degree of coupling between agents in terms of the number of flaws in the coordination problem (in this case, the number of merge pairs m). Agents who are *uncoupled* will have no steps to merge ($m = 0$), in which case the complexity of our algorithm will be a constant. At the other extreme, agents whose plans are *fully coupled* will have a number of steps to merge that scales with the size of their plans ($(nd)^2$), making the complexity of our algorithm exponential. Such agents will typically be better off planning from the outset as a team [16].

Somewhere in between these two extremes are situations where agents are *loosely coupled*, which we define as being the case when the number of interactions between the agents grows sublinearly with the number of agents. One simple example of such a sublinear relationship is to say that, as the number of agents grows linearly, the number of merge pairs m grows *logarithmically*. When this is the case, the complexity of our algorithm remains polynomial as the number of agents n scales.

This result is significantly smaller than Yang's complexity result, which remains exponential in the number of agents because of his use of a dynamic programming algorithm that is not affected by the interaction of the agents' plans. In contrast, our algorithm is sensitive to not only the number of steps in the plan, but the way in which they can be merged. Thus, as long as agents remain loosely-coupled, our algorithm can exploit this to reduce the complexity of the multiagent plan coordination problem in ways that Yang's algorithm cannot.[7]

4.4 An Empirical Comparison

We have shown how, in making reasonable assumptions about characteristics of the multiagent plan coordination problem, the worst-case performance of our search algorithm becomes tractable. However, we cannot always rely on the assumption that agents are loosely coupled, as there may be situations in which agent plans are highly interactive, but still need to be coordinated.

In this subsection, we demonstrate how, even when we relax the assumption that agents are loosely coupled, our use of a branch-and-bound approach to plan coordination can have significant computational benefits in the *expected case* performance of our algorithm, especially in comparison to Yang's algorithm. To make the comparison, we measure the computational cost of the algorithms in terms of the number of step comparisons that they need to do. For Yang's algorithm, for each of the d^n entries in the dynamic programming table (corresponding to the number of "cuts" through the agent plans), he considers 2^n possible subsets to merge, but we will instead be less stringent and assume that he only performs n^2 pairwise comparisons. In our algorithm, although in this restricted case it turns out that all possible pairs of steps to potentially merge can be found once at the beginning, we will handicap our algorithm by saying for each of the 2^m search states generated, it must compare all $(nd)^2$ pairs of plan steps again.

To test Yang's algorithm against our own, we ran both our algorithm and an implementation of his algorithm on a series of restricted multiagent plan coordination problems consisting of randomly generated plans for an increasing number of agents. Although we have implemented Yang's algorithm for the purposes of an empirical comparison, we could not practically run it on problems beyond four agents. Thus, we ran his algorithm on problems up to this number, and then for problems up to this size and larger, we have simply calculated the number of step comparisons based on the size of the dynamic programming table his algorithm would have to construct to solve the problem given the number of agents and the size of each agent's plan.[8] This number effectively serves as a lower bound on the computational complexity of Yang's algorithm, as any implementation of his dynamic programming algorithm must construct the table in order to identify the optimal solution.

The complexity of Yang's algorithm is independent of the coupling of the agents, and scales as the number of agents n or the length of their plans d grow. Our algorithm differs from his in this regard, and thus we consider three coupled cases in which we test the performance of our algorithm. In one, the number of merge pairs in the problem $m = \log(nd)$. This is the *loosely coupled* case. The second is where $m = 0.2(nd)$, and we call this the *tightly-coupled* case.[9] The last is where $m = 0$, which we call the *uncoupled* case.

For Yang's algorithm and for each coupling case, we generated a random problem by building a totally-ordered plan for each agent. Each of the plans consists of fifteen steps of different types, with a total of m merges between the agents possible. For the loosely-coupled, tightly-coupled, and uncoupled cases, the number of merges m was calculated accordingly. This conjunctive set of plans was fed as the initial input to each algorithm, and each algorithm was run until the optimal solution was returned. For each number of agents, we generated thirty problem instances and computed the average number of plan-step comparisons made by each algorithm. The results of our experiments are given in Figure 3.

As we can see in the graph, for these randomly-generated problems, our branch-and-bound algorithm clearly dominates in all coupling cases, especially in the uncoupled case, in which the algo-

[7] This is one reason why the complexity of Yang's algorithm is not a function of the number of types of plan steps, as counting the types would be one way of measuring the degree of similarity between different plans.

[8] The size of the table is the size of the agents' plans (assuming they are all of the same length) to the power of the number of agents.

[9] We used a coefficient of 0.2 because we did not want to assume that the agents were *fully-coupled*, which would be a degenerate instance of the problem, but still wanted the step merge flaws to scale linearly with the number of agents.

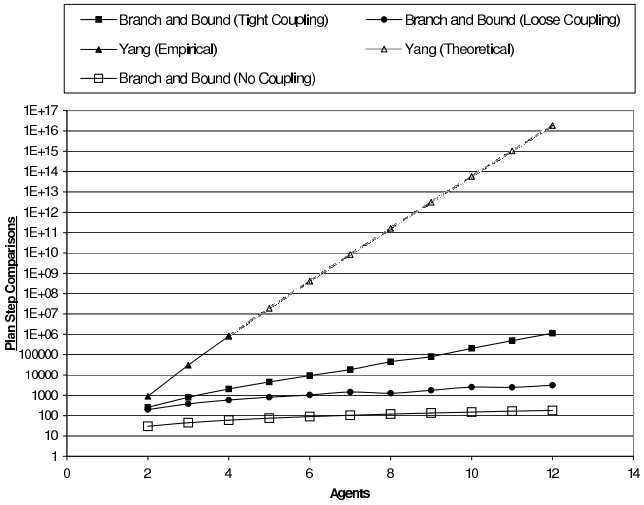

Figure 3: Empirical Comparison

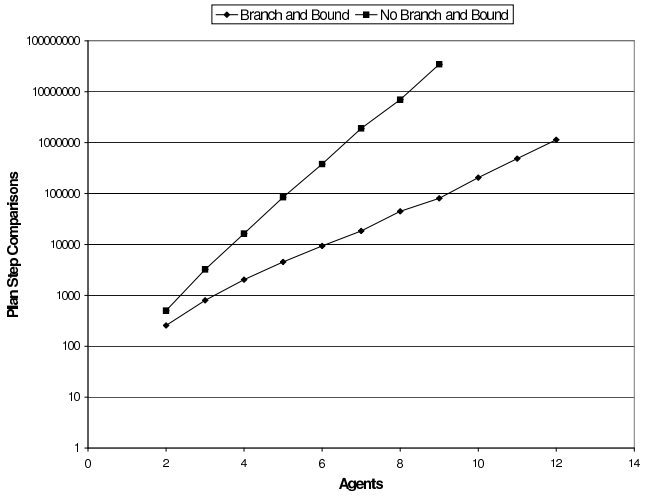

Figure 4: Branch-and-Bound Effectiveness

rithm can determine by looking at the starting plan that there are no merges to be performed, and then terminate the search before any more states are visited. Surprisingly, even as the number of merges grows in the tightly-coupled case, our algorithm scales well. This performance is almost entirely attributable to the effective bounding mechanism we use in our search, as the algorithm is able not only to find the optimal solution quickly using a depth-first search, but is able to use this optimal solution to prune large parts of the search space. Specifically, for these randomized problem instances, the algorithm could quickly find a solution in which it had implemented the majority of step merges, and could use this solution to bound much of the remaining search space.

We can see the effectiveness more clearly by comparing the algorithm's performance in the tightly-coupled case against the algorithm without the bounding mechanism. In Figure 4, we see the results of running our algorithm on the tightly-coupled problems from above, and on the same set of tightly-coupled coordination problems when the bounding mechanism is disabled, computing the average number of plan-step comparisons made (note again the graph's logarithmic scale). As we can see from the graph, the bounding test is a key component of our algorithm's efficient operation.

5. CONCLUSION AND DISCUSSION

In this paper, we have shown that assumptions made by past work on the single-agent plan coordination problem do not generally hold for the multiagent plan coordination problem. As a consequence, past methods [22] may not be well suited for solving the multiagent plan merging problem, as they may well be intractable as the number of agents grows. We have described what new assumptions we can make about the general multiagent problem, specifically the *loosely-coupled* multiagent problem, thus allowing us to derive an efficient and optimal plan coordination algorithm whose complexity compares favorably to the previous state-of-the-art single-agent plan merging techniques. Other current and related work not described in this paper includes work showing the efficiency of our algorithm on coordination problems more general than the ones that Yang considers, as well as work exploring ways of solving multiagent plan coordination problems in a distributed manner, [6].

Our future work includes the task of demonstrating more formal properties of our algorithm for coordinating *hierarchical* plans, combining the analytical work we have performed in this paper with the initial empirical results demonstrated in our earlier work [7], and the development of better search techniques to improve the expected-case performance of our search algorithm for practically-sized problems.

6. ACKNOWLEDGMENTS

This material is based upon work supported by NASA Training Grant NNG04GN49H, as well as the DARPA/IPTO COORDINATORs program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

7. REFERENCES

[1] BÄCKSTRÖM, C. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research 9* (1998), 99–137.

[2] BLUM, A., AND FURST, M. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (1995), pp. 1636–1642.

[3] BOUTILIER, C., AND BRAFMAN, R. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research 14* (2001), 105–136.

[4] CLEMENT, B., AND DURFEE, E. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proceedings of the Third International Conference on Autonomous Agents* (1999), pp. 252–259.

[5] CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. *Introduction to algorithms*, 6th ed. MIT Press and McGraw-Hill Book Company, 1992.

[6] COX, J., BARTOLD, T., AND DURFEE, E. A distributed framework for solving the multiagent plan coordination problem. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems* (2005).

[7] COX, J., AND DURFEE, E. Discovering and exploiting synergy between hierarchical planning agents. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems* (2003), pp. 281–288.

[8] DURFEE, E. H., AND LESSER, V. R. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics 21*, 5 (1991), 1167–1183.

[9] EL FALLAH SEGHROUCHNI, A., AND HADDAD, S. A coordination algorithm for multi-agent planning. *Lecture Notes in Computer Science 1038* (1996), 86–94.

[10] EPHRATI, E., AND ROSENSCHEIN, J. S. Divide and conquer in multi-agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (1994), pp. 375–380.

[11] HADAD, M., AND KRAUS, S. Combining temporal information by collaborative planning agents. In *AAAI Workshop on Planning with and for Multiagent Systems* (2002), pp. 23–30.

[12] HORTY, J. F., AND POLLACK, M. E. Evaluating new options in the context of existing plans. *Artificial Intelligence 127*, 2 (2001), 199–220.

[13] KAMBHAMPATI, S., TENENBAUM, M., AND LEE, S. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of the 9th National Conference on Artificial Intelligence* (1991).

[14] KAUTZ, H., AND SELMAN, B. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (1996), pp. 1194–1201.

[15] PECORA, F., RASCONI, R., AND CESTA, A. Assessing the bias of classical planning strategies on makespan-optimizing scheduling. In *Proceedings of the 16th European Conference on Artificial Intelligence* (2004), pp. 677–681.

[16] PYNADATH, D., AND TAMBE, M. An automated teamwork infrastructure for heterogeneous software agents and humans. *Journal of Autonomous Agents and Multiagent Systems 7* (2003), 71–100.

[17] THANGARAJAH, J., PADGHAM, L., AND WINIKOFF, M. Detecting & exploiting positive goal interaction in intelligent agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems* (2003), pp. 401–408.

[18] TONINO, H., BOS, A., DE WEERDT, M. M., AND WITTEVEEN, C. Plan coordination by revision in collective agent-based systems. *Artificial Intelligence 142*, 2 (2002), 121–145.

[19] VON MARTIAL, F. Interactions among autonomous planning agents. In *Decentralized AI*, Y. Demazeau and J.-P. Muller, Eds. North Holland, 1990, pp. 105–119.

[20] WELD, D. S. An introduction to least commitment planning. *AI Magazine 15*, 4 (1994), 27–61.

[21] WILKINS, D. E., AND MYERS, K. L. A multiagent planning architecture. In *Artificial Intelligence Planning Systems* (1998), pp. 154–163.

[22] YANG, Q. *Intelligent Planning*. Springer-Verlag, Berlin, 1997.