ABSTRACT
Multiagent systems sometimes undergo changes that cause coordination commitments to become insufficient or out of date, such that the coordinated agent plans need to be repaired or replaced. When recoordination becomes necessary, disruption to the commitments made by the agents in their original plans should be minimized. We approach the problem of minimizing disruption by augmenting pre-existing coordination technology by developing metrics and automated processes for it to rank and potentially recommend new coordination commitments more rapidly. In this paper, we explain, examine, and evaluate our new metrics and processes, demonstrating empirically that flexible measures of disruption can streamline the coordination process.

Categories and Subject Descriptors
1.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search; 1.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence - Multiagent systems

Keywords
Coordination of multiple agents, Conflict resolution, Multiagent planning, Biased search, Disruption

1. INTRODUCTION
Agents acting in complex, dynamic environments must often adjust their plans to take into account actions of other agents, such as to avoid potential conflicts. Possible conflicts can be detected by selectively exchanging and comparing portions of agents' individual plans, identifying inconsistent expectations, and adding synchronization actions and/or blocking some action choices to ensure conflicts cannot arise. Reaching agreement on what coordination commitments to adopt, and implementing them (which could trigger further downstream commitments and negotiations), can incur significant overhead. For example, in one of the application domains that we have studied, reaching commitments for multi-national coalition operations [13] can involve numerous levels of negotiation and diplomacy.

Limiting Disruption in Multiagent Replanning
Thomas Bartold
University of Michigan
1101 Beal Ave
Ann Arbor, MI 48109-2110 USA
1-734-763-9074
tbartold@umich.edu

Edmund Durfee
University of Michigan
1101 Beal Ave
Ann Arbor, MI 48109-2110 USA
1-734-936-1563
durfee@umich.edu

If some of the agents need to later revise their individual plans in response to unexpected dynamic changes in the environment, prior coordination commitments can be imperiled. To avoid having to incur another round of significant overhead, the agents might prefer to coordinate their changed plans as similarly as possible to how they coordinated their previous plans. We refer to changes in coordination commitments as forms of disruption.

We argue that agents should consider disruption when deciding on coordination commitments to make in changing circumstances. Obviously, there are many factors besides disruption that should be weighed when deciding on a coordination solution, including reducing costs for joint activities and maximizing parallel and independent activity of agents. However, the emphasis of this paper is on disruption. In particular, the contributions of the work we present in this paper revolve around how disruption can be estimated when comparing alternative coordination solutions, and how the search for coordinated solutions can be effectively streamlined towards finding less disruptive candidates sooner.

In what follows, we assume that the plans of a group of agents have been initially coordinated, but that a change occurs that forces one or more agents to revise their plans such that some coordination commitments must be violated. Rather than starting the planning process over again from scratch, or even just starting the coordination process over again, we want to reuse the results of the prior planning and coordination process to guide the search to restore coordination in a manner that minimizes disruption to the previous coordination commitments, effectively repairing the coordinated solution to fit the revised plans.

The idea of attempting to reuse prior planning and coordination effort when adapting to new situations is far from new. Indeed, issues of how and when an agent should decide whether to honor its prior plan commitments versus reconsidering them have been at the core of research in areas such as plan repair/replanning [8], reactive plan execution architectures [6], and plan management [7]. Similar issues arise in the multiagent literature, in terms of how an agent that has already coordinated its plans with other agents should react to unexpected circumstances. Relevant work includes that on conventions that agents abide by when they change their intentions [5], on polite behavior that attempts to minimize impact on others [11], and on tolerating minor inefficiencies and adhering to suboptimal plans rather than prompting recoordination [4]. Our work adds to this thread of research by developing some rudimentary metrics for particular kinds of disruption, and by demonstrating how such metrics can be heuristically employed to improve coordination search efficiency. In contrast with PGP [4], where inconsistency implied inefficiency, here any inconsistency could be catastrophic, and
coordination becomes imperative. Because the need for coordination can be time sensitive, coordination efficiency is an issue, and could benefit from prioritizing the order in which conflicts are addressed, as in constraint satisfaction problems [14].

In the next section, we illustrate concepts of disruption using a simplified application domain that we later elaborate for experimental evaluation. We then turn to metrics for estimating disruption, which forms the basis for comparing alternative candidate solutions. Given a search mechanism that generates candidate solutions, we can rank these solutions based on the metrics. However, given the assumed dynamics of the environment, it could be the case that candidate solutions generated later are better, and it would be desirable instead to bias the search process to generate less disruptive solutions sooner. We outline our techniques for instituting this bias in Sections 4-5. In Section 6, we evaluate the effectiveness of these techniques, and then we outline directions of our future work in the concluding section.

2. SIMPLIFIED EXAMPLE

To date, the primary application domain in which we have studied disruption has been coalition operations, where the plans of coalition partners might unintentionally interact, and so the partners coordinate their plans, and recoordinate their plans, only when circumstances change [13]. For ease of exposition and experimentation, however, in this paper we will concentrate on a simplified problem of coordinated access to potentially shared resources, in terms of avoiding collisions of aircraft that could share corridors. The alternative routes and locations are represented as edges and nodes (respectively) in a graph, as in Figure 1. The first agent, A1, wants to move from location A to location D, and can do so either via location B or via location C.

Our research has emphasized coordinating plans generated through HTN planning techniques, in the tradition of NOAH [9] and NONLIN [10]. Planning begins with high level goals and iteratively refines abstract steps until a complete task network is created. The plan hierarchies generated are all strictly trees. By retaining the hierarchy, coordination can be achieved at various levels of abstraction. In Figure 2, we show how A1’s plan could be represented hierarchically [8,12] with three abstract plan steps and four primitives. For A1, the “Cross AD” abstract plan can be refined to either abstract subplan “Via B” or “Via C”. Each of these subplans can be refined into a pair of primitive move operations. The plans used in our example are assumed to be totally ordered, although this is not required by our methods.

An extreme example of overlap is when a second agent, A2, also needs to go from A to D, and has access to the same routes. It therefore has an identical plan to A1. In this case, the agents could coordinate by synchronizing their actions, such as by having A2 wait to begin its movements until it receives a signal from A1, and A1 sending that signal once it reaches D. Alternatively, the agents could impose constraints on how each will execute its plan, such as by having A1 commit to not move through location C, and A2 commit to not moving through B. These (and other possible) coordination solutions impose constraints on what agents are allowed to do, what they are required to do, and/or when they can do various actions.

Assuming that agents have agreed on particular commitments and have gone through the (often considerable) effort to reserve the resources and institute the agreements to implement them, then they should resist making wholesale changes if the world changes. For example, a third agent, A3, could enter the picture; in the worst case, its plan might be the same as those of A1 and A2. The simplest, and least disruptive way, of coordinating A3 would be to simply force it to work around the coordinated plans of the other agents. For example, it might simply wait until A3 signals it (once A2 is done) if A2 was constrained to first wait for A1. Or, if A1 and A2 had committed to complementary routes, then A3 either would need to be signaled by both of them (in which case it could follow either route) or one of them (in which case it would have to traverse the same route as that one). But, given the delays of either of these, it might be preferable for A3 to be permitted to begin as soon as one or both of the others have completed only their first legs. This imposes more demands on those other agents, and is more potentially disruptive. If A3 had a time-critical delivery to make, then it might want to precede one or both of the other agents, which could more significantly disrupt the previously expected behaviors of A1 and A2.

Again, as illustrated in this example, our claim is not that wholesale changes should not be permitted, but rather that, all else being equal, less disruptive solutions should be preferred. In this example, and in less tightly-coupled examples (where agents have overlapping but not identical plans), we would like to have measures of how much disruption is involved. Doing so requires a clearer characterization of the types of interagent commitments that can be made, and the costs of making and breaking them.

3. COORDINATION COMMITMENTS

Coordination commitments can be viewed as modifications to single-agent plans that constrain their execution and augment their actions to ensure coordinated outcomes. We assume plans are organized hierarchically, as in our previous example illustrated in Figure 2. A plan \( P_i \) (for agent \( A_i \)) is represented as a tree where each node is a plan step that can be identified (or labeled) uniquely as accomplishing some goal or subgoal. An and plan step (denoted by the connecting line between its children) or an or plan step can be refined, replacing an and step with all of its
children, and replacing an or step with any one of its children. The pre- and post-conditions of an abstract plan step can be summarized from those of its children [1,2]. Primitive plan steps are neither and nor or because they cannot be refined, and represent actual actions to be taken by the agent, where each action has standard STRIPS representations of the action's pre- and post-conditions. Primitive plan steps also have costs and durations, where the interval over which the plan step takes place is captured as start and end time points. The preconditions, postconditions, costs, durations, and start/end time points of non-primitive plan steps can be derived bottom-up from their descendants. A plan step can also have in-conditions, permitting it to model temporary changes to the world (brought about by some of an abstract step's subsumed primitives and undone by other primitives) as well as more permanent effects of the step.

A coordination solution $C = (A, P, S, B)$ is composed of a set of agents, $A = \{A_1, A_2, ..., A_n\}$, where the number of agents $n = |A|$, the individual agent plans, $P = \{P_1, P_2, ..., P_n\}$, that are each internally conflict-free, and coordination commitments that resolve conflicts between the agent plans. There are two distinct types of coordination commitments: blocking commitments $B = \{B_1, B_2, ..., B_n\}$, and synchronization commitments, $S = \{S_1, S_2, ..., S_n\}$, partitioned into subsets associated with each agent. Blocking commitments made by an agent capture promises by the agent to block off some of its options (restrict its choices for an or plan step) when those options are responsible for potential conflicts. Synchronization commitments represent a decision by an agent to engage in a signaling operation with other agents, either waiting for the other agent to signal it such that it can continue with its plan, or sending a signal to another agent to release that other agent to continue.

Whenever the plans of any of the agents change in response to domain dynamics, including when agents enter or leave the domain, the agents' previously-held coordination solution, $C = (A, P, S, B)$, can become obsolete. The disruption ($\delta$) caused by a new coordination solution ($C'$) as compared to the previous (reference) solution ($C$), can be calculated solely by examining the sets of coordination commitments in the solutions $D(S'; B'; S, B)$. The simplest metric is to just count the number of new commitments added into the new solution and the number of old commitments dropped from the previously adopted solution. However, this unrealistically treats all commitment changes as equally disruptive. Instead, we weight changes differently depending on the commitment type, as now described.

### 3.1 Synchronization Commitments

To avoid conflicting actions, one agent may (at a specific point in its plan) need to notify another agent that the other agent may proceed (from a particular point in its plan). In our simple example, for instance, when $A_1$ has completed the action "Move AB", it could release $A_2$ to take its first moving action. Each synchronization commitment can be represented by a 5-tuple ($A_i$, points, type, $A_j$, point). The type of synchronization commitment will be either wait (for another agent), or release (another agent to proceed). Each synchronization commitment has a dual; that is, for every wait there is a release at some other agent, and vice versa. These commitments always occur right before or after a particular plan step, specified for $A_i$ as point. This means that synchronization messages always occur between (primitive) plan steps; however, they can occur during more abstract plan steps, so long as exactly when during those steps is ultimately derivable as those steps are refined. $A_i$'s commitment above would be represented as ($A_i$, move-ab-end, release, $A_2$, cross-ad-start).

We assume that disruption is additive, so that the total disruption by some coordination solution can be computed as a summation over individual disruptive changes to commitments. In what follows, therefore, we first quantify the disruption due to individual commitment changes, and then we go on to explain how we determine the disruption to the agent when there are multiple elements in one set of commitments or the other.

In our example, at the most abstract level, $A_1$ could wait for $A_2$ to complete its plan before starting its own, resulting in a single coordination commitment for each of the agents. This would be represented as the coordination commitment:

$$S'_{11} = (A_1, \text{cross-ad-start}, \text{wait}, A_2, \text{cross-ad-end})$$ (1)

and its dual:

$$S'_{21} = (A_2, \text{cross-ad-end}, \text{release}, A_1, \text{cross-ad-start})$$ (2)

When looking for matches, our algorithm examines the attributes of the various coordination commitments. Weighting each of these attributes appropriately can scale the quality of the match. That is, a commitment that matches on four out of five attributes should have lower disruption than one that can only be matched with a single attribute. For example, assume that given a change to the agents' plans, one possible new commitment is:

$$S'_{11} = (A_1, \text{cross-ad-start}, \text{wait}, A_2, \text{move-cd-end})$$ (3)

Compared to the original commitment (Eq. 1), an attribute does not match, so there is a change in commitments and this contributes to the disruption to $A_i$. Differences (deltas between commitments) associated with different attributes can be weighted differently. In this case, the other agent's time point (point2) does not match, so a delta ($\delta$) for other, $\delta_{oi}$ should be added to the total computed disruption to $A_i$.

The total disruption is simply a multi-attribute summation over the deltas, where a weighting ($\delta$ value) is associated with each non-matching attribute. In the case of synchronization commitments for any agent $A_i$, there are four attributes that might not match1, and their weights are given as $\delta_i$ ($A_i$'s plan step time point), $\delta_{oi}$ (the type of commitment), $\delta_i$ (the other agent), and $\delta_{oi}$ (the other agent's plan step time point).

If the synchronization constraint had instead switched the order of the plan executions, then a new commitment could instead be:

$$S'_{11} = (A_1, \text{cross-ad-end}, \text{release}, A_2, \text{cross-ad-start})$$ (4)

Compared to the original commitment (Eq. 1), this has three attributes that differ (only the agents remain the same). The disruption would thus be the sum $\delta_i + \delta_{oi} + \delta_{oi}$.

In general, the disruption between a pair of commitments can be directly calculated from the two commitments by treating them as arrays, and applying the formula:

---

1 Since disruption is evaluated for each agent separately, the first attribute of all constraints applied to a specific agent must match.
where \( S_f[0] \), the first attribute, is the agent and will be the same for any commitments being compared. Mismatches in any of the other four attributes will contribute the appropriate \( \delta_x \). \( D_{x,k} \) is a measure of disruption to agent \( A_j \) and the brackets indicate the indices of the synchronization constraints in the two sets of commitments.

### 3.1.1 Unmatched Synchronization Commitments

Not all plan commitments will match between solutions; one set of commitments may be larger than the other. A synchronization commitment from the old solution that is unmatched in the new solution could be disruptive to an agent, but the amount of disruption should depend on the type of commitment. If \( A_j \) were prepared to wait for \( A_i \) to release it to proceed, we would expect the absence of that signal to be disruptive. We can identify this kind of situation by noticing that a wait commitment has been dropped from the set of commitments for the agent. \( A_j \) must notice that the commitment is missing, otherwise it could wait forever for release and fail to achieve its goals. On the other hand, should \( A_j \) have been expected to release \( A_i \) in the prior solution, but now that requirement is lifted, the disruption to \( A_j \) would be less, because sending a release that is no longer being waited for only wastes time and bandwidth, rather than leading to deadlock.

We thus assign different values, \( \delta_x \) and \( \delta_y \), for the dropped wait and release modifications respectively. As per the above, we would qualitatively expect that \( \delta_x > \delta_y \).

When a commitment in the new solution has no corresponding commitment in the old, we need to include the disruption caused by the added unmatched commitment. Since we have seen that dropping release and wait commitments might cause different amounts of disruption, it may also be that adding different synchronization commitments might have different disruption. We assign the value \( \delta_x \) to any added wait commitment's disruption, and \( \delta_y \) for an added release commitment. We would expect that the disruption of adding a new synchronization commitment would be at least as great as a totally mismatched commitment, and so expect that its value (\( \delta_x \) or \( \delta_y \)) will be at least \( \delta_x + \delta_y + \delta_x + \delta_y \).

### 3.1.2 Multiple Synchronization Commitments

It is likely that there are multiple commitments in either the old or new solution (or both). In this case, we should match the plan commitments that are most similar to each other to get a true sense of the disruption. Clearly, we want to match identical commitments to each other, since they would not add any disruption to the agent. By matching the remaining pairs in order of minimized distance between them, we have a consistent method for pairing the commitments. When the number of plan commitments is not the same, there will be some left unmatched, and these will contribute to disruption as just described.

We use a greedy algorithm (Figure 3) that calculates the disruption \( (D_{S,j}) \) to an agent \( (A_i) \) caused by switching from one set of commitments \( (S_j') \) to another \( (S_j) \). It first finds identical matches between the sets of commitments (checks for disruption=0), and then, among the unmatched commitments, successively looks for the best remaining match between the sets. Each pair of matched commitments \( (S_f' \) and \( S_f) \) will add the appropriate amount \( (D_{S,jk}) \) where \( k \) is actually dependant on \( j \) to the disruption score (Eq. 5).

\[
D_{x,k} = \delta_x(S_f[1]+S_f[2]) + \delta_x(S_f[3]+S_f[4])
\]

where \( S_f[0] \), the first attribute, is the agent and will be the same for any commitments being compared. Mismatches in any of the other four attributes will contribute the appropriate \( \delta_x \). \( D_{x,k} \) is a measure of disruption to agent \( A_j \) and the brackets indicate the indices of the synchronization constraints in the two sets of commitments.
through C, and A₂ committing to not move through B. From the perspective of A₁, this disrupts its prior expectations of flexibility; however, it could be that A₁ had already decided to move through B anyway, so the disruption is minimal. Either way, we would want to estimate this disruption.

Further, let us say that A₂, for some reason, changes its mind and determines that it really needs to move via B. Now A₁ is perhaps more strongly disrupted in its expectations, needing to definitely adopt the opposite plan than what it had planned to do. We would expect that this would disrupt A₁ more severely, therefore. Finally, though, let us say that A₂ decides to abandon its goal entirely, leaving the world entirely to A₁. A₁ can thus remove its commitment to block the route via B. Of course, it could simply pretend like the commitment was still there, and continue to pursue a plan to move via C, so it need not be strongly disrupted.

Thus, the degree to which changes to blocking commitments will disrupt agents depends on whether these commitments are being added, removed, or switched. To capture these differences, we use δ₈ to model the costs of adding a blocking commitment, and δᵳ for a removed commitment, although from our simple analysis above it seems reasonable to assume that δᵳ=0. If a blocking commitment is removed while an unrelated commitment is added, we would expect the disruption to equal the sum of the disruptions of the two changes to commitments separately. However, if the commitments being added and removed are related, such as switching the blocking of an action with blocking its sibling, then the disruption could be higher than the sum of adding and removing constraints because an agent is forced away from an action it could have planned toward an action that it would not have planned. We use δᵳ for the disruption of a switched blocking commitment between sibling plan steps, and δᵳ for a non-sibling change to blocking commitments.

In general, the disruption of a new commitment relative to a prior commitment can be directly calculated from the two commitments by comparing the plan step attributes:

\[
Db_{ijk} = \delta_{ij}(B_{ij}[1]|B_{i'j}[1]) \& (B_{ij}[2]|B_{i'j}[2]) + \delta_{ij}(B_{ij}[1]|B_{i'j}[1]) \& (B_{ij}[2]|B_{i'j}[2])
\]

where B₁[0], the first attribute, is the agent and must be the same for the two commitments. B₁[1] is the plan step being blocked, and B₁[2] is the parent of the plan step Bᵳ[1], which allows us to identify sibling plan steps.

### 3.2.2 Blocking Commitment Representation

Although blocking commitments, like synchronization commitments, are imposed to successfully coordinate, they differ in that they do not refer to the other agents with whom an interaction is foreseen (in contrast with synchronization commitments, where the agent waiting and the one releasing are explicitly represented). The reason for this difference is fundamental. A blocking commitment is an indication that a child of an or node is removed from the final coordinated plan because a conflict was identified between the conditions of the or node and the conditions of another agent’s plan step. However, there may be more than one agent impacted from the blocking commitment. For example, if A₁ has blocked the route via C (Figure 1) to avoid collision with A₂, then agents that join later can also take advantage of this commitment when planning their routes. Later, even if A₂ changes its plans entirely, A₁ might still need to adhere to the blocking commitment for the sake of other agents. If the commitment explicitly modeled for whom the blocking was being done, then this kind of change would require a change in commitments being modeled. However, since, either way, A₁ is blocking the same plan steps, there should be no disruption from such a change. Hence, this information is not included in the commitment representation.

### 3.2.3 Multiplying Blocking Commitments

Because coordination might require multiple blocking commitments, we need to match them into pairs (to the extent possible) to compute disruption. Each agent (Aᵢ) is subject to a set of (b) blocking commitments Bᵢ = {Bᵢ₁, Bᵢ₂, …, Bᵢδ}. Given two sets of blocking commitments (Bᵢ and Bᵢ’), we can calculate the disruption (Dbᵢ) to an agent (Aᵢ) that switching from one set of commitments (Bᵢ’) to the other (Bᵢ) would cause. The algorithm used is the same as that for synchronization commitments, although it operates on the B sets instead of the S sets and a different evaluation function is used. Similarly to the Dₛᵢ equations (Eq. 6 & 7) we present the following Dᵇᵢ equations, where n is the number of new commitments, and o is the number of old commitments.

When n > o:

\[
Dbᵢ = \sum_{i=1..n} Db(b[i]) + \sum_{o=1..l} \deltaᵢ
\]

When o > n:

\[
Dbᵢ = \sum_{i=1..n} Db(b[i]) + \sum_{o=1..l} \deltaᵢ
\]

Note that calculating the disruption of two sets of blocking commitments is not reflexive, since the two sets may be of different sizes and δᵢ need not equal δᵢ.’

### 3.3 Estimating Overall Disruption

For any agent Aᵢ in the set of agents A = {A₁, A₂, ..., Aᵢ}, the associated sets of synchronization commitments are Sᵢ = {Sᵢ₁, Sᵢ₂, …, Sᵢδ} and blocking commitments are Bᵢ = {Bᵢ₁, Bᵢ₂, …, Bᵢδ}. Sᵢ and Bᵢ could each or both be empty. The disruption to an agent (Dᵢ) is a function of the new commitments (Sᵢ and Bᵢ) for the agent when compared to the old commitments (Sᵢ’ and Bᵢ’) for the same agent. The total disruption to a single agent is simply:

\[
Dᵢ = Dₛᵢ (Sᵢ, Sᵢ’) + Dbᵢ (Bᵢ, Bᵢ’)
\]

Where Dₛᵢ and Dbᵢ have been defined earlier (Eq. 6, 7, 9 and 10) and can be weighted relative to each other by weighting the deltas for synchronization commitments (δₛ, δᵳ, δ₁, δ₂, δ₃, δ₄, δ₅, δ₆) relative to those for blocking commitments (δ₁’, δ₂’, δ₃’, δ₄’, δ₅’, δ₆’). The total disruption (D) of the new coordination solution (C) relative to the old (C’) is just the sum of the disruptions to all the agents involved in the new solution.

\[
D(C, C’) = \sum_{i=1..n} Dₛᵢ (Sᵢ, Sᵢ’) + Dbᵢ (Bᵢ, Bᵢ’)
\]

The disruption calculation should guarantee that the better the commitments match, the lower the disruption will be. However, while we have defined constraints (inequalities) on some of these parameters, we have not suggested numeric values for them. Ultimately, these parameter values summarize an expected disruption cost that will be extremely domain dependent. While we will use some numeric values for our experiments later, our formulation purposely leaves these as user tunable parameters.
4. SORTING CANDIDATE SOLUTIONS

As solutions are generated by the top-down coordination process [1], they will each have an associated disruption value. We create a list of the solutions as they are generated and sort according to the disruption values. Whoever selects the adopted solution then has the ability to easily see the top candidates. We do not assume that disruption will be the only factor involved in selecting a solution, but it could be important.

To verify that our implementation is working properly, we can run the system with the same sets of agents and plans as were used to select the prior (reference) solution. As expected, the same solution always comes out with zero disruption. The only other solutions that have zero disruption are trivially identical solutions (one or more and nodes, which have no other constraints on them, are refined). The solutions with low (but non-zero) disruption are also very similar to the reference, and predominantly solutions where an and node with a constraint on it has been refined (so that the constraint now applies to a child plan step of the node).

Disruption can be used as a measure of quality of the solution, but we do not suggest that it be the only measure of quality used. Other measures of quality could be the length of time the overall solution requires at execution time (the level of concurrency), and the flexibility of the plans (retaining its or branches). We expect that an expert would select a solution that simultaneously minimizes the overall execution time, maximizes the plan flexibility, and minimizes the disruption to the agents. The first two can generally not be satisfied simultaneously, but the choice for the prior solution is typically preserved in the least disruptive subsequent solution. That is, when we start with a reference solution that has the shortest execution time, the solutions with the least disruption are among the shortest duration. Likewise, starting with a reference solution that is more flexible (generally more abstract), the solutions with the least disruption are among the most flexible. Qualitatively, disruption values make sense.

Although the implementation works as expected, doing the top-down hierarchical search for solutions is no faster in finding the least-disruptive solution (which from this point on in this paper we refer to as the “best” solution) than it was the first time around. If the best solution is a more abstract (serialized) solution, it is generated near the start of the search. If the best solution involves constraints between low-level primitives to achieve concurency, it will likely be generated late in the search. If low-disruption solutions are more desirable, and solutions might be selected under time pressure, we would like to generate them earlier in the search - we need to bias the search process towards finding them sooner.

5. IMPROVING SOLUTION GENERATION

Because a coordination solution consists of a set of partially-refined, single-agent hierarchical plans along with a set of coordination commitments, the coordination process must be capable of taking, as input, the plan hierarchies of the agents, and searching for these solutions. The coordination search involves two nested phases, an outer search though plan refinements, and an inner search for inter-agent coordination commitments [1,2]. The outer search iteratively refines abstract plan steps to generate the partially-refined hierarchies. Each of the refinements can be thought of as a cut through (or frontier of) the plans' hierarchies. When an or node is refined, furthermore, there will be multiple successor refinements, each representing the selection of one of the possible refinement plan steps, and thus committing to blocking the other possible refinements.

Given any particular combination of plan refinements, there may still be multiple ways to impose synchronization commitments between the steps to produce a coordinated solution, so a second inner search is performed on each frontier to find these. This search compares plan steps of different agents in all possible combinations to detect possible conflicts, and generates all ordering commitment possibilities that would avoid the detected conflicts. Note that, if the agents' plans are flat (not hierarchical), then there is no outer search process, and the method described here does not speed up the search for good solutions, but is still able to rank solutions according to the amount of disruption.

By default, the search for coordination solutions works top-down through the agents' plan hierarchies, attempting to find ways of resolving conflicts at more abstract levels (which involve fewer, larger plan steps) before seeking resolutions at more detailed levels (which can require substantially more effort to find). It works by initializing the outer search with the frontier containing all agents' most abstract plan steps, passing this into the inner search to generate all feasible solutions at the abstract level, then returning to the outer search to generate the next refinement of the frontier, passing that result to the inner search, and so on. This process generates solutions that are then sorted; compared to the solution generation process, the ranking and sorting process only adds an additional 3% of time and memory.

However, if the goal is to find less disruptive solutions more quickly, it should be possible to work not "top-down" but rather to radiate out from more likely candidates. Specifically, because coordination commitments involve the selection of particular refinements (for blocking undesirable ones) and imposition of synchronization commitments between particular plan steps, then minimally disrupting solutions should involve the same plan steps as the reference solution, to the extent possible. Refinements that have not gone far enough, or have gone too far, or have refined along very different or branches, hold less promise of yielding less disruptive solutions. The search should postpone the expensive inner search process on those frontiers until after more promising candidates are tried.

Rather than automatically passing the outer search nodes into the inner search as they are generated, therefore, the outer search nodes are preliminarily inspected for disruption. Because an outer search specifies a particular frontier of refined plan steps, and because disruption due to blocking only needs this information for each agent, part of the overall disruption measure can be found for the outer search node. A lower bound on the disruption due to synchronization commitments can also be estimated and added in.

Numerous outer search nodes can thus be generated and ranked before any inner search is done, and thus earlier applications of the inner search are more likely to give less disruptive solutions.

---

Notes:

2 For simplicity, we are not modeling more complex blocking constraints. When there are more than two children, blocking constraints considering all possible subsets of children of an or node could result in additional solutions, but those solutions could be reconstructed from solutions already being generated.
Furthermore, if minimal disruption is the only solution criterion, then branch-and-bound can be used to prune the search. A node generated by the outer search whose disruption solely based on blocking equals or exceeds the least disruptive solution found so far need never itself undergo the inner search. However, a child of that node in the outer search space might have a lower value of disruption (if the outer search node had not been refined far enough), so a node that will not undergo inner search should still have its successor (refined) nodes generated and evaluated. In the future, we will study how we can entirely prune nodes even in the outer search by knowing when none of their successors could possibly have lower disruption. This has not been of primary importance since the outer search is much less time-consuming than the (combinatorial) inner search.

5.1 Implementation
The new methods for measuring disruption, and biasing/pruning search based on minimizing disruption, have been implemented in the Multilevel Coordination Agent (MCA) [1,3]. The general outer search in the original code was functionally similar to that reported here, but had hardwired the top-down search and so required some rewriting to accommodate heuristic ordering procedures and pruning mechanisms. The actual functional differences that resulted are only those needed to bias the search towards nodes in the outer search space that have promisingly low levels of disruption, while also allowing the pruning of nodes with high disruption.

Place node on queue; The estimated disruption for an outer search node is used to insert the node into a priority queue. The node score is thus used to bias the search towards refinements that will allow the inner search to find a low disruption solution.

Perform inner search; A search is performed with the current node to find inter-agent ordering commitments that resolve any conflicts remaining between the plans in the node. The solutions will consist of the agents' original plans and a set of commitments for those plans that detail the inter-agent synchronizations added in the inner search, and the blocking commitments (if any) added in the outer search. Each of the solutions will have a disruption score associated with it.

6. SEARCH SPEED IMPROVEMENT
A main challenge we face in recorordination is dealing with the entry of an unexpected agent whose plan conflicts with the already coordinated solution. While it is clear that all sorts of other changes to a system can be disruptive, adding an agent (or multiple agents) is the situation where the computational effort goes up most dramatically in recorordination, and reducing that increase becomes important.

The agent plans we use in this evaluation are identical to our original plan (Figure 2), but are generated automatically with random names for the intermediate points. This models a number of agents all competing for flight corridors in a limited air space. Each agent accomplishes an abstract goal in one of two ways, requiring two distinct resources (the flight corridors) in each case. Other agents will have identically structured plans, where there may be conflicts over the flight corridors. We can vary three parameters in generating our test plans: the number of agents, the number of plan steps conflicting with other agents, and the number of agents that the conflicting plan steps conflict with.

Although we presented extreme levels of conflict between the agents in our initial example (all plan steps conflicting with all other agents), this is unrealistic. Instead, what is more likely to occur is that only a small portion of an agent's plan will conflict with the other agents' plans. To model this more realistic level of overlap between the plans, our first test generates plans where only one plan step for an agent conflicts, but with all other agents. In this first case (the two solid curves in Figure 5), we look at how varying the number of agents affects the overall time required to coordinate the plans using biased vs. unbiased search. The time to find all solutions in the unbiased search increases exponentially with the number of agents. Biased search prunes parts of the search space and, although it also increases exponentially, the slope on the log scale is slightly less, 1.2 instead of 1.5. When searching for the least disruptive solution in unbiased search, there is no way to be sure it is the least disruptive without an exhaustive search. With a biased search, the least-disruption solution is verified faster because the number of possible solutions generated, and the total search time, are reduced.

In all cases, the least disruptive solution was found approximately halfway through the entire unbiased search process, while it was found one-fifth of the way through the biased search process. Because the search must complete before we can guarantee that the least disruptive solution has been found, and because the speedup is only linear, detailed results are not presented here.

5.1 Implementation
The new methods for measuring disruption, and biasing/pruning search based on minimizing disruption, have been implemented in the Multilevel Coordination Agent (MCA) [1,3]. The general outer search in the original code was functionally similar to that reported here, but had hardwired the top-down search and so required some rewriting to accommodate heuristic ordering procedures and pruning mechanisms. The actual functional differences that resulted are only those needed to bias the search towards nodes in the outer search space that have promisingly low levels of disruption, while also allowing the pruning of nodes with high disruption.

Place node on queue; The estimated disruption for an outer search node is used to insert the node into a priority queue. The node score is thus used to bias the search towards refinements that will allow the inner search to find a low disruption solution.

Perform inner search; A search is performed with the current node to find inter-agent ordering commitments that resolve any conflicts remaining between the plans in the node. The solutions will consist of the agents' original plans and a set of commitments for those plans that detail the inter-agent synchronizations added in the inner search, and the blocking commitments (if any) added in the outer search. Each of the solutions will have a disruption score associated with it.

6. SEARCH SPEED IMPROVEMENT
A main challenge we face in recorordination is dealing with the entry of an unexpected agent whose plan conflicts with the already coordinated solution. While it is clear that all sorts of other changes to a system can be disruptive, adding an agent (or multiple agents) is the situation where the computational effort goes up most dramatically in recorordination, and reducing that increase becomes important.

The agent plans we use in this evaluation are identical to our original plan (Figure 2), but are generated automatically with random names for the intermediate points. This models a number of agents all competing for flight corridors in a limited air space. Each agent accomplishes an abstract goal in one of two ways, requiring two distinct resources (the flight corridors) in each case. Other agents will have identically structured plans, where there may be conflicts over the flight corridors. We can vary three parameters in generating our test plans: the number of agents, the number of plan steps conflicting with other agents, and the number of agents that the conflicting plan steps conflict with.

Although we presented extreme levels of conflict between the agents in our initial example (all plan steps conflicting with all other agents), this is unrealistic. Instead, what is more likely to occur is that only a small portion of an agent's plan will conflict with the other agents' plans. To model this more realistic level of overlap between the plans, our first test generates plans where only one plan step for an agent conflicts, but with all other agents. In this first case (the two solid curves in Figure 5), we look at how varying the number of agents affects the overall time required to coordinate the plans using biased vs. unbiased search. The time to find all solutions in the unbiased search increases exponentially with the number of agents. Biased search prunes parts of the search space and, although it also increases exponentially, the slope on the log scale is slightly less, 1.2 instead of 1.5. When searching for the least disruptive solution in unbiased search, there is no way to be sure it is the least disruptive without an exhaustive search. With a biased search, the least-disruption solution is verified faster because the number of possible solutions generated, and the total search time, are reduced.

In all cases, the least disruptive solution was found approximately halfway through the entire unbiased search process, while it was found one-fifth of the way through the biased search process. Because the search must complete before we can guarantee that the least disruptive solution has been found, and because the speedup is only linear, detailed results are not presented here.
In the second case (the two dashed curves in Figure 5), the agents are subjected to fewer conflicts; the conflicting plan step only conflicts with one or two other agents. The points do not appear to follow as straight a line because of even/odd numbers of agents. With an even number of agents, we can limit the conflicts between agents to a single other agent. With an odd number, the extra agent must conflict with two other agents, also increasing their conflicts, and making the coordination problem more difficult. Again, the biased search is an order of magnitude faster than the unbiased search, and the slope is 0.8 instead of 1.0. In general, we can prune away enough nodes using bias to finish the search for solutions for \( N \) agents in much the same time as the unbiased search does for \( N-1 \) agents. With larger numbers of agents, the relative timesavings gets even greater.

7. Conclusions And Future Work

In this paper, we described a method for measuring disruption to agents facing recordination. We have shown how this flexible measure can be used to bias and bound the search process to find suitable coordination solutions faster.

Although our research focused mainly on how using disruption measures can speed up the recordination problem, we have seen that there are instances where it could be used to speed up the initial coordination problem. For instance, coordinating \( N-1 \) agents and then coordinating \( N \) with bias can be an order of magnitude faster than coordinating the \( N \) agents without bias (when \( N>3 \)). We are interested in applying disruption measures to a wider range of problems, and intend to evaluate using disruption to coordinate larger numbers of agents by recordinating groups of locally coordinated agents.

Although our work so far has only included synchronization and blocking commitments in measuring disruption, we intend to add reasoning about temporal constraints to the disruption measure. When agents have synchronized their plans against clock times, changing those time constraints could also be disruptive to the agents and their other commitments. When appropriate, the added measures of disruption could make the biased search even faster.

Our current work on disruption is closely tied to HTN planning, and the kinds of commitments used in our metrics may not be applicable to all planning methods. In a wider context, applying the techniques described here to other plan coordination mechanisms should be possible.

8. Acknowledgments

This work benefited from discussions with Jeff Cox and Brad Clement and the code that they have written. DARPA (F30602-98-2-0142) supported this research.

9. References


[13] Control of Agent Based Systems Project
http://coabs.globalinfotek.com