Towards Flexible Teamwork in Persistent Teams

Milind Tambe and Weixiong Zhang
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{tambe,zhang}@isi.edu

Abstract

Teamwork is a critical capability in multi-agent environments. Many such environments mandate that the agents and agent-teams must be persistent i.e., exist over long periods of time. Agents in such persistent teams are bound together by their long-term common interests and goals.

This paper focuses on flexible teamwork in such persistent teams. Unfortunately, while previous work has investigated flexible teamwork, persistent teams remain unexplored. For flexible tamwork, one promising approach that has emerged is model-based, i.e., providing agents with general models of teamwork that explicitly specify their commitments in teamwork. Such models enable agents to autonomously reason about coordination. Unfortunately, for persistent teams, such models may lead to coordination and communication actions that while locally optimal, are highly problematic for the team's long-term goals. We present a decision-theoretic technique to enable persistent teams to overcome such limitations of the model-based approach. In particular, agents reason about expected team utilities of future team states that are projected to result from actions recommended by the teamwork model, as well as lower-cost (or higher-cost) variations on these actions. To accomodate real-time constraints, this reasoning is done in an any-time fashion. Implemented examples from an analytic search tree and some real-world domains are presented.

1. Introduction

Teamwork is critical in many multi-agent environments, such as, interactive simulations for training and education[17], RoboCup robotic and synthetic soccer[10], interactive entertainment[5], multi-robot deep sea or space exploration or reconnaissance, and internet-based information integration. A key requirement in many of these domains is team *persistence* or long-term existence. For instance, consider virtual environments for training[17]. Here, the Advanced Concepts Technology Demonstration battlefield simulation exercise (henceforth, referred to as ACTD) jointly conducted in the US and Europe in October 1997, lasted for multiple days. Participating teams of synthetic pilots were required to persist for at least a single mission execution, which lasted several hours. Ideally, a pilot team should have persisted through not one, but multiple such missions, without requiring a human-in-the-loop to debug behaviors between missions. Teams of robotic vehicles for deep sea or space exploration or reconnaissance have similar requirements for persistence. The RoboCup soccer tournament also requires player teams (robotic or synthetic) to persist for at least a full game[10]. While such persistence is a matter of degree (team longevity in different domains may vary), a persistent team contrasts with a team working together to accomplish a specific temporary joint goal, e.g., the interaction of personal software agents to set up a meeting among their users[6].

Persistent teams bring forth a range of research issues ranging from robust team performance over extended time periods (without requiring a human-in-the-loop) to more abstract issues such as maintenance of a team identity in the face of changing membership. We hypothesize that the key challenges in persistent teams arise from *reflective persistence*: a team's explicit reflection and reasoning about its persistence, in service of flexible teamwork. In particular, a team must reflect upon both its past persistence (e.g., for learning from experience) and future persistence (e.g., for appropriate resource allocation). This paper will focus on the issues of reflective future persistence for improved resource allocation. That is, teamwork actions in a persistent team, including coordination, communication or task-allocation actions, must be driven by the

team's long-term common interests and goals. Thus, a persistent team must not exhaust all its resources in coordinating for its current joint goal, if those resources are better preserved for the team's longer-term goals. Analogously, however, a persistent team may need to expend more than the necessary resources on its current joint activity, to better serve the team's longer term goals. One example of the latter phenomenon is team reorganization in anticipation of future tasks, as seen in Section 3.

Unfortunately, while previous work has recognized the importance of persistence in individual agents[3], it has so far failed to explore the issues in persistent teams. Nonetheless, foundational issues in flexible teamwork in general are being investigated. One promising approach that has emerged focuses on providing agents with explicit models of teamwork[15, 8, 14]. These models are based on previous theories of teamwork[12, 9, 4]. They enable agents to autonomously reason about coordination and communication in teamwork, providing improved flexibility. Such reasoning is driven by the model's explicit specification of team members' ideal behaviors in teamwork. However, teamwork theories and models are not explicitly motivated by persistence. Thus, as illustrated in Section 3, they may specify coordination actions that while locally optimal, consume enough of team resources to jeopardize the team's longer term goals.

This paper takes some initial steps to enable persistent teams to overcome such limitations in teamwork models. In particular, in the complex, dynamic domains of interest, it is not possible to optimally plan all coordination activities in advance. Therefore, team members dynamically reflect upon future persistence when coordinating. Essentially, members compute the long-term expected utility of the future impact of the coordination action suggested by a teamwork model. They also compute such utilities for variations of the suggested action, that tradeoff teamwork quality for resource consumption. Team members then select coordination or communication actions that maximize long-term expected team utility. Given dynamic domains, team members dynamically change their actions with new information. Of course, one key challenge here is efficient operationalization of the above idea. To this end, we propose a state-space formalism to model possible future team states and actions. A complete search of this state space is however impractical, since (i) a persistent team implies a search extending to all possible future team states; and (ii) hostile situations such as battlefields prohibit agents from prolonged deliberation with no action. Therefore, an *any-time*[1] search method is employed, based on bounded-lookahead search.

The paper demonstrates the above approach in three domains, and analyzes situations involving persistent teams where this approach will dominate a pure teamwork-model-driven approach. While a general analysis is provided, the paper focuses in particular on the STEAM model of teamwork[15]. STEAM is chosen since it is a state-of-the-art teamwork model, that has been successfully deployed in several real-world domains. For instance, pilot teams based on STEAM successfully participated in the ACTD exercise mentioned above[15]. STEAM-based soccer-players participated in the RoboCup'97 tournament, winning the third place prize in over 30 teams[16]. Nonetheless, our investigation has broader applicability. STEAM itself is based on the joint intentions theory[12], and is also influenced by the SharedPlans theory[4]. Thus, lessons learned here are applicable to other teamwork models based on such theories, such as the joint responsibility model[8] which is also based on joint intentions.

2. Background: STEAM

STEAM is an implemented model of teamwork, aimed at enabling development of individual agents that can engage in flexible teamwork [15]. STEAM uses joint intentions as its basic building block and it builds up a hierarchy of joint intentions corresponding to a team's goal or plan hierarchy. STEAM facilitates flexible teamwork via two classes of domain-independent actions. The first class of *coherence preserving* or *CP* actions is based on the commitments in the joint intentions theory, and aims to maintain coherence in the team. It includes the execution of the *establish-commitments* protocol[15] to establish a joint intention. Thus, given an arbitrary team plan OP, all team members execute the protocol to simultaneously select OP as their joint intention. *CP* actions also include an agent's commitment to establishing mutual belief in the team, when it privately discovers their joint intention to have been terminated (i.e., either achieved or unachievable or irrelevant). By requiring agents to jointly begin and terminate joint intentions, STEAM ensure full coherence within a team. STEAM uses a decision-theoretic approach to select the most cost-effective method of executing *CP* actions, e.g., it may rely on visual contact rather than explicit communication.

The second class of domain-independent actions in STEAM is *maintenance and repair* or *MR* actions. One key aspect of *MR* actions is an explicit specification of the dependency relationship of the joint intention on individual team members' activities. STEAM uses such

dependency specifications to infer the status of a joint intention, based on the status of individual members' activities. For instance, if a key individual team member fails to perform its activities, the joint intention itself may be unachievable. In such a case, STEAM's *MR* actions suggest an appropriate reorganization.

STEAM is currently implemented in Soar[17] as a set of about 280 production rules. For a concrete example, consider the operator (reactive-plan) hierarchy shown in Figure 1 for synthetic helicopter pilots developed using STEAM. This operator hierarchy is similar to reactive-plan hierarchies in architectures such as RAPS[2]. One key novelty however is *team operators* (reactive team plans), which explicitly express a team's joint activities, unlike the regular "individual operators" which express an agent's own activities. Thus, operators shown in [] such as [Engage] are team operators (others are individual operators). At any point in time, one path through this operator hierarchy is active, which is an agent's currently active joint intentions (corresponding to team operators), and intentions (corresponding to individual operators). STEAM's CP and MR actions are based on these joint intentions. As an example of an MR action, consider the team operator [travelling] in Figure 1, where one team member flies as the flight leader, while others follow the leader. All other members' activities are thus dependent on the flight leader. If the flight leader crashes [travelling] is inferred to be unachievable. Reorganization now occurs, as some other team member takes over the activities of the flight leader.

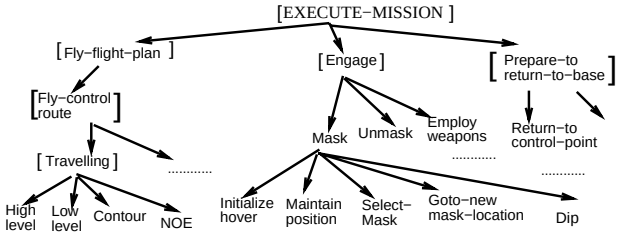


Figure 1. Attack domain: Portion of team operator (reactive-plan) hierarchy.

3. Implications for Persistent Teams

STEAM's *CP* and *MR* actions are focused on optimally performing its currently active joint intentions. In fact, STEAM uses decision-theoretic techniques to optimize execution costs of *CP* actions. Unfortunately for a persistent team, STEAM does not reason about the longer-term impact of its suggested *CP* and *MR* actions. For instance, helicopter may need to fly to the location of other helicopters to optimally communicate (it avoids breaking radio silence), but from a long-term perspective it may consume both precious time and fuel. Furthermore, STEAM does not consider the possibility of achieving partial coherence given the resource cost of full coherence.

Thus, a key issue for a persistent team is that the resources allocated (or not allocated) in *CP* or *MR* actions may be highly detrimental for its longer-term joint goals. The following examples are illustrative of such problems.

1. In the ACTD simulation (see Section 1), when a helicopter-pilot team reaches its simulated battlefield, it typically establishes a joint intention to plan attacking positions. Once such positions are planned, helicopters fly to those positions. At this point, agents establish a joint intention to engage the enemy. In one simulation run, when the helicopters reached the battlefield, the enemy began advancing towards them. Unfortunately, the *CP* action to establish a joint intention to plan attacking positions took a significant amount of time — some helicopters were not ready. Thus, before the agents could ready themselves to engage the enemy, the enemy was close enough to shoot down some of the helicopters.
2. In the RoboCup soccer simulation domain, three or more out of 11 soccer players act as defenders, to defend their goal from the opponents. Initially, the defenders establish a joint intention to look out for an attack by the opponents. The players have very limited vision, and thus, not all can simultaneously see an attack. If any one defender spots an attack, it must inform others that their current joint intention is achieved (*CP* action), so they can all jointly block the attack. However, since defenders can be positioned far apart, and since a player's shouting has limited range, significant time would be consumed if a player was to move to inform others. Meanwhile, the attacking player can bypass the defenders, thus defeating the defender's next joint goal of blocking the attackers.
3. In the ACTD simulation, the helicopter team is typically divided into two subteams. One *scout* subteam, consisting of two helicopters, is first sent forward to scout the battle position, while the second *attack* subteam remains hidden from the enemy. Upon completion of scouting, the two subteams together attack the enemy. In one run, one helicopter in the scout subteam crashed. Since this did *not* cause the relevant joint intention to be unachievable (one scout helicopter was still flying), no *MR* action was executed to reorganize the subteams. However, human experts suggested reorganization. In particular, given a threat to the remaining scout helicopter in the battle position, they suggested that one helicopter from the attack subteam should join the scout subteam.

In the first two cases, a key problem is that STEAM insists on pursuing *CP* actions to reach a coherent team state, even though these actions consume a significant amount of time (resource). Furthermore, STEAM does not reason about the long-term impact of the resources consumed by these *CP* actions. Indeed, this resource consumption is highly problematic for the team's longer-term goals. In the third example, STEAM does not execute an *MR* action, since the current joint intention is still achievable, and, once again, STEAM does not reason about the long-term impact of this decision — that without an *MR* action in its present state, there is a threat to its team member in the future.

4 Reasoning about Persistence

To overcome the limitations of STEAM, we have designed and implemented an approach called STEAM-L (short for STEAM with lookahead). STEAM-L uses a decision-theoretic approach, enabling a persistent team to reflect upon its future, and maximize long-term expected team utility. With STEAM-L, team members compute the expected utility of the future impact of the coordination action suggested by the teamwork model (STEAM). STEAM-L also computes expected utility of variants of the suggested coordination action, where the variants may tradeoff team coherence for resource consumption. Thus, for instance, variants of a *CP* action may inform only a certain percentage of team members rather than all of the team. While this variant may lead to a lower-quality team state in the short term (due to reduced coherence), resources saved lead the team to a better state in the long term. Such a *CP* variant is an illustration of STEAM-L's introduction of flexibility in the commitments in teamwork. Thus, while STEAM's decision-theoretic reasoning does not change the basic commitments of individual joint actions — it only minimizes costs for embedded *CP* actions — STEAM-L's reasoning can change the nature of these commitments.

4.1 State-space representation

For efficient operationalization of the lookahead reasoning in STEAM-L, we have cast it as state-space search over team's future states. There are two types of states (nodes) that need to be modeled. The first type, called action nodes, are where agents can apply actions. The second type are outcome nodes, where agents wait for the outcome of their actions. Figure 2 illustrates a simple state space, where square and circle nodes represent action and outcome nodes, respectively. Each action node may have multiple possible actions. Since agents may expend resources when taking actions, there is a resource cost associated with an edge outgoing from an action node. The number on an outgoing edge from an outcome node is the probability that an outcome occurs. In particular, due to uncertainty in the real-world, agents can only estimate the likelihood of particular outcomes of their actions. As seen in the figure, action nodes are followed by outcome nodes, and vice versa. Associated with the leaf node in the tree is the final situation cost. The resulting search tree is essentially a decision tree[13].

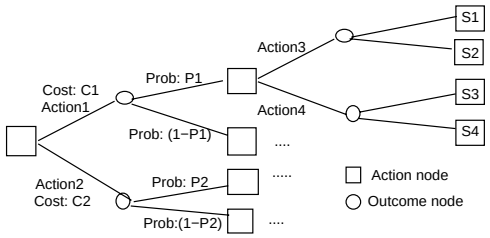


Figure 2. Decision tree search.

The actions in this search are of two types. The first type are all teamwork-related actions suggested by STEAM, specifically *CP* actions and *MR* actions. In this paper, STEAM-L focuses on agents' decisions with respect to *CP* and *MR* actions, and it explores their variations in its lookahead search. The second type of actions are domain-specific actions, such as the pilots' operator hierarchy as shown in Figure 1. In principle, variations of these actions could also be explored in the lookahead search, but that will be outside the scope of this paper. Furthermore, given the hierarchy of domain-specific operators provided, STEAM-L does not simulate the effect of every low-level action. Instead, it searches over abstract high-level operators.

A state in this search space is an agent's model of the team's overall state. It includes both the team's mutual belief and the private beliefs of team members. However, not all private beliefs of other members need be modeled, but only those relevant to initiation and termination of the relevant joint intentions.

4.2 Any-time lookahead search

In the search tree in Figure 2, an agent selects among the multiple possible actions at the action node by computing expected utilities of different actions, and selecting an action that maximizes expected long-term team utility. Even though the number of team's future possible states is generally huge, agents must compute expected utilities efficiently in an anytime fash-

ion, and have a decision on next action ready all the time to respond to unexpected events. To this end, STEAM-L carries out an iterative-deepening lookahead search [11]. STEAM-L is always activated at an action state where a decision on the next *CP* or *MR* action needs to be made. The algorithm runs in iterations with each subsequent iteration searching to a deeper depth than the previous iteration. The first iteration runs to depth one or a depth that is determined by an estimate on the total time available for lookahead search. Each iteration of the algorithm uses best-first or depth-first search. When the search reaches the frontier, it applies a static node evaluation to compute the node cost $f(n)$ of node n. The costs of the frontier nodes are then backed up to the root. For internal node n, its node cost is backed up from the costs of its children $n_1, n_2, \cdots, n_b$ as follows:

$$f(n) = \begin{cases} min\{f(n_1), f(n_2), \cdots, f(n_b)\}, \\ \text{if } n \text{ is an action node;} \\ \sum_{i=1}^{b} p_i f(n_i), \\ \text{if } n \text{ is an outcome node.} \end{cases} \quad (1)$$

The algorithm terminates whenever it is required to provide a decision on the next action, which is a move from the current action state toward the best action state on the search frontier of the most recent iteration.

The static node evaluation or static node cost $f(n)$ of a node n is a function of the node's resource cost $r(n)$ and situation cost $s(n)$, i.e., $f(n) = F(r(n), s(n))$, where $F(\cdot)$ is a function. STEAM-L computes expected utility of future states, where expected utility is computed as the minimum expected backup cost, using the backup rules of (1).

While both STEAM and STEAM-L use joint-intentions based teamwork model, they have two main differences. First, STEAM-L conducts a lookahead search, while STEAM's response is compiled in based on an examination of the direct outcomes from an action. The significant advantage of looking ahead is that it can discover a trap within the lookahead horizon, so that agents can avoid a path leading to the trap. Indeed in a scenario where a disastrous situation is just a few steps away from a favorable action, the original STEAM can fail. One such case is the helicopter-pilot team joint attack example discussed in Section 3. Second, given the resource costs of STEAM's teamwork actions, STEAM-L explores lower-cost (but locally suboptimal) variations of those actions. In other words, STEAM ignores the cost of spending resources which may cause problems in the long term.

5. Experimental Results

The main goal of our experimental study is to investigate the effectiveness of STEAM-L's lookahead search for team persistence with resources being taken into account. To this end, we compare STEAM-L and STEAM head to head on three domains, an artificial search tree model as well as two real domains: helicopter teams and soccer-player teams. On the search tree model, we examine the probabilities that STEAM-L wins against STEAM, in terms of the total amount of resources both algorithms consume and the quality of the final states they can reach. On the real domains, we compare the performance of the algorithms on examples such as those in Section 3.

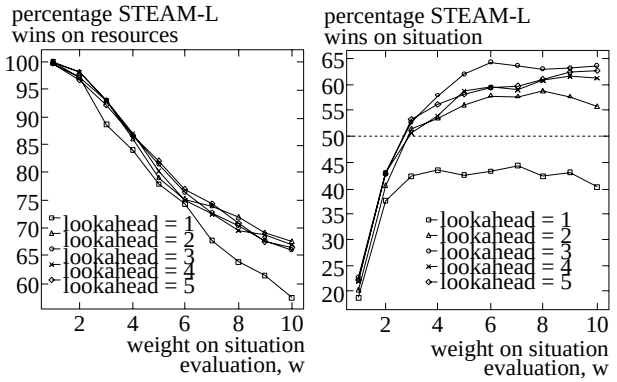
Another purpose of our experimental study is to examine how resource cost and situation cost will effect STEAM-L's performance. In this study, we use a linear combination of these two costs to compute node cost. Specifically, the node cost $f(n)$ of a frontier node n is computed as $f(n) = r(n) + w \cdot s(n)$, where $r(n)$ and $s(n)$ are the resource cost and situation cost of n, and w is a weight on situation cost, a parameter of STEAM-L which can be tuned for a good performance.

5.1. Artificial search tree

The motivations to use an artificial search tree are that it is easy to generate and reproduce so that results can be easily verified; it is easy to manipulate so that different features of state space can be examined closely; and most importantly results from the domain help us to better understand the phenomena in real domains. The artificial search tree is a variation of an incremental random tree [18].

One important property of situation costs in real world problems is their dependence. Two situation costs have some degree of dependence if their corresponding action nodes share common edges on the paths from the starting node to them. This is because the actions corresponding to the common edges have the same effects on the action nodes in the future. The degree of dependence depends on the number of edges these action nodes share in common. We use the following method to introduce dependence in the situation costs in our artificial search tree. We first assign random numbers, drawn from the same distribution, to action nodes. We then compute the situation cost on an action node as the sum of its random number and the situation cost on its direct action-node parent.

In our experiments, edge (resource) costs are random numbers uniformly chosen from a set of $\{1/M, 2/M, \cdots, 1 - 1/M\}$, and random numbers on



(a) Total resource cost. **(b) Final situation evaluation.**

Figure 3. STEAM-L vs. STEAM on trees with branching factor 3 & depth 40.

action nodes are similarly chosen from another set of $\{1/M, 2/M, \cdots, 1 - 1/M\}$, where $M = 2^{32} - 1$. Figure 3 shows the results on random tree with uniform branching factor 3 and depth 40. Notice that a tree with depth 40 is equivalent to a task that requires 20 actions, where the other 20 levels of nodes are outcome nodes. The results are averaged over 500 instances.

Figure 3(a) shows that by taking resource costs into account and looking deeper, STEAM-L spends less total resources than STEAM with a large percentage (y-axis). Figure 3(a) also shows that when the weight on situation cost, w, increases, the percentage that STEAM-L wins drops. This is because although with a large w, STEAM-L tends to take an action with a good (or small) situation cost, the final state may carry a large resource cost. Figure 3(b) shows that when w is small, increasing w can significantly increase the possibility that STEAM-L wins in terms of final situation cost. However, when w is large enough, larger than 6 in Figure 3(b), increasing w does not help STEAM-L significantly. Figure 3 indicates that there is a tradeoff between total resource costs and final situation cost. A strategy that uses more resources tends to reach a final state with a high quality. Figure 3 also means that by carefully adjust the parameter w, STEAM-L can outperform STEAM in both resource costs and situation cost if STEAM-L searches deep enough.

5.2. Applications

STEAM-L has been successfully applied to key examples where STEAM earlier faced problems in the helicopter-pilot and soccer player teams. STEAM-L was provided the expected plan (i.e., the expected future sequence of team operators), estimates of resource costs for these team operators and related teamwork (CP and MR) actions, likelihoods of expected outcomes, and situation costs of expected outcomes. An outcome involving a loss of coherence was rated poorer than one without such a loss. Outcomes where a goal was scored (in RoboCup) and where enemy would engage the helicopters from close range (in the helicopter domain) were rated very poorly. The following discusses results of STEAM-L's application to the illustrative problems from Section 3.

1. STEAM-L does a five-step lookahead, to recognize the problem in full synchronization (establishing the first joint intention), and instead selects partial synchronization in the helicopter team in limited time. Basically, upon seeing the enemy, the helicopter-team estimates the total time available before it can engage the enemy. STEAM-L's five-step lookahead search indicates that full synchronization for the two joint intentions established, as suggested by STEAM, consumes significant amount of time (resource cost), and leads to sufficiently high likelihood of actual loss of some helicopters to enemy fire (very high situation cost). Partial synchronization has lower cost, since it consumes lower amount of time (resource cost) and leads only to a possible loss of coherence of one-two helicopters (which may not participate in further team activities).
2. STEAM-L's four-step lookahead search shows that if a defender who spots the attack just shouts once to inform others, without moving towards others, that would maximize expected utility. Essentially, by shouting once, this defender makes it highly likely that at least one other defender will hear it (but not necessarily all defenders). This increases the likelihood of blocking the attack (given at least two defenders); not shouting and acting alone would decrease that likelihood. Yet, shouting, unlike moving does not consume the key team resource of time (shouting costs milliseconds, while moving costs seconds).
3. STEAM-L's three step lookahead enables a helicopter from the *attack* subteam to join the *scout* subteam, since lookahead search discovers the threat to the single helicopter in the scout subteam. One key assumption made here is that initial allocation of tasks to subteams is assumed to be appropriate. Thus, any significant change in the number of members in a team leads agents to reason about possible member reorganization (rather than waiting until the relevant joint intention is unachievable).

STEAM-L is not yet in a mature state, unlike STEAM. In particular, STEAM-based helicopter teams have been fielded in several large-scale synthetic exercises mentioned in Section 1. While in general this participation has been successful, key problems have emerged. Teamwork flexibility provided by STEAM-L in the helicopter teams can address these problems.

With respect to RoboCup, we now realize that the strategy followed by the players fielded in RoboCup'97 was identical to the outcome from STEAM-L. The success of the team provides a limited validation to STEAM-L's reasoning. Conversely, in RoboCup'97 this strategy was pre-compiled, and STEAM-L's reasoning now validates this strategy. Learning from STEAM-L's reasoning to pre-compile a response appears appropriate in situations involving significant time pressure, such as RoboCup, and remains an issue for future work.

5.3. Analysis of Results

In real domains, in key problematic examples, such as those in Section 3, STEAM-L takes actions that are better than those recommended by STEAM. These results match those on random trees in Figure 3. Yet, in a majority of the cases, in our real domains, STEAM-L selects the same *CP* or *MR* action as one recommended by STEAM. This is a surprising discrepancy: STEAM-L almost always differs from STEAM in random trees, but not as much in the real-world domains (although it does differ as discussed in Section 5.2).

The reason is an important regularity in the expected utility of team states in real-world domains. Consider first situations with insignificant resource (time) constraints, e.g., a helicopter team when there is no enemy nearby. Here, a *CP* action can lead a team to a coherent state. *STEAM-L's lookahead search on such a coherent state does not degrade its expected utility.* Informally, there are no hidden negatives that STEAM-L discovers in the future of this coherent state. A non-*CP* action, however, will lead the team to a non-coherent state, which has a lower expected utility. STEAM-L's lookahead search on such a state leads to further degradation in the expected utility, as STEAM-L may discover further negatives of non-coherence in its lookahead. Thus, STEAM-L's lookahead only confirms the appropriateness of STEAM's selection of a *CP* action. Such regularity in the expected utility is absent in random trees.

In situations with significant time constraints (e.g., when enemy vehicles move close to a helicopter team), another regularity is observed, so that STEAM-L's choice differs significantly from STEAM. Here, while a *CP* action leads to a coherent state, it takes up too much of the available time. STEAM-L's lookahead now discovers significant negatives or traps in the future of this coherent state, e.g., a future state is seen in which some team pilots are possibly shot down. Thus, lookahead leads to a drastic reduction in the expected utility of coherent states under resource constraints. Such situations are a trap for STEAM, as it does not look beyond the next action. Thus, STEAM-L chooses in such cases to select a variation of a *CP* action, while STEAM rigidly follows a *CP* action.

Based on the above analysis, it appears STEAM-L becomes more effective in state space where resources are scarce and situation costs vary dramatically. Thus, it would appear beneficial to invoke STEAM-L's lookahead selectively, only in such cases.

6. Summary, Discussion, Future Work

Recent progress in theories of teamwork[4, 12, 9], as well as implemented models of teamwork inspired by such theories[8, 14, 15], have led to an improved understanding of teamwork, and to complex teamwork applications. We have begun to understand that teamwork is distinct from pure coordination, and the importance of commitments in teamwork. In our own previous work, we have applied STEAM[15], a state-of-the-art teamwork model, in several real-world, synthetic domains. This paper takes a step beyond the state-of-the-art, by investigating an important novel phenomenon in teamwork, that of persistent teams. Although persistent teams remain unexplored, they are an important requirement in many multi-agent applications. In addition to identifying persistent teams, contributions of this paper include the presentation of: (i) key issue of resource allocation for coordination in teamwork, given long-term goals of persistent teams; (ii) a decision-theoretic formalization of this resource allocation problem; (iii) STEAM-L as a specific decision-theoretic approach, based on augmentation of STEAM with anytime lookahead search; (iv) application and analysis of STEAM-L in several domains. STEAM-L leads to improved flexibility in the commitments in teamwork, motivated by long-term team interests. Given that STEAM is rooted in the joint intentions theory[12], and also borrows from SharedPlans[4], the results bear upon other teamwork models based on such theories.

While persistent agents and teams remain uninvestigated in general, one exception is Horvitz's work on continual computation[7]. He provides theoretical models for how persistent agents should expend idle time, particularly for solving future problems, given probability distributions of expected problems and their expected costs. He also models the issue of expending a fraction of the current problem-solving time for future problems. STEAM-L complements Horvitz's exploration in two ways. First, STEAM-L focuses on coordination and communication in agent teams rather than individual behaviors. Second, Horvitz's work con-

cerns allocating computational resources to a set of immediate next problems, while our work focuses on selecting the next team action using information collected from a search into future team states.

One major challenge for future work is formalizing commitments in the context of persistent teams. In particular, with STEAM-L, team members may not necessarily fulfill their commitments (for instance, to inform others), if doing so harms the longer-term interests of the team. Such flexibility in teamwork commitments is important in uncertain environments. To formalize such flexible commitments, a reviewer for this paper suggests adding to the "relativizing clause" in the joint intentions framework[12]. However, this topic requires further investigation.

Another major challenge is understanding the interaction between agent persistence and team persistence. Here it is useful to consider a categorization of teams along two dimensions: persistence of teams and persistence of members. We can thus consider at least four team types. First, in a persistent team consisting of persistent members (PTPM) there is no change in team membership. Second, in a persistent team consisting of non-persistent members (PTNM), team membership may change over time. Third, in a NTPM, agents temporarily form a team for a specific objective. An NTNM is a non-persistent team with non-persistent members. This paper has focused on PTPM, but discussed one issue (reorganization) of PTNM. Analyzing these different team types is an interesting area of further research.

7. Acknowledgements

The research was supported in part by NSF grant IRI-9711665, in part by NSF grant IRI-9619554, and in part by contract N66001-95-C-6013 from ARPA/ISO. We thank Randy Hill, Jon Gratch and Paul Rosenbloom for discussion of issues related to the ACTD demonstration, and the RoboCup simulation group for discussion of issues related to RoboCup.

References

[1] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 979–984. IJCAI-89, Morgan Kaufmann, August 1989.

[2] J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1987.

[3] S. Franklin and A. Graesser. Is this an agent or just a program? a taxonomy for autonomous agents. In *Proceedings of the third international workshop on agents, theories, architectures, and languages*. Springer-Verlag, New York, 1996.

[4] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.

[5] B. Hayes-Roth, L. Brownston, and R. V. Gen. Multiagent collaobration in directed improvisation. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[6] T. Haynes, S. Sen, N. Arora, and R. Nadella. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)*, 1997.

[7] E. Horvitz. Models of continual computations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1997.

[8] N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75, 1995.

[9] D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhard, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems, Lecture notes in AI 830*. Springer, NY, 1992.

[10] H. Kitano, M. Tambe, P. Stone, S. Coradesci, H. Matsubara, M. Veloso, I. Noda, E. Osawa, and M. Asada. The robocup synthetic agents' challenge. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, August 1997.

[11] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[12] H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press, 1990.

[13] H. Raiffa. *Decision analysis*. Addison Wesley, Reading, MA, 1968.

[14] C. Rich and C. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)*, 1997.

[15] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124, 1997.

[16] M. Tambe, J. Adibi, Y. Alonaizon, A. Erdem, G. Kaminka, S. Marsella, I. Muslea, and M. Tallis. ISIS: Using an explicit model of teamwork in Robocup97. In *RoboCup-97: The first robot world cup soccer games and conferences*. Springer-Verlag, Heidelberg, Germany, 1998.

[17] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

[18] W. Zhang and R. E. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79:241–292, 1995.