Towards Flexible Teamwork in Persistent Teams: Extended Report

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 teamwork, 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 based on Markov decision processes 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.

Keywords: Multi-agent systems, Teamwork, Persistence, Markov decision processes

1. Introduction

Teamwork is critical in many multi-agent environments, such as, interactive simulations for training and education[34], RoboCup robotic and synthetic soccer[20], interactive entertainment[13], multi-robot deep sea or space exploration or reconnaissance, and internet-based information integration. An increasingly important requirement in many of these domains is that of *persistent teams*, i.e., teams that persist over long periods of time. For instance, consider virtual environments for training[34]. 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. The RoboCup soccer tournament also requires player teams (robotic or synthetic) to persist for at least a full game[20]. Teams of robotic vehicles for deep sea or space exploration or reconnaissance have similar requirements for persistence. For instance, future robotic missions to Mars will require teams of robots to persist for multiple years, continually working together, to accomplish long-term mission goals. Indeed, we expect the need for team persistence to continually grow, as increasingly complex large-scale applications make periodic human intervention extremely difficult, costly and tedious.

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[14] or cooperative negotiations between two software-agents aboard (future-generation) aircraft to avoid a collision[16]. Indeed, truly persistent teams of the future would require us to address a range of novel issues for teams. Clearly, one key issue is robust team performance over extended time periods (without requiring a human-in-the-loop). Another key issue is a team's organizational adaptation with experience in a given environment, for instance, by changing tasks assigned to different individuals or subteams[5, 36]. Such an organizational change would not be meaningful in a non-persistent team, since it may not persist to accumulate experience as a team. Persistent teams also involve more abstract issues, such as establishing a team identity, which encompasses certain "organizational culture" and/or standard-operating-procedures (that may evolve over time). An interesting issue then is to maintain this team idenity and the features it encompasses over time, particularly in the face of team adaptations and changing team membership. Establishment and maintenance of a team identity are irrelevant for a non-persistent team.

We hypothesize that the key challenges in persistent teams arise from *reflective persistence*. That is, persistence in itself is not particularly significant; indeed, team members could together just wait (executing "no ops") for extended periods. Instead, the key issue of interest is a team's reflection and reasoning about its persistence, causing appropriate modifications in the team's behaviors. Such reflection could be done off-line by a human team designer, planning for all of the contigencies ahead of time. However, such off-line reflection may be problematic in domains with increasing complexity and scale, given

the increased complexity of off-line design[18]. Instead, for improved flexibility in such domains, the team itself would need to reflect upon both its past persistence (e.g., for team adaptation and reorganization from experience) and future persistence (e.g., for appropriate resource allocation).

In this paper, we certainly do not intend to address the entire range of novel issues brought forth by persistent teams, as mentioned above. Instead, we take an initial step, focusing on one key issue: 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[10], 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[31, 17, 29]. These models are based on previous theories of teamwork[23, 19, 11]. 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 focuses on enabling 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*[3] 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[31, 30]. 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, teams of synthetic attack-helicopter pilots based on STEAM successfully participated in the ACTD simulation exercise mentioned above[31]. STEAM-based soccer-players participated in the RoboCup'97 and RoboCup'98 tournaments, winning the third place prize in RoboCup'97 and coming in fourth in RoboCup'98 in over 30 teams that participated in both tournaments[33, 32]. Nonetheless, our investigation has broader applicability. STEAM itself is based on the joint intentions theory[23], and is also influenced by the Shared-Plans theory[11]. Thus, lessons learned here are applicable to other teamwork models based on such theories, such as the joint responsibility model[17] which is also based on joint intentions.

The rest of this paper is organized as follows: Section 2 provides background on STEAM. Section 3 discusses the implications of the use of the teamwork model in persistent teams. Section 4 discusses STEAM-L that uses a representation of Markov decision processes (MDP) and finds approximate solutions to the MDP by value iteration using any-time lookahead search. Section 5 presents experimental results from STEAM-L from one artificial domain and two real domains. Section 6 discusses related work. Finally, Section 7 summarizes and presents issues for future work.

2. Background: STEAM

There are two key aspects of STEAM that are relevant for persistent teams. First, it provides an implemented framework for team development. Second, it provides an explicit, general teamwork model for team-

work reasoning during execution. The team development framework is based on two separate hierarchies, each with key constraints:

- *Team organization hierarchy and roles:* In STEAM, a team may have a flat or a hierarchical organization (where a team may be recursively composed of subteams). Thus, for instance, a *company* of attack helicopter pilots (8 helicopters) may be composed of two *teams*, which in turn consist of individuals. A critical feature of this organization hierarchy is that it may be based on particular *roles*. Roles are of two types:

  Persistent roles: These are long-term assignments of roles to the individuals or subteams in the organization. For instance, in the synthetic attack-helicopter domain, a team may be designated as the scouting team, or an individual may be designated the commander. This assignment typically will not change in the short term.

  Task-specific roles: These are shorter-term assignments of roles, based on the current task and situation. For instance, in formation flying in the attack-helicopter domain, the assignment of the leader of the formation is determined based on the type of formation. For instance, a particular helicopter *H1* may be the leader when flying into the battlefield, but another helicopter *H2* may be the leader of the formation when flying out of the battlefield.

  The assignment of roles to individuals or subteams is based on their capabilities. However, this assignment may not be provided ahead of time, so that individuals may need to volunteer or be requested to fill in the roles. Furthermore, if an individual's or subteam's capability degrades during performance, roles may be reassigned (as discussed in more detail later in this section).

- *Team activity hierarchy:* STEAM relies on an explicit representation of team activities in the form of *team operators* or reactive team plans. For a concrete example, consider the operator (reactive-plan) hierarchy shown in Figure 1 for synthetic helicopter pilots developed in STEAM. This operator hierarchy is similar to normal reactive-plan hierarchies in architectures such as RAPS[9], PRS[28], or Soar[25]. STEAM is itself currently realized as an enhanced version of Soar[25]. However, the key novelty here is the *team operators* (reactive team plans) in this hierarchy. Thus, operators shown in brackets, i.e., [], such as [Engage] are team operators (others are individual operators, which express an agent's

own activities). At any time, only one path through this hierarchy is active in a pilot agent.

As with individual operators, team operators also consist of: (i) precondition rules to help activate an operator; (ii) application rules to apply active operators; and (iii) termination rules to terminate active operators. However, while an individual operator applies to an agent's private state (an agent's private beliefs), a team operator applies to an agent's *team state*. A team state is the agent's (abstract) model of the team's mutual beliefs about the world, e.g., the team's currently mutually believed strategy. There is of course no shared memory, and thus each team member maintains its own copy of the team state, and any subteam states for subteams it participates in.

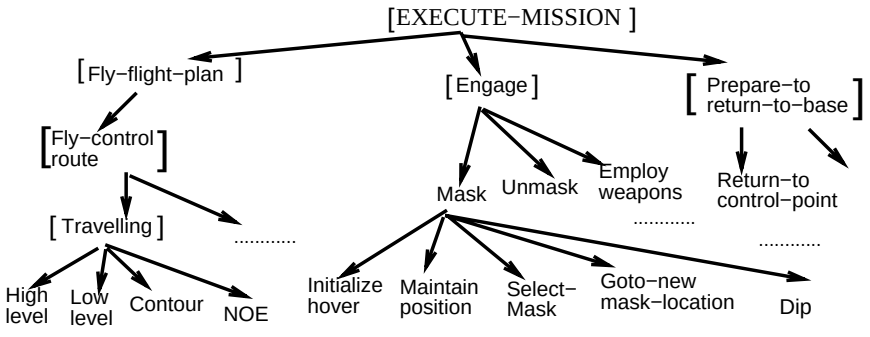


Figure 1. Attack helicopter domain: Portion of team operator (reactive-plan) hierarchy. Operators enclosed in [ ] are team operators, others are individual operators.

The key here is that there is an organization hierarchy independent of the task hierarchy, with an *implicit* expectation that the organization will be persistent. This paper will focus on making explicit the implications of this previously implicit expectations of persistence.[1]

The mapping between the two hierarchies mentioned above is centered on roles. In particular, suppose a team is executing a team operator, with several suboperators. Then, the role of a subteam or individual in that team may constrain the suboperators it can execute in service of the team operator. Thus, if a team Θ is executing a team operator [OP], then the assignment of a *role* γ_i to a member μ of Θ, constrains μ to a certain subset σ_i of the suboperators of [OP]. Furthermore, roles may have specific *coordination relationships* among them. For instance,

[1] Note that STEAM does not prohibit two agents from different parts of the organization to form a non-persistent team, if this is required.

the execution of operators for one role may be dependent on operators for another role. The concept of roles has been discussed before in the multi-agent literature[19]. The key in STEAM is the instantiation of this concept as part of the team organization hierarcy, with its use as a constraint on the selection of the sub-operators of a team operator, and its use in expressing coordination relationships.

The second, and perhaps the more important contribution of STEAM is that it provides an explicit, general-purpose teamwork model, which encode domain-independent directives or actions that explicitly outline team members' responsibilities and commitments in teamwork. In essence, it attempts to provide agents with commonsense knowledge of teamwork, that would enable them to autonomously reason about coordination and communication in teamwork. Thus, it aims to substantially reduce the effort in encoding agent teams, and improve teamwork flexibility. STEAM uses the joint intentions theory[23, 6] as the basic building block of teamwork, but it is also strongly influenced by the SharedPlans theory[11, 12], and constraints realized in practical applications.

Each agent runs the STEAM teamwork model separately and no shared memory is assumed. The tie between the teamwork model running in each agent and the previously discussed team operators is as follows: when executing (sub)team operators, each agent brings to bear all of STEAM's teamwork reasoning, which facilitates its communication and coordination with its teammates. Currently, STEAM's teamwork knowledge could be categorized into three classes of domain-independent teamwork actions. The first class is *coherence preserving* or *CP* actions. These require that team members communicate with others to ensure coherent initiation and termination of team operators. Thus, for instance, if an agent privately discovers that the currently selected team operator is either achieved, or unachievable, or irrelevant, then it must not just terminate the team operator on its own. Instead, it must communicate this unachievability information about the team operator to its teammates, so that teammates do not waste their resources, and that the team as a whole coherently terminates the team operator. The second class of *monitor and repair* or *MR* actions detect if a team task is unachievable due to unexpected member (individual or subteam) failure. It then leads the team into reorganization — via reassignment of roles — to overcome this failure. As an example of STEAM's application, consider the synthetic attack-helicopter pilot team in a situation where the pilot team is flying in formation. If one of the helicopter pilots sees some unanticipated enemy vehicles on their flight route, STEAM's *CP* actions require that this pilot inform its teammates about the en-

emy units (since it makes the relevant team operator unachievable), so that the team reacts coherently to the enemy.

STEAM requires the third set of *selectivity-in-communication* or *SC* actions because attaining coherence via CP actions for each team operator can sometimes cause excessive communication. Improved execution strategies, that aim at attaining coherence without always invoking CP actions, can reduce such communication overheads. To this end, STEAM relies on decision-theoretic communication selectivity. The key idea is to explicitly reason about the costs and benefits of different techniques for attaining mutual beliefs during the execution of CP or MR actions. For instance, in some cases, if there is high likelihood that the relevant information can be obtained by other teammates via observation (even if with a slight delay), then costly communication can be avoided. In contrast, communication becomes essential if there is low likelihood that teammates can obtain the relevant information independently, and if there is a high cost to not making this information available to the teammates. The key here is safe and efficient execution of a decision to attain mutual belief.

STEAM's teamwork model has also been currently encoded in the Soar integrated architecture, in the form of 283 rules[31]. These rules, with documentation, traces and pointers to their usage are available at (*www.isi.edu/teamcore/tambe/steam/steam.html*).

3. Implications for Persistent Teams

STEAM's *CP* and *MR* actions are focused on optimally performing its currently active team operators. In fact, STEAM uses decision-theoretic techniques in its *SC* actions to optimize execution costs of *CP* actions. Unfortunately for a persistent team, STEAM does not reason about the longer-term impact of its optimal execution of the suggested *CP* and *MR* actions. For instance, to execute a *CP* actions, a helicopter pilot *P1* may need to fly its helicopter to the location of other helicopters to communicate information regarding enemy vehicles. Since such direct communication avoids breaking radio silence, it may optimally fulfill the helicopter team's current joint goal of scouting for enemy vehicles. However, from a long-term perspective — where the helicopter team must next attack the enemy vehicles — *P1*'s locally optimal actions may consume precious time, so that the enemy vehicles move out of range, and can no longer be attacked. Furthermore, when executing *CP* actions STEAM does not consider the possibility of achieving partial coherence in a team, given the resource cost of attaining full coherence. For instance, coherence may be attained in

only a portion of a team (say 70% of the team members), rather than the entire team.

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 provide further illustration 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 be able 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 would insist 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 team operator 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 presented in Section 3, we have developed 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. Just as with STEAM, each team member runs STEAM-L separately (again no shared memory is assumed) — thus, each member computes 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 original decision-theoretic reasoning (SC actions) does not change the basic commitments of individual joint actions — it only attempts to minimize the *execution* costs for embedded *CP* actions — STEAM-L's reasoning can sometimes change the nature of these commitments.

In the following, Section 4.1 discusses the representation adopted in STEAM-L, while Section 4.2 discusses STEAM-L's search algorithm.

4.1. State space and Markov decision process

For efficient operationalization of the lookahead reasoning in STEAM-L, we have cast it as a Markov decision process (MDP)[26] in the

state space of team's future states. Given that the lookahead reasoning involves uncertain actions and outcomes, it fits the MDP framework very well, allowing us to leverage the extensive research in MDPs. More specifically, a state in this state-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 team operators. At a state, an agent simulates the execution one of the available operators (by the team), which corresponds to the actions in the MDP. Since agents may spend resources when taking actions, for instance firing a missile during a combat, there are costs associated with actions. Furthermore, due to uncertainty in the real world, the outcomes of executing team operators (agents' actions in our MDP) are not predictable, and agents can only estimate the likelihood of particular outcomes. Thus, an action may lead the current state to one of the possible future states (outcomes). We assume that agents' selection of a particular action depends only on the current state. Therefore, the overall lookahead search can be naturally cast as a MDP, in which the MDP's states are the team's future states, state transitions correspond to the team's current and future executable operators or actions, and transition probabilities are the probabilities of going from one state to another after an action.

To differentiate two types of sources of costs, we adopt a decision tree [27] representation in this research, although an MDP is a more compact representation of the decision tree. There are two types of 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. The outcome nodes are transient, meaning that they do not correspond to stable states. They are introduced merely for the clarity of representation. 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. As seen in the figure, action nodes are followed by outcome nodes, and vice versa. Associated with a frontier node in the tree is the final situation cost.

State spaces are in general graphs. In this research, we concentrate on state spaces whose structures are trees for two reasons. First, a state-space tree is still very general, as it can used to represent a state-space graph with duplicate nodes[38]. Second, tree structures are conceptually simpler to address, at least in this initial step to address

persistent teams. This representation may be modified for efficiency, but that remains an issue for future work.

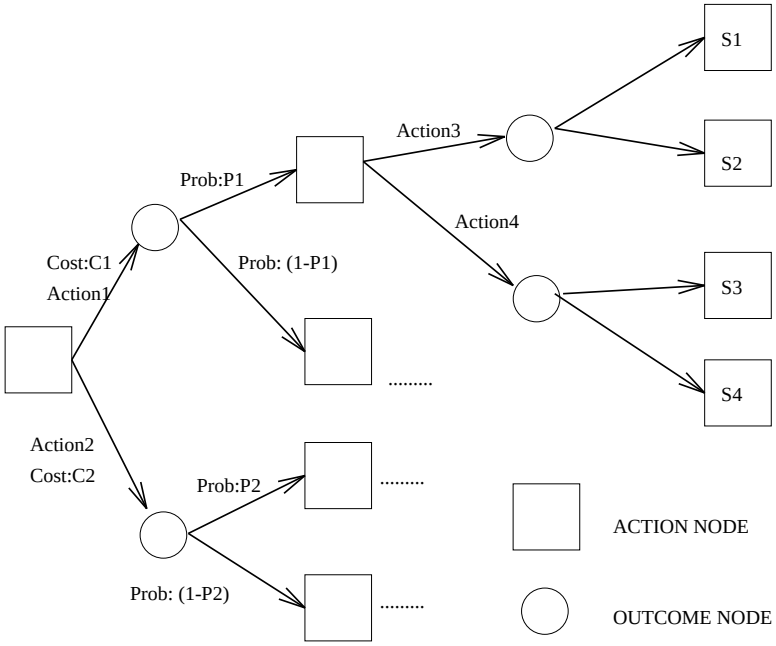


Figure 2. Decision tree search.

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. STEAM-L focuses on agents' decisions with respect to *CP* and *MR* actions, and it explores their variations in its lookahead search. For instance, a CP action may require full coherence for a team operator such as [Engage]. In STEAM, this requires the entire team to commit to [Engage], without regard to the time it takes to ensure the entire team's commitment. A variation of this CP action may suggest partial coherence. Partial coherence may choose to obtain commitments within a fixed time limit. Such a partial coherence action may be estimated to obtain commitments from 80% of the team within the fixed time limit, with high likelihood (0.8). Another variation of the CP action may suggest no coherence. With no coherence, any commitments may be those that arise implicitly. Other such variations can also be considered in the lookahead search in STEAM-L. These variations on CP and MR actions enable a STEAM-L agent to reason about the coordination for the entire team (or subteam) executing the relevant team operator (e.g., [Engage]). That is, the search is not focused on its own individual coordination.

Figure 3 depicts the search in progress. Three variations of the CP actions are shown. Starting from the state S1, the partial coherence action is shown with 0.2 probability to reach state S2, while with 0.8 probability to reach state S3. State S2 in this case obtains commitments from a lower percentage of the team than state S3, so that S2 has a higher situation cost than S3. No-coherence is shown to achieve different results than partial coherence. Full coherence is the third option. It has a very high probability of obtaining commitments from the entire team (0.99), which has no situation cost associated with it. These numerical estimates of likelihood focus on just the teamwork actions (as discussed below, we do not consider variations of domain-specific operators). These estimates are based on heuristics that reflect our prior experience. Fortunately, since these actions are domain-independent, the relevant likelihood estimates may not necessarily be re-engineered for each new domain. Furthermore, they need not be extremely accurate. Indeed, as we illustrate in later sections, the lookahead may be tolerant of errors.

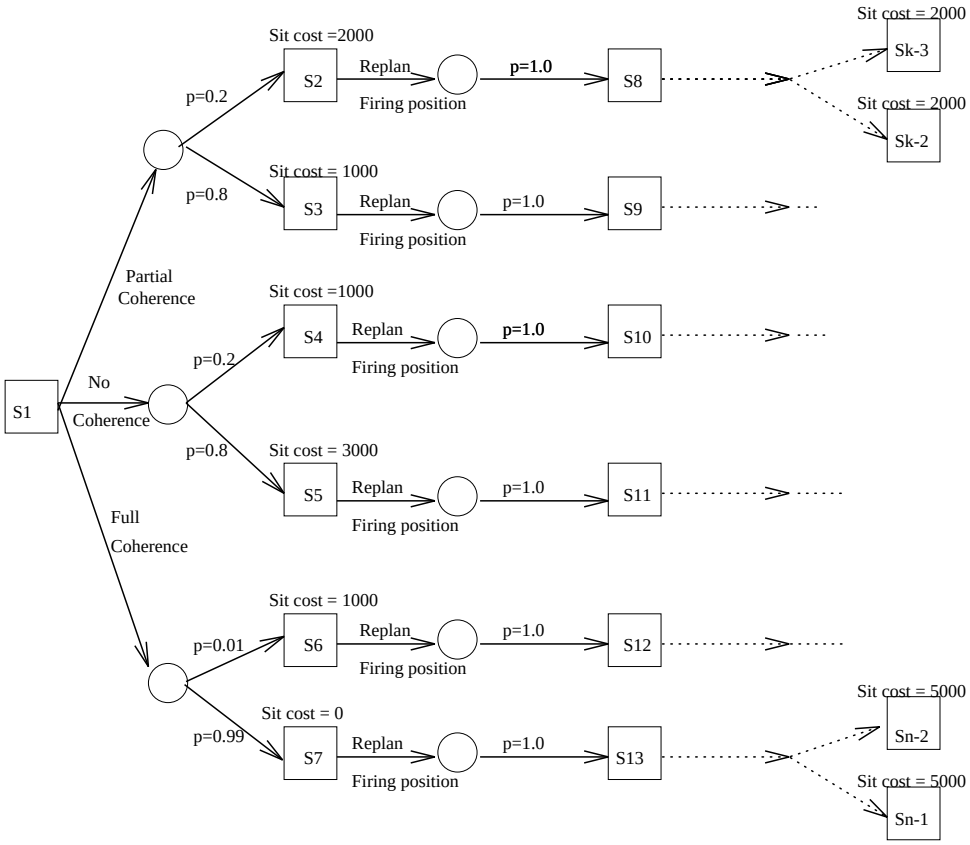


Figure 3. STEAM-L's decision tree search in the attack-helicopter domain.

The second type of actions in STEAM-L's lookahead search are domain-specific actions, such as the [Engage] operator, or the [replan-firing-position] operator. In principle, variations of these actions could also be explored in the lookahead search, but that will be outside the scope of this paper. Thus, as shown in Figure 3, these actions (such as [replan-firing-position]) are assumed to achieve the desired result with the probability of 1.0. 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.

STEAM-L's lookahead search process is separate from the actual execution of both the team operators and STEAM's CP/MR actions; the lookahead is truly attempting to lookahead prior to the execution. Thus, to realize STEAM-L, new lookahead search code had to be added to STEAM. This lookahead search is well-informed about teamwork actions, and thus will automatically insert the relevant teamwork-related CP and MR actions in the search, e.g., full-coherence, partial-coherence and no-coherence shown in Figure 3 occur prior to the [replan-firing-position]. However, at present, the expected chain of execution of the domain-specific operators has to be pre-defined by hand for STEAM-L's lookahead search, e.g., the team operator [replan-firing-position] is typically followed by an individual operator for get-to-firing-position, and so on. In the future, providing abstract models of the operators (e.g., abstracted preconditions) may enable STEAM-L to automatically infer the expected execution path of the domain-specific operators as well.

4.2. Value iteration using any-time lookahead search

In the search tree of Figure 2, an agent selects among the multiple possible actions at a state by computing expected utilities of different actions, and selecting an action that maximizes projected long-term team utility. This long-term utility reflects an agent's view of its team's future states, combining the cost of the resources consumed and the rewards obtained in the future states that the agent team may reach. In terms of the search tree mentioned earlier, the expected utility of a node (state) is determined by static node cost evaluation of the search frontier and node cost backup rules, which are discussed below.

Even though the number of team's future possible states is generally huge, agents must compute expected utilities efficiently in an anytime fashion, and have a decision on next action ready quickly to respond to unexpected events. To this end, STEAM-L carries out an iterative-deepening lookahead search [21]. STEAM-L is always activated at a

state where a decision on the next *CP* or *MR* action needs to be made. When a decision needs to be made, the decision from the last iteration of lookahead search will be used. 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 runs a state-space search algorithm, such as best-first search or depth-first search. In our experiments, we use depth-first search due to its small space requirement. 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 rolled back 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 state toward the best child state, the one with the smallest expected cost in 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. The resource cost $r(n)$ is the total cost of consumed resources, such as time used and operators applied, in order to reach the current node n, and the situation cost assesses the situation quality of the node, such as the inability of one or more team members to participate in a joint activity at that node. 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).

For an example of STEAM-L's computation, consider the example in Figure 3. With the first iteration, STEAM-L expands states S2 through S7. Here, it returns full-coherence as the best action at state S1, since it is one that maximizes expected utility, i.e., it is the action with the smallest expected cost. In the next iteration, STEAM-L expands the next frontier, considering the domain-specific action [replan-firing-position], and considering the situation costs of the nodes along that frontier. In the third iteration, further actions will be explored. As the search horizon deepens, full-coherence may no longer be returned as the action at S1 that maximizes the expected utility. Indeed, as seen in the figure, full-coherence appears to lead to a much higher situation cost.

The iterative process of lookahead search corresponds to the value iteration method for solving finite horizon MDP problem [26]. Starting with lookahead depth one in iteration one, the process continues to extend its lookahead horizon until it reaches the depth of a goal node. In this process, a lookahead of depth k populates values at a search horizon k steps back to the initial starting state. In essence, the iterative lookahead search is an approximation method for computing the optimal long-term expected team utility, which determines action selection. The approximation comes from two factors. First, goal states and their horizons (their depths in the search tree) are unknown a priori. Second, due to time pressure and unknown goal depth, iterative lookahead search progressively extends its search horizon, and assumes that the quality of heuristic evaluations at a search frontier improves with the search frontier approaching goal states. Given the information within the current lookahead horizon, the action taken based on Equation (1) is optimal. This is because this equation follows Bellman's principle of optimality for stochastic dynamic programming [2], which ensures that the action determined by the value at the root node is optimal within the current search horizon.

While both STEAM and STEAM-L share the same theoretical foundations in their 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. STEAM-L thus adds to the coordination options available to the agents, e.g., partial coherence rather than full coherence, thus introducing additional flexibility in teamwork.

Unfortunately, STEAM-L's lookahead and additional coordination actions could also lead to an increased possibility for disagreement among agents' beliefs on the right teamwork coordination action to apply.[2] In particular, while such disagreements are a possibility in STEAM[31], the increased flexibility in STEAM-L could potentially exacerbate the situation. Such disagreements could arise because each agent, with its own STEAM-L reasoning, could come to a different

[2] We thank the reviewers of the AAMAS special issue for raising this concern.

conclusion about coordination. For instance, one agent may believe that partial coherence is the right action at a given time, while another believes that no coherence is a better action.

In practice, such disagreements may not be as problematic for several reasons. First, the disagreements are not expected be common. Indeed, our earlier practical experience with STEAM, and current experience with STEAM-L indicates that agents do not frequently reach widely varying conclusions about the right coordination action to follow. This is because agents' estimates of likelihoods and situation costs do not vary widely — at least in homogeneous teams, where STEAM-L is being applied. Second, small variations in estimates may not cause differences in agents' conclusions, as shown in Section 5.1. Third, by convention, agents may agree to adopt the team coordination action proposed by one of the agents, e.g., either a designated leader, or whoever has made the decision first. However, if disagreements do occur and conventions turn out to be impractical, agents may need to negotiate. Here, we have been investigating negotiation techniques such as argumentation[35], which would be applicable in such situations. Indeed, agents may even deliberately search different parts of the space, so that they divide up the search effort, and search more effectively.

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 static node evaluation. Specifically, the static 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.

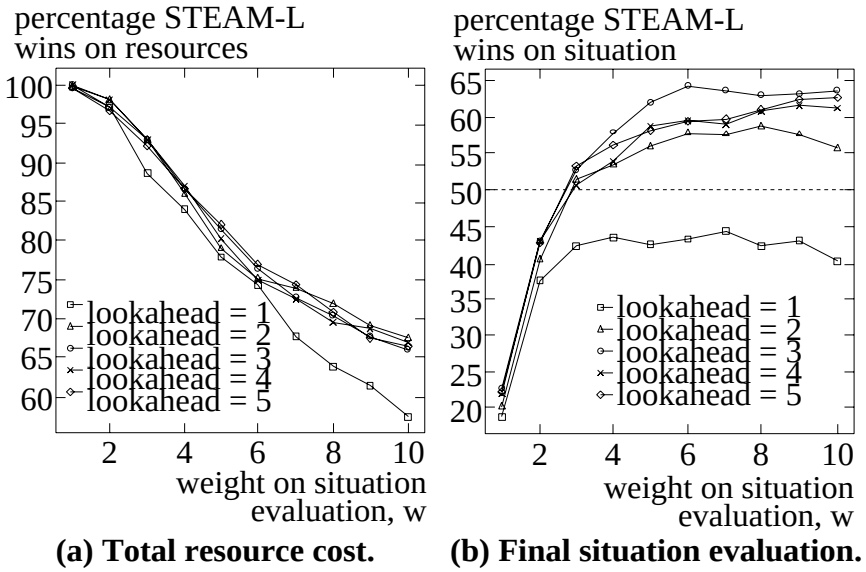
5.1. Artificial search tree

The artificial search tree that we will use in our experiments is a variation of an incremental random tree [38]. The motivations to use such a 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 the results from the domain help us to better understand the phenomena in real domains. In addition, the use of artificial trees can assist us to examine some of the important features that may be difficult to measure in a real domain, as we will see from our experiments presented in this section.

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 future states or nodes. The degree of dependence depends on the number of edges these 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 nodes. We then compute the situation cost $s(n)$ of a node n as the sum of the random number of the node and the situation cost of its parent node. We use the same scheme to derive resource costs. Specifically, we assign random numbers to the edges in the tree, and compute the resource cost $r(n)$ of a node n as the sum of the numbers on the edges on the path from the root to the node. In our experiments, random edge costs are uniformly chosen from a set of $\{1/M, 2/M, \cdots, 1-1/M\}$, and random numbers on nodes are similarly chosen from another set of $\{1/M, 2/M, \cdots, 1-1/M\}$, where $M = 2^{32} - 1$.

Incremental random trees (the basis of our artificial search trees) appear to be good models for evaluating the performance of lookahead search. The incremental nature of the situation and resource costs in incremental trees reflects the property that heuristic estimates of future goal states become more accurate when the current states are closer to the goal states. In the soccer game case, for instance, the estimate whether a player controlling the ball can score is more accurate when the player brings the ball closer to the goal.

We experimentally compared STEAM-L against STEAM on artificial search trees described above. Figure 4 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 horizontal axes are the



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

Figure 4. STEAM-L vs. STEAM on trees with branching factor 3 & depth 40.

weight w on situation cost in a static evaluation $f(n) = r(n) + ws(n)$, and the vertical axes are the percentage of instances on which STEAM-L outperforms STEAM. We considered different lookahead depths in our experiments. The results are averaged over 500 instances.

Figure 4(a) shows that by taking resource costs into account and looking deeper, STEAM-L spends less total resources than STEAM with a large percentage. It also shows that when the weight on situation cost w increases, the percentage that STEAM-L wins on total amount of resources used 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 4(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, such as larger than 6 in Figure 4(b), increasing w does not help STEAM-L significantly.

Figure 4 also means that by carefully adjusting the parameter w, STEAM-L can outperform STEAM in both resource costs and situation cost if STEAM-L searches deep enough. For the artificial search trees considered in Figure 4, for instance, a good value of w is around 6 and 7. When $6 \leq w \leq 7$, STEAM-L outperforms STEAM more than 70% on the total amount of resources used and more than 55% on the final situation evaluation.

In lookahead reasoning, the exact resource and situation costs of future states are not known, and must be estimated based on some domain heuristics. The accuracy of the heuristic estimates will have impact on the accuracy of the decision of next move. The impact of errors in these estimates can be directly examined on our artificial search tree model, which is difficult in a real domain since it is often difficult to estimate the accuracy of our domain heuristics. To investigate the impact of such errors, we add a random noise to resource and situation costs in the lookahead search. We then examine how noise in the estimates can cause a change in the decision on the next action the agent team will take. In other words, we examine the robustness of lookahead reasoning under inaccurate heuristic estimates of resource and situation costs. Specifically, we examine the probability that the decision on the next action will be the same in terms of the ratio of noise to the original heuristic estimates.

To accomplish this in an artificial search tree, we introduce another set of random numbers and add them to the resource and situation costs, and run the same lookahead search algorithm on this noise-effected tree. We then compare the decision on the next action based on this noise-effected tree to the decision on the same search tree without the noise. We obtain an experimental probability that lookahead search reaches an identical decision by averaging the results from 500 instances. In these experiments, we used two sets of artificial search trees. The trees of the first set are the same as those used in the previous experiments. The ones of the second set are the same as those in the first set, but with additional random numbers being added to the resource and situation costs. The random numbers represent noise and can take positive or negative values. In our experiments, we took these random numbers uniformly from $\{-r, -r+1, \cdots, 0, \cdots, r-1, r\}$, where $r <= 1 - 1/M$ and $M = 2^{32} - 1$. Recall that $1 - 1/M$ is the maximum value used for the resource and situation costs. Then, the ratio between r and $1 - 1/M$ can be considered as the noise to signal ratio. Denote this ratio as ρ.

Figure 5 shows the results of our experiments, averaged over 500 instances. These results show that lookahead reasoning is fairly robust in the presence of inaccuracy in heuristic estimates. Figure 5(a) shows the probability that the lookahead search makes the same decision in terms of the noise to signal ratio ρ on search trees with branching factor 3 and depth 40. The lookahead depth is fixed at 5. The figure shows that lookahead search is fairly robust. Even if when there is 100% noise, lookahead search is still able to make a correct decision on the next action more than 60% of the time. In contrast, a random decision is only about 33% correct. Figure 5(b) shows the robustness of lookahead

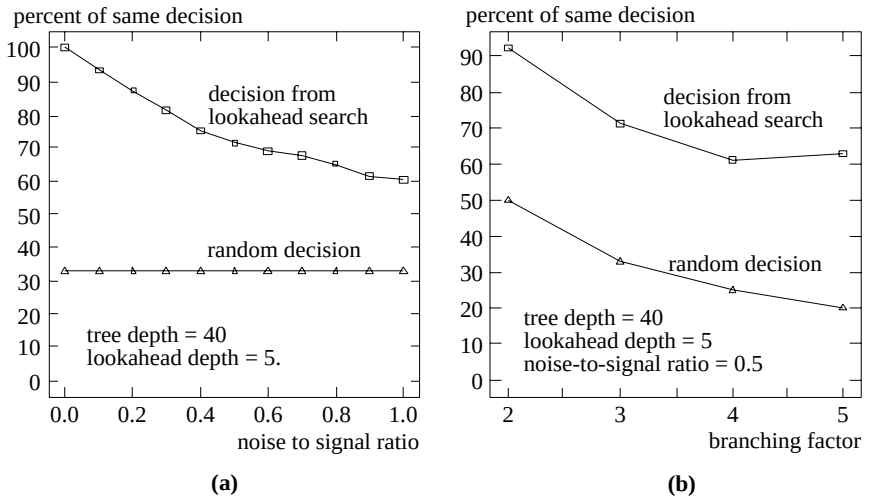


Figure 5. Robustness of lookahead search, measured as percentage of correct moves under noise.

search in terms of the branching factor of a search tree with lookahead depth at 5 and noise ratio at 50%. Although the probability of correct decision drops when the branching factor increases, it is more than twice of that for a random decision. For instances, on a binary tree, the percentage of correct actions of random decision is 50% and that of lookahead search is more than 92%; and on a tree with branching factor 5, the percentage of correctness of random decision is 20% and that of lookahead search is more than 62%. Overall, these results indicate that lookahead reasoning may not suffer significantly even in the presence of some inaccuracy in heuristic estimates.

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. We have discussed the issue of obtaining likelihood estimates for teamwork related actions earlier in Section 4.1. For situation costs, the estimates used in our experiments are based

on domain heuristics. While these estimates may be imprecise, the results would appear to be quite tolerant of errors in the heuristic estimates. This is confirmed in part by our experimental analysis of noise (in cost estimates) in artificial search trees in Section 5.1, and in part, our analysis of the search trees in the real domain, which tend to be more structured (as discussed in Section 5.3).

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 team operator), 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 team operators 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 team operator 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. However, in RoboCup'97 this strategy was pre-compiled, rather than performing the entire search in real-time. Experience-based 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 4. 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.

The above analysis does not imply that the results on artificial trees are not useful. Instead, those results help us understand where STEAM-L's lookahead search would be most effective — in state space where resources are scarce and situation costs vary dramatically. Indeed, the lookahead search in STEAM-L would appear very effective to detect possible dramatic situation changes. Thus, it would appear beneficial to invoke STEAM-L's lookahead selectively, only in such cases. Furthermore, the results on artificial trees indicate that in such circumstances (where STEAM-L's lookahead will make a difference), the lookahead may be tolerant of errors in heuristic estimates of costs and likelihoods.

6. Related Work

Multi-agent collaboration, coordination and teamwork in general have been explored extensively in the multi-agent literature. We have discussed some of this work in Section 1. More relevant to the work discussed in this article are issues of persistence. Focusing first on existing work on teamwork theories[23, 19, 11] and implemented teamwork models[31, 17, 29], as Section 1 mentions, they have not explicitly taken persistence issues (as outlined in this article) into account. Thus, for instance, they do not include reasoning about the long-term implications of coordination actions that appear locally optimal, but are highly suboptimal in the long term.

Thus, persistent teams present an important challenge for formalization in teamwork theories. Indeed, STEAM-L has a key theoretical implication for the notion of commitments in teamwork. In joint intentions work[6], all commitments are formalized as hard commitments, to be dropped only when they are achieved, unachievable or irrelevant. Commitments are yet to be formalized in SharedPlans[11], and in [19], commitments can be dropped for selfish reasons (which appears inappropriate for teamwork, at least in general). STEAM-L presents a novel approach to introduce significant flexibility in such commitments, important in increasingly uncertain environments. In particular, team members may not necessarily fulfill their commitments (for instance to inform others), if doing so harms the longer-term interests of the team. Thus, the flexibility in commitments is motivated not by selfish

interests, but by team interests themselves. To formalize such flexible commitments, one suggestion is to add STEAM-L's lookahead computation to the "relativizing clause" in the joint intentions framework[23][3] However, this suggested modification by itself is insufficient. In particular, the decision in STEAM-L is not a just binary one – fulfilling the commitment or not. Instead, STEAM-L admits the possibility of partial coherence, where there may be different likelihoods associated with different degrees of coherence within a team.

STEAM-L is also related to distributed coordination frameworks, particularly, the *Generalized Partial Global Planning*(GPGP) framework [7] that is concerned about generating longer term coordination schedules. This rich body of work has a long history, evolving from earlier work, such as partial global planning (PGP) [8]. It is difficult to discuss this work in depth in brief, but we will highlight some key aspects relevant to this paper. GPGP is based on the TAEMS (Task Analysis, Environmental Modeling and Simulation), a task modeling framework, which has been reported to be used in multiple domains. In TAEMS, tasks are hierarchically represented (tasks decomposed into subtasks), with leaf-level nodes representing problem-solving methods. Subtasks of a task may represent alternative ways of accomplishing the task, and may be related to each other via interactions like enablement or facilitation (e.g., one subtask S1 enables another subtask S2). There is a quality associated with each task, with a quality accumulation function based on the quality of the subtasks. Given a group of coordinating agents, each may have a subjective view of a high-level task, since each may see only a portion of the task (and there may be many such high-level tasks being executed in parallel). Each agent must locally select and schedule its portion of the subtasks and methods, to meet the criteria specified in terms of quality, cost, duration etc. Obviously, given that each agent is only responsible for a portion of the task, it must coordinate with others. Here, GPGP defines a set of (at least) five modularized, extendible, domain-independent coordination modules that enables the different agents to discover the coordination relationships with each other, and to enforce these relationships. For instance, to handle the enablement relationship among subtasks (e.g., S1 enables S2), GPGP defines a hard-task relationship module. This module sends a commitment from an agent with the enabler subtask (S1 in S1 enables S2) to another agent (with the S2 subtask), to perform S1 by a deadline. Other such mechanisms are defined, and together they place constraints (commitments) on an agent's local scheduler,

[3] This suggestion was made by a reviewer for our paper [37], which was an earlier version of the current article.

so that shedules may attempt to honor commitments, while also using commitments made by other agents.

In general, the STEAM framework and GPGP appear to have complementary strengths, and indeed, a hybrid architecture could potentially outperform either one of these. For instance, TAEMS and GPGP do not *explicity* address joint goals and joint commitments which are central in STEAM. For instance, the notion that a team of agents is jointly commited to the joint goal of *Engage*, is not the same as one agent in the team enabling another one to engage, or all agents performing different subtasks of engage which enable each other. Instead, with joint commitment in STEAM, the team of agents jointly take on the responsibility to engage the enemy, to monitor the progress of engagement, to inform each other about such progress, to change roles if an agent can not do its part, and so on. While the local scheduling in GPGP is similar to the lookahead in STEAM-L, there are key differences: (i) STEAM-L's MDP process ususally reasons about the set of joint goals (team operators) of the team, rather than individual agent's own subtasks; (ii) the results of STEAM-L's reasoning process relate to coordination to be adopted in the entire team, and not among two individuals; (iii) the MDP process itself appears different than the scheduler used in GPGP. Howver, this MDP based process is not as evolved as the multi-criteria scheduling work in GPGP, and could possibly make use of such work if appropriately extended to handle joint goals and commitments.

Finally, while persistent agents and teams remain uninvestigated in general, one exception is Horvitz's work on continual computation[15]. 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 concerns 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.

Markov decision process (MDP) is one of the main problem representations for problem solving involving uncertainty, especially decision-theoretical planning, in AI [4]. Our approach of reasoning into multi-agents' future states using iterative lookahead search is in fact an approximation method for solving MDP. Specifically, it is a value iteration method to find the optimal action within a given computation resource. Our approach falls into the category of real-time dynamic pro-

gramming [1], which extends real-time heuristic search [22] to domains with unpredictable actions.

7. Summary, Discussion, Future Work

Recent progress in theories of teamwork[11, 23, 19], as well as implemented models of teamwork inspired by such theories[17, 29, 31], have led to an improved understanding of teamwork, and to complex teamwork applications. In our own previous work, we have applied STEAM[31], 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 a representation of Markov decision process and anytime lookahead search; (iv) application and analysis of STEAM-L in different 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[23], and also borrows from SharedPlans[11], the results bear upon other teamwork models based on such theories.

Several issues for future work have been mentioned earlier, such as experience-based compilation of STEAM-L's reasoning via methods such as explanation-based learning[24] for time-critical situations, selective application of STEAM-L's reasoning to avoid unnecessary computation, and formalization of commitments in the context of persistent teams.

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.

8. 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, the RoboCup simulation group for discussion of issues related to RoboCup, and Keith Decker and Tom Wagner for discussions related to STEAM's relationship with GPGP. We also thank the reviewers of ICMAS'98 and of this special issue of AAMAS — they helped us to significantly improve the quality of this article. This article extends our earlier conference paper [37] from ICMAS'98.

References

1. Barto, A. G., S. J. Bradtke, and S. P. Singh: 1995, 'Learning to act using real-time dynamic programming'. *Artificial Intelligence* **72**, 81–138.
2. Bellman, R. E.: 1957, *Dynamic Programming*. Princeton University Press.
3. Boddy, M. and T. Dean: 1989, 'Solving Time-Dependent Planning Problems'. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. pp. 979–984.
4. Boutilier, C., T. Dean, and S. Hanks: to appear, 'Decision theoretic planning: Structural assumptions and computational leverage'. *Journal of Artificial Intelligence Research*.
5. Carley, K. and D. Svoboda: 1996, 'Modeling organizational adaptation as a simulated annealing process'. *Sociological methods and research* **25**, 136–168.
6. Cohen, P. R. and H. J. Levesque: 1991, 'Teamwork'. *Nous* **35**.
7. Decker, K. and V. Lesser: 1995, 'Designing a family of coordination algorithms'. In: *Proceedings of the International Conference on Multi-Agent Systems*.
8. Durfee, E. and V. Lesser: 1991, 'Partial global planning: a coordination framework for distributed planning'. *IEEE transactions on Systems, Man and Cybernetics* **21**(5).
9. Firby, J.: 1987, 'An investigation into reactive planning in complex domains'. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
10. Franklin, S. and A. Graesser: 1996, '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*. New York: Springer-Verlag.
11. Grosz, B. and S. Kraus: 1996, 'Collaborative plans for complex group actions'. *Artificial Intelligence* **86**, 269–358.
12. Grosz, B. J. and C. L. Sidner: 1990, 'Plans for Discourse'. In: P. R. Cohen, J. Morgan, and M. Pollack (eds.): *Intentions in Communication*. Cambridge, MA: MIT Press, pp. 417–445.
13. Hayes-Roth, B., L. Brownston, and R. V. Gen: 1995, 'Multiagent collaobration in directed improvisation'. In: *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*.
14. Haynes, T., S. Sen, N. Arora, and R. Nadella: 1997, 'An automated meeting scheduling system that utilizes user preferences'. In: *Proceedings of the International Conference on Autonomous Agents (Agents'97)*.
15. Horvitz, E.: 1997, 'Models of continual computations'. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
16. Jacolin, L. and R. Stengel: 1998, 'Evaluation of a cooperative air-traffic management model using principled negotiation between intelligent agents'. In: *Proceedings of the National Conference of the American Institute of Aeronautics and Astronautics (AIAA)*.
17. Jennings, N.: 1995, 'Controlling cooperative problem solving in industrial multi-agent systems using joint intentions'. *Artificial Intelligence* **75**.
18. Kaminka, G. and M. Tambe: 1998, 'What is wrong with us? Improving robustness through social diagnosis'. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
19. Kinny, D., M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhard, and E. Werner: 1992, 'Planned team activity'. In: C. Castelfranchi and E. Werner (eds.): *Artificial Social Systems, Lecture notes in AI 830*. Springer, NY.
20. Kitano, H., M. Tambe, P. Stone, S. Coradesci, H. Matsubara, M. Veloso, I. Noda, E. Osawa, and M. Asada: 1997, 'The RoboCup Synthetic Agents'

Challenge'. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.
21. Korf, R. E.: 1985, 'Depth-first iterative-deepening: An optimal admissible tree search'. *Artificial Intelligence* **27**, 97–109.
22. Korf, R. E.: 1990, 'Real-time heuristic search'. *Artificial Intelligence* **42**, 189–211.
23. Levesque, H. J., P. R. Cohen, and J. Nunes: 1990, 'On acting together'. In: *Proceedings of the National Conference on Artificial Intelligence*.
24. Mitchell, T. M., R. M. Keller, and S. T. Kedar-Cabelli: 1986, 'Explanation-based generalization: A unifying view'. *Machine Learning* **1**(1), 47–80.
25. Newell, A.: 1990, *Unified Theories of Cognition*. Cambridge, Mass.: Harvard Univ. Press.
26. Puterman, M. L.: 1994, *Markov Decision Processes*. John Wiley & Sons.
27. Raiffa, H.: 1968, *Decision analysis*. Reading, MA: Addison Wesley.
28. Rao, A. S., A. Lucas, D. Morley, M. Selvestrel, and G. Murray: 1993, 'Agent-oriented architecture for air-combat simulation'. Technical Report Technical Note 42, The Australian Artificial Intelligence Institute.
29. Rich, C. and C. Sidner: 1997, 'COLLAGEN: When agents collaborate with people'. In: *Proceedings of the International Conference on Autonomous Agents (Agents'97)*.
30. Tambe, M.: 1997a, 'Agent architectures for flexible, practical teamwork'. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
31. Tambe, M.: 1997b, 'Towards flexible teamwork'. *Journal of Artificial Intelligence Research (JAIR)* **7**, 83–124.
32. Tambe, M., J. Adibi, Y. Alonaizon, A. Erdem, G. Kaminka, S. Marsella, and I. Muslea: 1999a, 'Building agent teams using an explicit teamwork model and learning'. *Artificial Intelligence* **110**, 215–240.
33. Tambe, M., J. Adibi, Y. Alonaizon, A. Erdem, G. Kaminka, S. Marsella, I. Muslea, and M. Tallis: 1998, '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.
34. Tambe, M., W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb: 1995, 'Intelligent agents for interactive simulation environments'. *AI Magazine* **16**(1).
35. Tambe, M. and H. Jung: 2000, 'The benefits of arguing in a team'. *AI Magazine* **to appear**.
36. Tambe, M., W. Shen, M. Mataric, D. Goldberg, P. Modi, D. Pynadath, Z. Qiu, and B. Salemi: 1999b, 'Teamwork in cyberspace: Using TEAMCORE to make agents team-ready'. In: S. Murugesan (ed.): *AAAI FALL Symposium on Social Agents*.
37. Tambe, M. and W. Zhang: 1998, 'Towards flexible teamwork in persistent teams'. In: *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)*.
38. Zhang, W. and R. E. Korf: 1995, 'Performance of linear-space search algorithms'. *Artificial Intelligence* **79**, 241–292.