A Distributed Framework for Solving the Multiagent Plan Coordination Problem

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

ABSTRACT

We examine whether and how the Multiagent Plan Coordination Problem, the problem of resolving interactions between the plans of multiple agents, can be cast as a Distributed Constraint Optimization Problem (DCOP). We use ADOPT, a state-of-the-art DCOP solver that can solve DCOPs in an asynchronous, parallel manner using local communication between individual computational agents. We then demonstrate how we can take advantage of novel flaw-assignment strategies and plan coordination algorithms to significantly improve the performance of ADOPT on representative coordination problems. We close with a consideration of possible advances in framing our DCOP representation of the Multiagent Plan Coordination Problem.

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, Distributed constraint optimization

1. INTRODUCTION

The *Multiagent Plan Coordination Problem* (MPCP) 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. Our work [6] has explored a coordination algorithm that would systematically resolve flaws in multiagent plans. Although there was nothing inherent in this work that required the algorithm to be run centrally, we did not specify any protocol to allow agents to solve the problem in a decentralized manner.

Since the MPCP is fundamentally a multiagent problem, it is natural to think about ways of utilizing multiple agents to help to solve the coordination problem together, rather than arbitrarily relying on a single agent to carry out the task alone. The problem of how multiple agents solve a MPCP in a decentralized manner can be seen as a type of Distributed Constraint Optimization Problem (DCOP). Our work in this paper thus maps the decentralized MPCP into a DCOP, where coordination flaws (such as conflicts or redundancies between agents' actions) are treated as variables, where flaw repairs are the possible values that the variables can take on, and where temporal and causal relationships between actions impose constraints on legal flaw repair combinations.

By casting our decentralized MPCP as an instance of a DCOP, we can exploit the large body of work that has been going into developing fast and powerful techniques for solving DCOPs. Specifically, in this paper we make use of ADOPT, a distributed constraint optimization framework developed by Modi, Shin et. al. [11]. We show how many of the features of ADOPT can be gainfully employed for solving decentralized multiagent plan coordination problems. We also show how to extend the ADOPT framework to take advantage of problem structure to improve its performance on representative MPCPs.

The contributions of this paper are therefore threefold. First, we examine the degree to which the multiagent plan coordination problem can be solved in a decentralized manner by modeling the problem in the well-characterized DCOP framework. Second, we present a case study in applying domain-independent tools for solving DCOPs (in this case, ADOPT) to a specific multiagent problem. Third, we introduce a new strategy for solving the decentralized multiagent plan coordination problem within the DCOP framework through an innovative blending of general-purpose DCOP techniques (such as exploiting problem locality) and our (problem-specific) plan-coordination algorithms.

The remainder of this paper is structured as follows. In Section 2, we describe our formalization of the Multiagent Plan Coordination Problem (MPCP), and characterize our plan-space search algorithm designed to solve the MPCP. In Section 3, we show how the MPCP can be cast as a Constraint Optimization Problem (COP). In Section 4, we illustrate how we can take our COP formulation of the MPCP and (with some appropriate modifications) use ADOPT, a state-of-the-art DCOP framework, to solve the MPCP. In Sections 5 and 6, we explore and evaluate locality-based flaw-resolution strategies and ADOPT's own bounded-error approximation technique, both which significantly improve ADOPT's performance on solving the MPCP. Finally, in Section 7 we present our conclusions and future work.

2. BACKGROUND

In this section, we review the plan and plan problem formalisms used in our work, as well as our multiagent plan coordination algorithm. The concepts described in this section are essentially identical to those described in [6] (and are themselves derivatives of those described in [3, 13, 1]), but we summarize them here (though in less detail) for the sake of making this paper self-contained.

2.1 Planning Concepts and Problems

A standard definition of a grounded partial-order, causal-link (POCL) definition of a plan [13] is as follows:

DEFINITION 2.1. *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*. In the single-agent planning problem, a plan flaw is either a *causal link threat flaw* or an *open condition flaw*.

DEFINITION 2.2. *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 $\langle s_j, s_k \rangle \notin \prec_T$.*

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 2.3. *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).

A plan may be *consistent* (meaning it has no flaws) but still may not be *optimal*. Although there are many ways of defining plan optimality, a standard measure of optimality adopted in the planning community is the number of steps in the plan (where fewer is better), and we will adopt it here too.

2.2 Multiagent Plan Coordination

Although there has been a significant amount of work done on the single-agent planning problem, less attention has been paid to the problem of formulating plans for multiple agents to carry out. Because of the inherent complexity of the multiagent planning problem, many researchers [8, 4, 12, 5] have explored a divide-and-conquer approach to multiagent planning, in which agents individually plan for themselves, and then *coordinate* with each other by adjusting their plans given the presence of the other agents. This divide-and-conquer approach is a compelling one to adopt for the multiagent planning problem, as long as the process of *coordinating* the independently-formed plans of the agents can be done in an efficient manner, potentially exploiting the computational parallelism possible because multiple agents can work on the coordination problem simultaneously.

The *multiagent plan coordination problem* can be seen 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).

Just as the single-agent planning problem can be seen as the problem of transforming an *inconsistent* plan into a *consistent* plan, so can the Multiagent Plan Coordination Problem. Before giving a formalization of plan flaws in the multiagent plan coordination problem, we first describe how the POCL plan model is extended to handle the *concurrency* that can arise in multiagent planning and execution. Specifically, the operator model is augmented with *in-conditions* (or "during" conditions) [4], which describe the state of the world that holds during the execution of a plan step (this augmentation also relaxes the assumption that actions are instantaneous). Operators (and thus steps) are described by their preconditions, inconditions ($in(s_i)$), and postconditions. With this extended action model, a multiagent POCL plan can be described as follows:

DEFINITION 2.4. *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 $(\langle s_j, s_i \rangle \in \prec_T) \lor (\langle s_i, s_j \rangle \in \prec_T)$. 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 $=$.

One new source of inconsistency in multiagent plans is a *parallel step threat flaw*:

DEFINITION 2.5. *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 based on 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,

---

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

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.

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 2.6. *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 [14]. 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 [14]) 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 $\prec_C$.

4. Remove s_i from S.

2.3 A Multiagent Plan Coordination Algorithm

To solve the MPCP, in work described elsewhere [6], 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.

3. THE MULTIAGENT PLAN COORDINATION PROBLEM AS A CONSTRAINT OPTIMIZATION PROBLEM

In our work on multiagent plan coordination [6], no explicit commitment was made with respect to how the coordination algorithm was to be deployed, either in a centralized or decentralized manner. To demonstrate the flexibility of our work, we have integrated it with a distributed constraint optimization framework called ADOPT [11], thus allowing multiple agents to solve an MPCP concurrently.

To perform this integration, we first illustrate how we can cast the MPCP as a *constraint optimization problem*, or COP. A constraint optimization problem consists of n variables $V = x_1, x_2, ..., x_n$, where the possible values of each variable are drawn from a set of discrete domains $D_1, D_2, ..., D_n$. The goal of the problem is to find some settings of values to variables that minimizes the *global cost function* over the set of variables. The global cost function is computed by aggregating the costs of the violated constraints in the problem.

The concept of mapping a planning problem to a constraint satisfaction or optimization problem is not a new one. Do and Kambhampati [7] describe a method of translating GraphPlan's planning graph [2], into a Constraint Satisfaction Problem (CSP) that can then be solved using standard CSP solvers. More recent work by Lopez and Bacchus [9] extends this work, bypassing the Graphplan structure altogether to better exploit the structure of the planning problem, resulting in even better computational performance as well as the generalization of their method to richer planning models.

Given the previous description of the MPCP, we can cast it as a COP, where the optimization measure we use is identical to that of the coordination problem, that of minimizing the number of steps, subject to the constraints that all threat flaws are resolved, and that the temporal constraints on the steps in the plan remain acyclic.

We convert an MPCP to a COP in the following way. Given a multiagent plan coordination problem M, we will create a variable for each possible plan step merge flaw m, and for each possible threat flaw t. For each step merge flaw $m = \langle s_a, s_b \rangle$, we create a domain of size two, $\{i, m\}$, where i corresponds to the redundancy being ignored and m corresponds to the merge being performed, the second step substituting for the first step. For a causal-link threat flaw $t = \langle s_a, s_b, s_c \rangle$ (meaning step s_c threatens a link between s_a and s_b), we create a domain of size three ($\{i, p, d\}$), where i corresponds to the threat being ignored, p corresponds to the threat being resolved by promoting the clobbering step before the link, and d corresponds to demoting the clobbering step after the link. Note that the threat can be ignored and a solution is still possible if either step involved in the causal link, or the clobbering step itself, is removed by a step merge.

One of the difficulties we encountered when mapping the MPCP

to a constraint optimization problem is that the standard COP formulation as used in ADOPT requires that the set of variables and constraints on the set of variables are static; they cannot change as assignments are made to variables. However, a feature of the multiagent plan coordination problem is that as steps are merged, new flaws can be introduced.

For example, consider the multiagent plan in Figure 1. The numbered boxes are steps (boxes with g symbols are goal steps), and lowercase letters represent conditions (effects or preconditions). An edge labeled with a condition represents a causal link between the steps the edge connects. Conditions adjacent to a step and not on an edge are considered effects of the step. Here, step 2 can merge with step 4, such that 2 replaces 4 in the plan, because 2 achieves condition c. In its initial form, there is no step merge flaw involving step 3, but if the step merge is performed between 4 and 2, then the causal link between step 3 and 4 is removed, reducing step 3's necessary effects to b, allowing step 5 to merge with it. In addition, after performing this second merge, there will now be a causal link between step 5 and step $g1$, and a new threat between step 1 and this link.
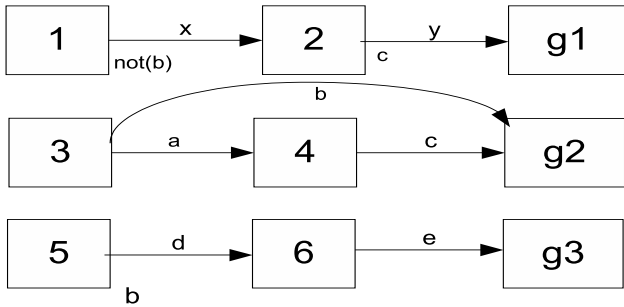

Figure 1: Example Coordination Problem

If we were simply to create a variable for each threat or step merge that was identifiable before any step merges are performed, our mapping of instances of the MPCP to the COP formulation would be incomplete, and some threats and step merges would be overlooked. Because ADOPT requires a static constraint network, we need to initialize it with a complete representation including all flaws that are contingent on the resolutions of other flaws. As we saw in the example, step merge flaws can arise because step merging changes the causal links in the multiagent plan, thus potentially changing the necessary effects of plan steps as well. In the limit, all of a step's outgoing causal links could be removed because of step merging, allowing it to be removed without needing another step to replace it (we can think of this step as merging with the "null" step). To deal with this, we will need to keep track of the presence of the plan steps whose removal could change the necessary effects of another step, allowing it to be merged away. We do this by creating variables representing the presence or removal of relevant plan steps in addition to step merge and threat flaws. Step variables have domains of $\{p, r\}$, as the step can either be present or removed from the plan.

The set of step merge flaw variables and plan step variables can be constructed in a straightforward, iterative way. We begin by identifying all step merges that are possible without the removal of any other steps. For each step merge flaw $m = \langle s_a, s_b \rangle$, we create the step merge variable, as well as a step variable for s_a. We now check to see if there are additional step merges that could take place if any of the steps we just created variables for were removed. As we saw in the example, new step merges may be possible because

the removal of steps may reduce the set of necessary effects of other steps that support these removed steps. If it ever is the case that *all* of the steps that a step s is supporting become removable, then we create a special step merge variable $m = \langle s, null \rangle$, allowing step s to be removed if it does not causally support any other steps in the plan. The process repeats until no new step merges are discovered. This process will terminate in at most n iterations (where n is the number of steps in the multiagent plan) as there are at most n steps that can be removed from the plan. Each iteration takes worst-case n^2 time (the amount of time it takes to perform a pairwise comparison of each steps' effects with the updated necessary effects of each other step).

After the set of all possible merge pair variables has been computed, handling threats can be accomplished in a more straightforward way. First, all initial threats present in the MPCP are identified, and variables are created for them. Then, if there exists a threat $t = \langle s_a, s_b, s_c \rangle$ and a step merge flaw $m = \langle s_a, s_d \rangle$, then we create a new threat variable representing $t' = \langle s_d, s_b, s_c \rangle$, as if m is resolved such that s_d replaces s_a, the threat to the adjusted causal link needs to be dealt with. Since the number of possible step merge flaws in the plan is polynomial in the number of steps in the multiagent plan, the number of threat flaws is as well, and the set of threat variables can also be constructed in time polynomial in the size of the plan.

3.1 Constructing Variable Constraints

The relations between flaw variables are specified by constraints over the variables. We will describe the constraints between flaw variables by indicating the nogood sets between the variables, that is, combinations of values that the connected variables cannot take. For example, a binary constraint between variables v_i and v_j can be described by a set $\langle (0,1), (1,0) \rangle$, meaning that the constraint prevents variables v_i and v_j from being assigned values 0 and 1, or 1 and 0 respectively. To illustrate of the constraints, we will again make reference to the MPCP in Figure 1, and its COP representation, illustrated in Figure 2.
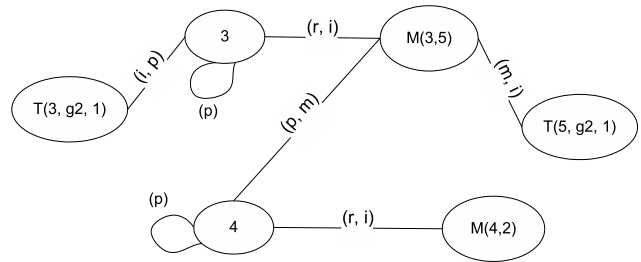

Figure 2: Matching COP

A **Handle Threats** constraint ensures that a threat flaw be handled either by ordering the steps, or by removing any of the three steps involved in the threat. For a variable t representing a threat flaw $\langle s_a, s_c, s_d \rangle$, we add a constraint c between the threat variable whatever subset S of the three steps involved in the threat have variables in the COP, where $c = \langle (i, p_1, ... p_{|S|}) \rangle$. This constraint ensures that a threat is resolved if all of the plan steps in the threat are still in the plan. If the threat arises after a step merge, then the constraint includes the step merge variable that creates it as well, and $c = \langle (i, p_1, ... p_{|S|}, m) \rangle$ (as the threat does not have to be addressed if the step merge is not performed). In Figure 2, there is a Handle Threats constraint between the variable for threat $\langle 3, g2, 1 \rangle$ and step variable 3, as the threat must be either be resolved or step 3 must be removed. There are a polynomial number of threat flaws

in the plan, and each threat constraint is only of arity five, so these constraints can be calculated in polynomial time.

A **No Transitive Merges** constraint prevents a step from replacing another step if it is removed. Formally, for a variable m representing step merge flaw $\langle s_a, s_b \rangle$ and variable s representing step s_b, we add a constraint c between m and s, where $c = \langle (m, r) \rangle$. Note that this constraint is only added if a variable representing s_b is in the COP (meaning that s_b could be removed from the plan). Figure 2 has no constraints of this type. Computing these binary constraints involves looking at each pair of plan step merge flaws, which are polynomial in number. Thus, these constraints can be computed in time polynomial in the size of the plan.

A **Constrain Step Merges** constraint ensures that a merge between a step s_a and step s_b will not happen unless the steps needing necessary conditions from s_a that s_b cannot support are not present in the plan. For each step s_c that step s_a causally supports, but s_b cannot, we create a constraint between step merge variable $\langle s_a, s_b \rangle$ and step variable s_c where the constraint is (m, p). In Figure 2, there is a constraint of this type between the variable for step merge $\langle 3, 5 \rangle$ and step variable 4, as the step must be removed before the merge can be performed. These constraints can be computed simultaneously with the step merge variables, thus involving no extra computation.

A **Constrain Step Removal** constraint prevents a step from being removed unless another step replaces it. For each variable representing a plan step s and the set $M = \{m | \langle s, s' \rangle\}$ of merge pair flaw variables (that is, all merge flaw variables that remove s from the plan) add a constraint between the step variable representing s and this set of step merge variables $\langle (r, i_1, \ldots i_{|M|}) \rangle$. In Figure 2, there is a constraint of this type between step merge variable $\langle 3, 5 \rangle$ and step variable 3, as well as between $\langle 4, 2 \rangle$ and step variable 4. There will be at most n of these constraints (one for each step that could be removed), each with maximum arity of $n \quad 1$ (the maximum number of step merge flaws involving the removal of the same step). Thus, these constraints can be calculated in polynomial time.

A **Reward Step Removal** constraint ensures that step merging will always be done, if possible. For a variable s representing a plan step, we add a unary constraint $\langle (p) \rangle$. There are constraints of this type on plan steps in Figure 2. These unary constraints can trivially be computed in linear time. Not that, in general, all conceivable merges cannot be done, so these unary constraints will tend to overconstrain the problem. This is a motivation for doing constraint optimization, instead of constraint satisfaction.

A **Temporal Consistency** constraint to ensure that the multiagent coordinated plan remains acyclic in terms of the temporal constraints over the steps in the plan. Unlike the previous constraints, which can be generated in polynomial time, is not computationally feasible to explicitly enumerate all possible nogoods for a Temporal Consistency constraint (and thus they are not captured in Figure 2), but the constraint can be checked by implementing possible flaw resolutions in the temporal network representing the agents' plans, and checking the resultant network for cycles, which can be done in $O(n^2)$ time. This, in effect, is how we will test this constraint in the COP encoding of an MPCP.

In the worst case, temporal consistency constraints would hold over *all* flaws in the system, hampering the ability of ADOPT to solve the COP formulation of the MPCP in parallel. However, depending on the structure of the MPCP, we can *partition* the flaw variables so as to restrict the scope of the constraints to just be over flaw variables inside each partition. To partition the variables in a straightforward way, we form groups of flaws based on the plans the flaws are associated with. That is, two flaws belong to the same partition if they involve steps from the same plan. Furthermore, two

flaws that share a plan with a common third flaw are also placed in the same group. Finally, a flaw cannot be in more than one group, so in the worst case all flaws will be in a single partition.

In our work, we assume that agents require consistent plans (meaning acyclic plans without any causal-link threat flaws), but also want plans that have as few steps as possible (where the greatest number of step merge flaws have been repaired). To produce this, we make the cost of violating all constraints except the Reward Step Removal Constraint infinite, as such violations will result in inconsistent plans. The cost of violating a unary Reward Step Removal Constraint is one, as for each step merge we reduce the overall cost of the multiagent plan by one step.

4. DISTRIBUTED MULTIAGENT PLAN CO-ORDINATION USING ADOPT

Distributed Constraint Optimization Problem (DCOP) frameworks enable agents to converge on a globally consistent solutions to DCOPs in a distributed manner. Recently, Modi, Shin et. al. [11] have developed a DCOP framework called ADOPT. Advantages of ADOPT include the ability to optimize a global function via local communication, the ability for agents to compute their variable values in an asynchronous manner, and the ability to provide quality guarantees so that trade-offs between solution quality and computation time can be considered.

A straightforward way of using ADOPT is to convert the MPCP into a COP using our transformation from the previous section, and then create an ADOPT agent for each COP variable. The ADOPT agents then use the constraints and the ADOPT protocols to converge on a globally optimal solution. One obvious problem with this formulation is the fact that the MPCP involves an existing set of agents, and so it does not make sense to think of "creating" ADOPT agents. Modi, Shin et. al. [11] indicate how to address this concern, by allocating to each agent a *set* of variables, combining the set of variables into a single variable with a domain that is the cross-product of the domains of the original variables. For the MPCP, to minimize the size of these variable domains, we embed our coordination algorithm (Algorithm 1) in each ADOPT agent to prune invalid local solutions from this cross product of possible flaw variable choices. As illustrated in Figure 3, the agents first run the coordination algorithm on their domains, eliminating all values that result in infinite-cost constraint violations, and then iteratively communicate their remaining values to converge on a globally optimal set of flaw resolution choices.
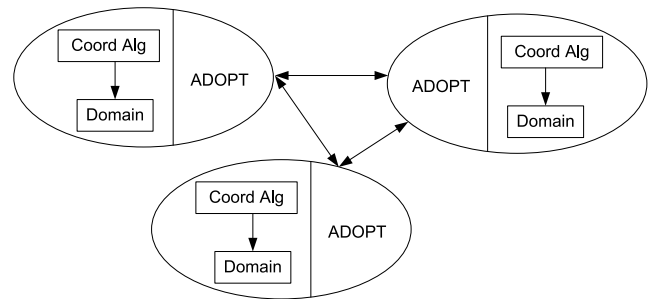

Figure 3: Distributed System View

Since the optimization constraints in ADOPT are not just inter-flaw variable constraints but *inter-agent* constraints, the scope of (and thus the number of agents involved in) the Temporal Consistency Constraints will depend on how the *agents* can be partitioned. Two agents belong in the same partition if any of their flaws share

the same partition, or if there is a third agent that has flaw variables that share partitions with both agents' flaw variables.

Finally, if each agent has responsibility for flaws that it is participating in, then the question of how these flaws are found remains unanswered. Although the development of protocols for flaw discovery is beyond the scope of this paper, it is not difficult to envision the agents using a simple pairwise plan exchange protocol, where each agent sends its plan to each other agent, and where the other agent responds with the relevant flaw information between the plans. We consider the development of more sophisticated protocols a challenge to be addressed in our future work.

5. ALTERNATIVE FLAW DISTRIBUTION STRATEGIES

For ADOPT to be able so solve the DCOP version of the MPCP, we need to figure out which flaws will be handled by individual ADOPT agents. In the previous section, we indicated how we could modify ADOPT to handle this issue, by having an agent handle multiple flaws, and then using the plan coordination algorithm to enable agents to prune locally-invalid combinations of values for these variables, thus improving on the speed of the search. However, the exact method of distribution was not specified.

In this section, we compare the performance of two alternative flaw distribution strategies, measuring the number of messages agents must exchange before ADOPT terminates with the globally-optimal solution. The first strategy we consider is a load-balancing one, in which step variables are given to the agent whose plan the step is from, but flaw variables are distributed randomly in equal numbers among the ADOPT agents. The second strategy attempts to exploit problem *locality*, by having agents take responsibility for the flaws that involve their associated plans in addition to the steps in their plans. Since coordination flaws by definition involve more than one agent's plan, when allocating a flaw shared by two or more agents, the agent with the fewest number of current flaws will take responsibility for the flaw (thus preserving some degree of load-balancing).

The advantage of locality-based flaw distribution is that flaws that involve the same plans are more likely to interact with each other via the problem constraints, which has two obvious benefits. First, since an agent uses the multiagent plan coordination algorithm to locally prune flaw variable combinations that result in constraint violations of infinite cost, it is more likely to prune more combinations, resulting in a faster overall search. Second, this locality-based assignment makes partitioning the agents (and thus restricting the scope of the Temporal Consistency Constraints) easier, as such a distribution only gives agents flaws that were part of the same original partition, and thus the original flaw partitions are preserved.

To test this hypothesis, we compared these two alternative flaw distribution strategies on randomly generated MPCPs. Given a set of agents, we organized them in a ring topology, such that each agent only interacts with the two agents on either side of it (thus providing some degree of locality to the problems).[3] For each agent, we created a plan consisting of ten steps, totally ordered. Then, we randomly generated plan step merge flaws and threat flaws between the agents by randomly selecting the first agent (in a uniform manner) and then picking its counterpart to participate in the flaw by randomly selecting one of the agent's neighbors. Since, as we indicated earlier, flaws can arise based on how other flaws are handled, with thirty percent probability we would make a step merge flaw dependent on an existing step merge variable. In ad-

dition, if a threat was created that threatened a step that could be merged away, we would also create a threat variable for the replacing step as well. For n agents, we created $n/2$ step merge flaws and $n/2 + n$ mod 2 threat flaws. For both alternative flaw assignment methods, we generated thirty random problems and computed the median number of messages that the ADOPT agents had to exchange before converging on the optimal solution,[4] increasing the number of agents from two to ten.
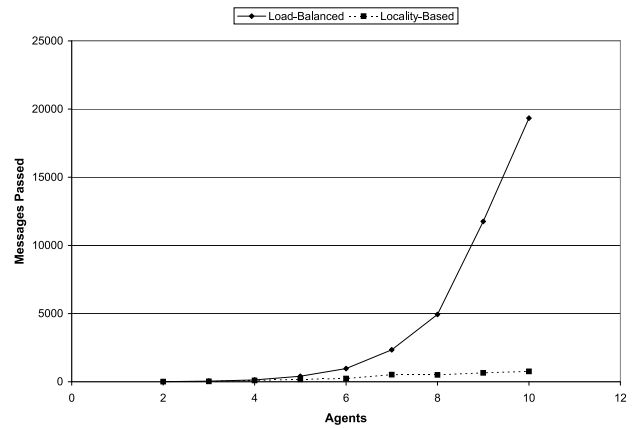

Figure 4: The Advantages of Locality-based Flaw Assignment

As we can see in the graphs of the performance of the Locality-Based and Load-Balanced strategies in Figure 4, the benefits of our locality-based flaw assignment strategy are substantial, significantly reducing the number of messages agents must exchange to converge on a globally optimal solution. A caveat, though, is that the load-balanced strategy suggests what might happen if there is no locality to exploit. In more tightly coupled problems, it could turn out to be the case that partial centralization [10] or even fully centralizing the MPCP and using Algorithm 1 alone, is more effective.

6. BOUNDED-ERROR APPROXIMATION

One of the key features of ADOPT is its ability to provide quantitative quality guarantees when time limitations require agents to settle for approximately optimal solutions. That is, when provided a bounding number, ADOPT will return a solution that is guaranteed to be no worse than the quality of the optimal solution, plus the bounding number. This functionality is particularly useful when solving MPCPs that do not have strong local structure, unlike the experiments in the previous section.

When given an approximate bound, ADOPT can solve COP formulations of MPCPs significantly faster than when it is required to return an optimal solution, even when the MPCPs lack any locality. To demonstrate this, we ran ADOPT on MPCPs in which the agents were composed of a fully-connected agent topology. That is, all agents interacted with all other agents in the problem. Otherwise, problems were generated the same way they were generated in the previous section. We then compared the performance of ADOPT on these problems by giving ADOPT a bound of zero and a bound of five (meaning that the agents would be willing to accept a solution which had five fewer steps merged than the optimal solution.) As we can see in the graph in Figure 5, this marginal sacrifice in optimality has had a significant impact in the performance

---

[3]Alternative topologies are considered in the next section.

[4]Messages passed is a reasonable metric as ADOPT agents do not have to perform any other computation-intensive tasks.

of ADOPT, even when it could not leverage problem locality. In addition, despite the bound, the approximate approach always returned solutions within one merge of the optimal solution.
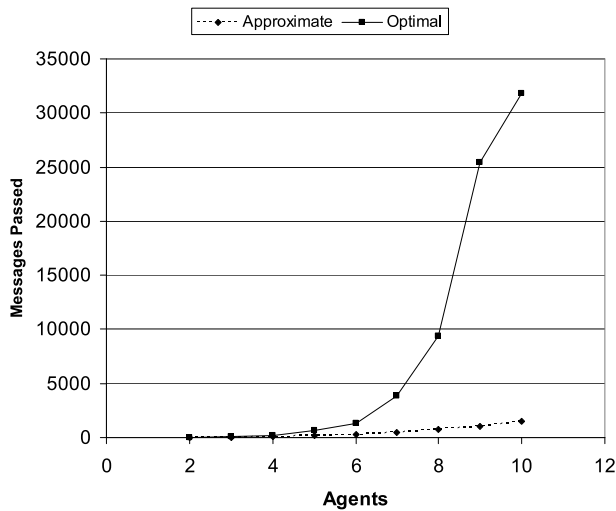

Figure 5: The Advantages of Bounded-Error Approximation

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a case study in mapping a domain-specific problem into a domain-independent DCOP formulation, so that we can make use of multiple agents in the system to solve the MPCP together. We have also explored novel flaw-assignment strategies aimed at improving the performance of ADOPT on solving the MPCP, and shown how even when these strategies prove ineffective, we can make use of ADOPT's bounded-error approximation functionality to quickly find near-optimal solutions. In the future, we plan to explore alternative possible COP encodings of the MPCP that contain more localized constraints, thus allowing more parallel computation, and to explore more fine-grained temporal partitioning strategies, aimed at decomposing the MPCP problem even further so as to better localize the temporal constraints on the COP encoding of the MPCP. Finally, we plan on more formally establishing the correctness of our reduction of the MPCP to a DCOP.

8. ACKNOWLEDGMENTS

9. 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] 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.

[6] COX, J., AND DURFEE, E. Efficient algorithms for multiagent plan coordination. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05)* (2005).

[7] DO, M. B., AND KAMBHAMPATI, S. Planning as constraint satisfaction: solving the planning graph by compiling it into csp. *Artificial Intelligence 132* (2001), 151–182.

[8] 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.

[9] LOPEZ, A., AND BACCHUS, F. Generalizing graphplan by formulating planning as a csp. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, p. 954.

[10] MAILLER, R., AND LESSER, V. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)* (2004), pp. 438–445.

[11] MODI, P. J., SHEN, W.-M., TAMBE, M., AND YOKOO, M. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal* (2004).

[12] 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.

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

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