Towards Robust Teams with Many Agents

Gal A. Kaminka and Michael Bowling

November 2001
CMU-CS-01-159

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Agents in deployed multi-agent systems monitor other agents to coordinate and collaborate robustly. However, as the number of agents monitored is scaled up, two key challenges arise: (i) the number of monitoring hypotheses to be considered can grow exponentially in the number of agents; and (ii) agents become physically and logically unconnected (unobservable) to their peers. This paper examines these challenges in teams of cooperating agents, focusing on a monitoring task that is of particular importance to robust teamwork: detecting disagreements among team-members. We present YOYO, a highly scalable disagreement-detection algorithm which guarantees sound detection in time linear in the number of agents despite the exponential number of hypotheses. In addition, we present new upper bounds about the number of agents that must be monitored in a team to guarantee disagreement detection. Both YOYO and the new bounds are explored analytically and empirically in thousands of monitoring problems, scaled to thousands of agents.

Keywords: Artificial Intelligence, Distributed Artificial Intelligence, Coherence and coordination

1 Introduction

Agents in realistic, complex, multi-agent domains must monitor other agents to accomplish their tasks, detect failures, coordinate, and collaborate. Indeed, the importance of agent monitoring in deployed multi-agent systems has long been recognized in theory (e.g., [4, 7, 8]), and in practice, ranging from industrial systems (e.g., [12]), to virtual environments for training and research (e.g., [21, 22]), to human-computer interaction (e.g., [16]), and multi-agent robotics (e.g., [18, 2]). Agent monitoring infrastructure is of particular importance in teams of cooperating agents, since the correct execution of teamwork mandates that team-members come to agree on the task that is jointly executed by the team, and manage interdependencies among team-members [4, 8].

As the number of agents in the system increases, a monitoring agent cannot monitor all activities of all other agents at all times [12, 7, 8]. Much of previous work has therefore explored different approaches to reducing monitoring activities to accommodate the bandwidth in the target application, by relying on focused communications [12, 20], using plan-recognition to infer agents' state based on their observable behavior [10, 16], and monitoring a shared environment [22].

However, key challenges raised by a scale-up in the number of monitored agents remain largely unaddressed. First, as bandwidth is used more selectively, less information is available about monitored agents, and thus there is some uncertainty about their state. However, in many monitoring tasks (e.g., coordination, teamwork monitoring) the computational complexity of reasoning about multiple agents and their possible interactions can increase exponentially in the number of agents, even under limited uncertainty [14]. Second, previous monitoring approaches rely, in general, on the monitoring agent to be able to communicate or observe all monitored agents. However, as the number of agents increase, agents become more physically and logically separated, and thus a monitoring agent may not be able to observe (or communicate with) the agents it is to monitor. We call this challenge *limited connectivity*.

This paper addresses these difficulties in the context of a particularly important monitoring task in robust multi-agent teams—that of *detecting disagreements among teammates*. Theoretical and empirical research on teamwork in synthetic agents [4, 12, 8, 20, 15] and in humans, (e.g, [3]) stresses agreement as a cornerstone to effective teamwork (although the literature differs in the terms used and in grounding agreement in various theoretical and practical constructs). Thus disagreements are a source of great concern in all of these different investigations (see Section 2 for details).

We present two sets of contributions. First, we present YOYO, a disagreement-detection monitoring algorithm, which navigates the (potentially exponential) space of monitoring hypotheses by representing only hypotheses in which all agents are in agreement. This allows YOYO to represent the relevant state of all monitored agents together, in a single highly scalable structure. YOYO is an example of a *Socially-Attentive* monitoring algorithm, exploiting knowledge of the social relationships in the monitored team. It is based on earlier work on visualization [13], but differs from it in many ways (see Section 6).

A second set of results tackles the challenge of limited connectivity by providing new bounds on the number of agents that must be monitored in a team to detect disagreements. Previous work has shown analytically that disagreement detection can sometimes be guaranteed if all team-members monitor all of certain key agents in the team [14]. However, in practice, limited connectivity restricts the the usefulness of this bound, as often not all key agents can be observed or communicated with. To address this, we show analytically that sound and complete detection can be guaranteed in practice even if all key-agents monitor each other, but other agents just monitor one key agent. In addition, we show that unfortunately, while some potential for limiting the number of monitored agents exists in centralized settings, the worst case still requires monitoring all agents in a team.

These results are motivated by practical concerns raised in our application domains, as monitoring settings become realistic and the number of agents is scaled up. Using the techniques presented, a monitoring agent can detect disagreements in large teams, involving thousands of agents, that are guaranteed to perform specific monitoring tasks under conditions of limited connectivity. In addition to the analytic results, we present an empirical evaluation in thousands of monitoring problems, scaled to thousands of agents.

This paper is organized as follows. Section 2 presents motivating examples and background. Section 3 presents an overview of the monitoring task. Section 4 presents the YOYO algorithm. Section 5 presents new bounds on the number of agents that must be monitored. Section 6 addresses related work, and Section 7 concludes.

2 Motivation and Background

Teamwork literature, addressing human and synthetic teams, has often emphasized the importance of team-members being in agreement on various features of their state, such as goals, plans, and beliefs[1]. Teamwork theory often defines agreement as a state of mutual belief, where agents reason to infinite recursion about their beliefs and their beliefs in others' beliefs in a proposition. For instance, *SharedPlans* theory requires team-members to mutually believe in a shared recipe [8] during the planning and execution phases of the task; the *Joint Intentions* framework emphasizes mutual belief in the team goals' selection, as well as in team-members' beliefs about the goals' achievability and relevance [4, 15]. Other investigations of agent teams have emphasized agreement on team plans to be jointly executed by team-members [12], on hierarchical team operators [20], on tasks to be executed collectively [18], etc. Investigations of human teamwork have not only emphasized agreement on the joint task, but also agreement on features of the environment that are important to the task being carried out by the team [3].

However, the literature also recognizes that achieving and maintaining agreement can be difficult. Teamwork theory recognizes that attainment of agreement by mutual belief is undecidable [9] and must therefore be approximated in practice. Such approximations frequently involve assumptions of trustworthiness of team-members, of foolproof communications [12], of team-members being able to observe each other [10], and/or of a mutually-visible environment. As is often the case with approximations, they sometimes fail in practice (e.g., due to communications failures), and

[1]Of course, the literature also addresses other critical features of teamwork aside from agreement. But agreement is a repeating theme in recent work.

therefore team-members may find themselves in disagreement with each other. Such disagreements are often catastrophic, due to the unique importance of agreement in collaboration.

It is therefore critical that teams are monitored to detect such disagreements. A monitoring agent that identifies the state of team-members can compare the state of different team-members and detect differences on state features that, by design or by selection, should have been agreed upon [14]. However, as the number of monitored agents is scaled up, two challenges arise: (i) monitoring algorithm complexity due to uncertainty about the state of agents; and (ii) difficulty to observe or communicate with all agents (*limited connectivity*). We have come to realize the need to address these challenges while working on developing robust multi-agent teams in two dynamic, complex, domains: *ModSAF*, a commercially-developed, high-fidelity virtual environment, where we have been involved in the development of synthetic helicopter pilot agents that carry out a variety of missions [21]); and *RoboCup soccer simulation*, a dynamic research-oriented simulation which requires real-time teamwork and coordination, where we have been involved in the development of both soccer-playing agents and a coach agent [22].

Monitoring Algorithm Complexity As discussed above, agents cannot continuously communicate with a monitoring agent about their state. Thus in general, the monitoring agent has uncertainty about the state of monitored agents, i.e., the the monitoring agent entertains multiple hypotheses as to the state of each individual agent. To detect disagreements, the monitoring agent must compare the state of one agent to the state of another. Since there may be multiple hypotheses as to the state of each of the agents, the monitor must select possible combinations of the hypotheses of different agents, to serve as the basis for the decision on whether a disagreement has occurred. However, the number of combinations of individual hypotheses can grow exponentially in the number of agents. Thus it would seem that we would need to go through an exponential number of hypotheses just to pick those that are useful for monitoring purposes. Consider an example, borrowed from [14]:

Example 1. In the ModSAF domain, three helicopters are executing the WAIT-FOR-SCOUT plan, in which one of them (role: *scout*) is flying towards the enemy while its two teammates (role: *attackers*), have landed. Once the scout identifies the enemy, it radios back to the attackers, causing all three agents to switch to the JOIN-SCOUT plan, in which the scout lands, while the attackers fly forward to join it. Due to a radio equipment malfunction, one attacker failed to receive the message from the scout, causing it to continue waiting. Thus a state of disagreement occurs among the agents. Suppose this attacker is monitoring its teammates by observing their actions and inferring their currently executing plans: The scout has landed, and so may have started the JOIN-SCOUT plan, or may be responding to a command to land immediately (the HALT-ORDER plan). The other attacker, flying towards the scout, may be executing its role in the JOIN-SCOUT plan, or a completely different plan (the FLY plan). Although there are only two hypotheses for each individual monitored agent, there are *four* hypotheses as to the overall state of the team (Table 1), and as the size of the team grows, the number of hypotheses increases exponentially.

Monitoring Attacker	Other Attacker	Scout
WAIT-FOR-SCOUT	FLY	JOIN-SCOUT
WAIT-FOR-SCOUT	FLY	HALT-ORDER
WAIT-FOR-SCOUT	JOIN-SCOUT	JOIN-SCOUT
WAIT-FOR-SCOUT	JOIN-SCOUT	HALT-ORDER

Table 1: Hypotheses for state of team in Example 1.

Limited Connectivity As the number of agents grows, agents become more logically and physically distributed, and cannot maintain continuous contact with each other. We use the term *limited connectivity* in a general sense, to denote both limited ability to observe a particular agent's actions (e.g., because of occlusion or physical distribution), and limited communications (e.g., due to interference, range, or reliability issues). Consider the following example:

Example 2. In the RoboCup domain, the 11 soccer-playing agents switch between two high-level game plans, triggered by referee messages. The INTERRUPT plan is called in the beginning of the game, at half-time, etc., and requires players to place themselves in pre-determined home positions, standing still. The PLAY plan requires players to play soccer freely. Due to failures on the agents perception skills, some players sometimes fail to hear the referee, and so fail to switch plans. They are then in disagreement with their peers who did hear the referee. Under perfect connectivity, players would be able to communicate with their teammates, or see their teammates standing still, and thus detect disagreements. However, players have communication range and bandwidth limitations, and have a limited field of view. They can therefore not normally see or communicate with all their teammates. They therefore face difficulties in detecting disagreements.

Our own previous work [14] has shown that sometimes only certain *key agents* must be monitored to guarantee detection. And yet, every agent must monitor all of these key agents. Furthermore, in a worst case, all agents are key agents, and therefore would seem to require full connectivity. Indeed, this is the case in Example 2.

3 Disagreement Detection

Disagreement detection involves a key step of representing the state of monitored agents, such that the state of different agents can be compared to detect disagreements. This section briefly describes a general representation of monitored agents' states, and a basic inference algorithm which uses the representation for observation-based and communication-based monitoring. The representation, inference algorithm, and their use for disagreement detection have been discussed in detail in [14].

3.1 Representation and Inference

Much of contemporary theoretical and empirical work on teamwork (collaboration), both in synthetic agents and in humans, has emphasized agreement on a hierarchical

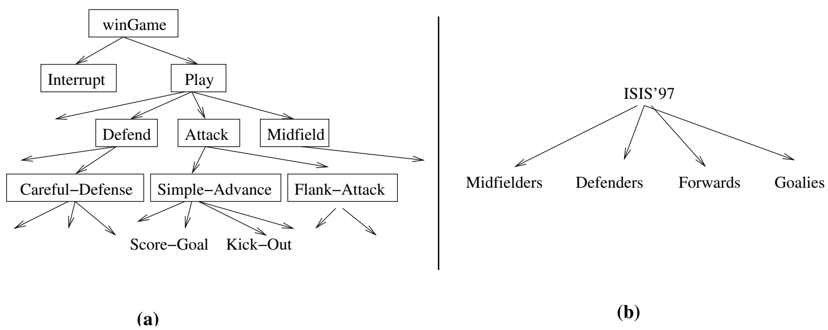


Figure 1: Plan-hierarchy (a) and team-hierarchy (b) in the RoboCup domain. Boxed plans denote team-plans, which must be agreed-upon by team-members.

recipe, or plan, as a key to effective teamwork, (see, for instance, [12, 8, 20]). Given this emphasis, we focus on a monitoring representation that follows two key constraints: (i) representing agents in terms of their currently executing plans (and plan-steps); (ii) allowing the designer, or monitoring agent to mark plans that have to be agreed upon, so that they are executed *jointly* (together) by all members of a team (or subteam). These two constraints give rise to two structures that are used by the monitoring system: A plan-decomposition hierarchy, and a team organization hierarchy. These have been fully described in [23, 13], and so we only provide a brief overview here.

A plan-hierarchy is used to represent a monitored agent's plan. It is defined to be a directed connected graph, where vertices are plan steps, and edges signify hierarchical decomposition of a plan into sub-plans. Each vertex has at most one parent (i.e., one incoming hierarchical decomposition edge); a plan that conceptually has many parents (i.e., it is a component in the decomposition of different parent plans) is represented by different vertices in the plan-hierarchy. Multiple outgoing edges signify alternatives available to the agent, of the first subplan to be executed. The graph forms a tree along hierarchical decomposition edges, so that no plan can have itself as a descendent. A vertex with no children edges denotes an atomic step.

For example, Figure 1-a presents a portion of the plan-hierarchy used to monitor the ISIS'97 RoboCup Simulation team [22]. The top-level plan, WINGAME, is selected by all players as soon as they join a game. It has one first child, the INTERRUPT plan, which is assumed to be selected by the agent whenever the game is interrupted by the referee. WINGAME's other child, PLAY, follows INTERRUPT in order of execution, and is selected when the game is currently playing. Thus INTERRUPT and PLAY follow each other to the end of the game. In service of PLAY, players choose a plan (ATTACK, DEFEND, etc.) based on their role in the team: forwards, defenders, etc. (discussed later). This decomposition continues. For instance, at a particular given moment, a forward may be monitored to be engaged in executing the following path (from root to leave): WINGAME — PLAY — ATTACK — SIMPLE-ADVANCE — SCORE-GOAL.

To make use of the representation, a monitoring agent must have a way of associating information it senses about another agent with paths in a plan-hierarchy used to represent the monitored agent's state. An algorithm for doing this, called RESL, has been previously described in [14] and is presented here briefly: The designer of the monitoring system associates with each plan a set of *observables*, condition monitors that tie in sensor readings and received communications with particular plans in the hierarchy. When a condition monitor matches the sensor reading (e.g., when a message is received that is consistent with the plan in action, or when an action associated with a plan is observed), we tag the plan *matching*. If its observables fail to match, the plan is tagged *not-matching*. RESL infers the state of unobservable plans from their children and parents: An otherwise untagged parent with at least one successfully-matched child is tagged successfully-matching, otherwise it is tagged as failing to match. And an untagged child with a successfully-tagged parent is tagged successfully-matching, unless all of its own children are tagged as failing to match. In this way, all plans in the hierarchy are tagged as matching or not-matching the observations. Multiple matching siblings denote multiple hypotheses. The process is linear-time in the size of the plan hierarchy.

To monitor multiple agents using the above representation, we construct a separate plan-hierarchy for each monitored agent. When a specific agent is observed, its state is updated in the hierarchy which represents it. This method has been successfully used in monitoring agents deployed in ModSAF [21], RoboCup [22], the civilian evacuation simulation, and enterprise scheduling [23]. In general, it can be useful in monitoring agents whose behavior is controlled by a hierarchical process, e.g., hierarchical behaviors [2], Soar [17], or HAC [1]. Its generality is derived from its use in monitoring rather than execution and control. It avoids any details which determine how the actual decisions of agents are made (e.g., in preferring one decomposition over another) since it is only used to organize and keep track of their decisions after the fact.

3.2 Detecting Disagreements

In order to detect disagreements, the monitoring agent must first know which plans are ideally to be agreed upon. Plans in the hierarchy must be marked as *team plans* [12, 20]. We assume that team plans are marked by the designer, or by the monitoring agent, for instance based on executable teamwork models such as STEAM [20] or GRATE* [12]). Different subteams can execute different team plans. In our experiments, team plans were known since the monitored agents had actually used STEAM, and therefore used team plans explicitly. In Figure 1-a, team plans are boxed: WINGAME, PLAY, and INTERRUPT are to be executed by the all members of the RoboCup team ISIS'97. MIDFIELD, DEFEND, etc. are to be executed jointly only by members of the corresponding subteams of ISIS'97 (*midfielders*, *defenders*, etc.).

The monitoring agent, after hypothesizing the state of agents as previously discussed, matches team plans across members of teams—if agents are in agreement about their selected team plans, then all is well. If agents are not in agreement about their team plans, then a disagreement is announced. Note that agents do not have to be in agreement about all plans—only about those plans that are marked as team plans. Furthermore, the plan-hierarchies used for different agents may themselves be different (other than in the team plans), facilitating monitoring of behaviorally-heterogeneous agents.

As previously discussed there can be in general multiple hypotheses for each individual, and an exponential number of hypotheses for the team as a whole. To evaluate hypotheses, previous work has formally defined the *coherence* of a multi-agent hypothesis as the ratio of the number of agents to the number of different selected team plans [14]. Intuitively, coherence is a measure of the Agreement in a team. For example, the hypotheses in Table 1, Rows 1, 2, & 4 have coherence level of 1. The maximally-coherent hypothesis (Row 3) has coherence level 1.5 (3/2), since unlike the other hypotheses, it has at least two agents in agreement. Optimal coherence has all N agents in agreement, and has value N.

It was shown that as long as the monitoring process is complete (see formal definition in Section 5), the agent is guaranteed that its detection results are going to be *sound*, by selecting *maximally-coherent* hypotheses: Any detection is going to be of a real failure [14, Theorem 1], but not all failures are guaranteed to be detected. Soundness is very important to a monitoring system, since it prevents false alarms which would otherwise task the agent needlessly. However, a difficulty emerges in applying this result in monitoring an increasing number of agents. While only a single maximally-coherent hypothesis is needed, the process of finding such an hypothesis potentially still requires *going through all exponential number of hypotheses* to rank them based on coherence. Formally, the space required for monitoring multiple agents using an array of plan-hierarchies (as described above and in previous work) is $O(NL)$, where N is the number of agents, and L is the size of the plan-hierarchy. However, sorting through the hypotheses can take $O(L^N)$, as previously demonstrated (Example 1).

4 Scaling Up Monitoring

To address the challenges raised by the time and space complexity of previously known techniques, we present YOYO, an algorithm that utilizes knowledge about the team organization to carry out disagreement detection in *linear time*. The intuition behind YOYO is to represent only coherent hypotheses (of which there is a linear, not exponential, number), and then recognize disagreements as cases where the representation fails. YOYO is based upon earlier work on the YOYO* visualization algorithm [13], but differs from it in several important ways (discussed in depth in Section 6).

The key idea in YOYO is to represent all agents together, in *a single shared plan hierarchy*. The shared plan-hierarchy is fully expanded to contain the plans and transitions for all subteams, annotated so that YOYO can determine which subteam is to take which transitions, execute which plans, etc. A plan P in this hierarchy, when tagged as successfully matching, represents the hypothesis that all agents in the team associated with P are executing P. Observations about agents are then matched against the shared plan-hierarchy. Intuitively, the process of detection proceeds as follows: If some team members are executing P, while others are executing a different plan Q, and assuming the observations allow us to differentiate P, Q then both will be marked matching and not-matching at the same time, and we will know that a disagreement has occurred.

However, members of different subteams execute different plans by design. Therefore, YOYO needs to differentiate cases where members of the same team have selected different plans P, Q, and cases where members of different teams have selected P, Q. To do this, YOYO exploits knowledge of the social structure within the monitoring system, in the form of a designer-specified *team-hierarchy*, a tree-like structure which encodes knowledge about the relationships between teams, subteams, and team-members: Each node in the team-hierarchy corresponds to a monitored organizational unit. The top (root) team represents the entire monitored team. These teams are then split into several subteams, etc., until the leaves of the hierarchy contain roles of individual agents, if such different roles exist. For instance, Figure 1-b presents the team-hierarchy of the ISIS'97 RoboCup team [22], composed of a root node for the entire team, and four nodes for its four subteams.

The team-hierarchy contains pointers from each node to plan nodes in the shared plan-hierarchy. The plans pointed to are the hypothesized coherent plans of the monitored team, and thus multiple pointers are allowed from a single team-hierarchy node. The pointers in the team-hierarchy point at the lowest-level plans that are consistent with the observations, and are to be executed by the team in question. For example, suppose all RoboCup players are executing the PLAY plan together, and that members of each subteam are in agreement with their teammates on the plan chosen for the subteam. A player that observes its teammates using the team- and plan-hierarchies in Figure 1 will have pointers from the ISIS'97 node in the team hierarchy (Figure 1-b) to the PLAY plan in the plan-hierarchy (Figure 1-a). Each of the subteam nodes in the team-hierarchy will have pointers to plans in the plan-hierarchy which are executed by the different subteams. For instance, the *Forwards* subteam may have a pointer to the SIMPLE-ADVANCE plan, signifying the all members of the subteam are executing this plan.

YOYO (Algorithm 1) maintains the pointers such that the hypotheses they represent are coherent with each other at all times. If it fails, then this means that the team's state is unambiguously incoherent, i.e., a disagreement exists. YOYO operates as follows: When an observation is made about an agent (called the *focus*), we not only update the pointers for this agent, but also re-align the pointers of its parent (sub)teams, such that their own pointers point at plans that are coherent with the new hypothesized state of the focus. We then go up and down the team-hierarchy to re-align the pointers of the other agents which are either part of the focus' subteam or its siblings' subteam. This is done by moving the pointers of non-focus agents (and the subteams of which they are members) such that they point at a plan that is coherent with the plans hypothesized for the focus. If the initial set of pointers for any non-focus agent is already coherent, no re-alignment is necessary. If no plan can be found for them, or if all plans for a team are tagged both matching and not-matching at the same time, then a state of disagreement has been detected.

For example, suppose that the players are known to be executing the INTERRUPT plan (Figure 1). Suppose now that a *defender* observes a *forward* player running towards the opponent goal (i.e., executing the ATTACK plan, in service of PLAY). The defender tags Interrupt as matching, and Play as not matching (based on its own selected plans). However, YOYO (executed by the monitoring defender) will create pointers for observed attacker to point to the ATTACK plan (lines 1–8), and will tag as not-matching other plans, in particular INTERRUPT. It will then enter the loop on line 9. For the ATTACK plan, it will propagate its successful tag up to the PLAY plan (line 12), and the create a pointer from the root node in the team hierarchy, representing the ISIS'97 team (the root in Figure 1-b) to PLAY, since the team that executes PLAY is the ISIS'97 team (the parent of the the *Forwards* subteam). It will then climb up in both hierarchies (lines

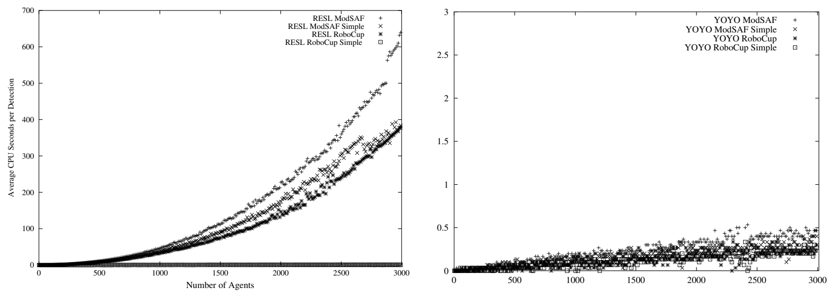
Algorithm 1 YOYO(**plan hierarchy** L, **team-hierarchy** H)

1: **for all** observations O_i at time t **do**
2:   **for all** plans X that have conditionals testing O_i **do**
3:     let A_i be the agent observed in O_i
4:     **if** X matches O_i and X executable by A_i **then**
5:       tag X as successfully matching
6:       create pointer from A_i node (in H) to X
7:     **else**
8:       tag X as not matching
9: **for all** plans X tagged matching or not matching **do**
10:   $T \leftarrow team(X)$, $P \leftarrow X$
11:   **while** $parent(P) \neq null$ **do**
12:     propagate any tags to $parent(P)$ (match or not match)
13:     **if** $team(parent(P)) = parent_{team}(T)$ **then**
14:       propgate tags down to untagged plans
15:       create pointer from $parent_{team}(T)$ to $parent(P)$
16:       $T \leftarrow parent_{team}(T)$
17:     $P \leftarrow parent(P)$
18: **for all** teams T in H **do**
19:   **if** T only points to plans matching and not matching **then**
20:     a disagreement has occurred.

16 and 17) and begin another iteration. Later on, the same process will be repeated for the Interrupt plan. Since both Interrupt and Play are pointed to from the node representing the team ISIS'97, a disagreement will be detected.

YOYO requires a slightly modified inference procedure than previously described (Section 3). First, in following children transitions, YOYO is careful to only take paths legal to the team in question (i.e., plans and transitions that the team is allowed to execute in its role): It thus makes the assumption that transitions in the plan hierarchy are marked for the subteams that are allowed to take them. Second, YOYO must use a time-stamp to tag plans, so that observations that arrive simultaneously (but processed serially) will cause a detection of disagreements (if one exists), instead of overwriting the effects of each other. Per the example above, if the monitoring defender observes another forward to be executing Interrupt, while the first forward is executing Play, then the inference process for the two observations would tag these plans as both matching and not-matching at the same time, and a disagreement would be detected.

Evaluation YOYO's first part matches plans against all observations (lines 1–8), and thus takes $O(NL)$, where N is the number of agents (observations), and L the size of the plan-hierarchy. The nested loops potentially traverse the entire plan-hierarchy $O(L)$ for each team. Since the team-hierarchy grows with N, we use that to denote its size; a traversal of the team-hierarchy is $O(N)$. The propagation down in line 14 may still traverse the entire $O(L)$ plan-hierarchy (in a theoretical worst case). The process thus takes $O(NL^2)$. Finally, the disagreement detection goes from every team in the team-hierarchy to every plan (in the worst case), thus $O(LN)$ again. Overall, YOYO's complexity is $O(NL^2)$. The key to this complexity is that YOYO only maintains coherent hypotheses. If it cannot, then a disagreement has occurred—but YOYO does



(a) RESL (b) YOYO

Figure 2: Comparing YOYO to RESL. The X axis marks the number of agents monitored; the Y axis denotes computation time in CPU seconds. *Note (a) and (b) have different ranges on the Y axis.*

not represent the underlying incoherent hypothesis. This time complexity should be contrasted with RESL's $O(L^N)$. YOYO's space complexity also compares well with RESL: With each additional agent, YOYO's space requirements grow by one node which represents the agent in H. In contrast, RESL uses an additional copy of the entire plan hierarchy for every additional agent.

To provide empirical evaluation, the run-time performance of RESL and YOYO was compared as the number of monitored agents is scaled up. Trials were carried out in 4 different domains: ModSAF, RoboCup (both previously described), ModSAF-simple (uses smaller plan-hierarchy than ModSAF), and RoboCup-simple. RoboCup-simple involved no uncertainty in monitoring: It is not intended to be realistic, but allows exploration of the performance boundaries of RESL and YOYO. In each trial, the number of monitored agents was fixed, and then a monitoring problem (given by the observables available to the monitoring agent) was randomly generated. The same monitoring problem was given as input to RESL and to YOYO, and their execution time (matching observations, inference, and disagreement detection) was recorded. 30 trials were done for each fixed number of agents.

Figure 2 summarizes the results of the comparison, projecting average computation time (per trial) against the number of agents. Figure 2-a shows that the computation time for RESL in the ModSAF, ModSAF-simple, and RoboCup domain is clearly non-linear as predicted. A monitoring problem of 3000 agents takes, on average, 400–650 CPU seconds in these domains. However, in the RoboCup-simple domain the curve is linear since there is no uncertainty in the domain. As a result, the number of hypotheses to be considered does not grow exponentially, and the only factor in run-time complexity is observation matching and inference.

In contrast to RESL, YOYO's complexity curve (Figure 2-b) is roughly linear (due to the randomness of the monitoring problems, YOYO solves some problems quicker than others of the same size, thus the averages do not form a perfect linear curve), and its execution time is measured in *milliseconds*, rather than seconds. Note that even when monitoring 3000 agents, YOYO takes less than half a second to complete its task (compared to 400–650 seconds).

The difference in performance is not just a function of YOYO's maintenance of a single structure (rather than than the N structures maintained by RESL), but the fact that it considers only coherent hypotheses, of which only a linear number (in the size of the plan hierarchy) exist. However, YOYO's scalability comes at the expense of the ability to represent failure hypotheses: When a disagreement is detected, YOYO knows that it has occurred, but cannot identify what agents are involved, or the extent of the disagreement. Thus for more advanced monitoring tasks, such as diagnosis [14], YOYO has to be augmented by mechanisms that allow the monitoring agent to reconstruct the hypotheses underlying the disagreement. In contrast, RESL facilitates explicit reasoning about each agent separately from its teammates. It is also the case that YOYO's run-time complexity is still dominated by a key factor—the number of observed monitored agents: As long as simultaneous observations are coming in about agents, YOYO still needs to process all of them, much like the full array approach. We address ways to reduce the number of agents which require observations in the next section.

5 Limited Connectivity

As the number of monitored team-members increases, it becomes increasingly difficult to monitor all of them (Section 2). Furthermore, as we have seen, the run-time of monitoring algorithms can be dominated by the number of monitored agents, even with efficient algorithms such as YOYO. Thus a key question is how to guarantee detection results while limiting the number of agents that must be monitored. This section provide new bounds on the number of agents that must be monitored to guarantee disagreement detection.

We begin with a short overview of *key agents*, first introduced in [14] for their important role in disagreement detection. We show that centralized monitoring requires monitoring only (but all) key agents to guarantee sound detection. We also re-examine the role of key agents in distributed monitoring settings. Previous work has shown that sound and complete detection can take place if all agents monitor all key agents. We provide a stricter bound and show that non-key agents must monitor only a single key agent. However, key agents must still monitor each other.

Key Agents for Disagreement Detection We abstract the underlying monitoring algorithm, such as YOYO, by providing the following notation when discussing agent A's hypotheses as to the state of an agent B: Suppose B's state is P (for instance, P is a plan selected by B). We denote by $M(A, B/P)$ the set of agent-monitoring hypotheses that A constructs based on communications from B, or inference based on B's observable behavior. In other words, $M(A, B/P)$ is the set of all A's hypotheses as to B's state, when B's state (e.g., selected plan) is P. Note that when A monitors itself, it has direct access to its own state and so $M(A, A/P) = \{P\}$.

We make the following definitions which ground our assumptions about the underlying monitoring process that implements M:

Definition 3. Given a monitoring agent A, and a monitored agent B, we say that A's monitoring of B is *complete* if for any plan P that may be executed by B, $P \in M(A, B/P)$. If A is monitoring a team of agents $B_1, \cdots, B_n$, we say that A's *team-monitoring* of the team is *complete* if A's monitoring of each of $B_1, \cdots, B_n$ is complete.

Monitoring completeness is commonly assumed (in its individual form) in plan-recognition work, (e.g., [19, 6, 11]), and generally holds in our own applications. It means that the set $M(A, B/P)$ includes the correct hypothesis P, but will typically include other matching hypotheses besides P. Using this notation, we can now formally explore the role of key agents in disagreement detection.

Key agents have the property that their behavior when executing two given plans is sufficiently unambiguous, such that any agent that monitors them and is executing either one of the two plans can identify with certainty whether a disagreement exists between it and the key agents. Thus these agents play an important role in limiting the number of agents that must be observed to guarantee disagreement detection. We repeat here the formal definition of key agents from [14].

Definition 4. Let P_1, P_2 be two team plans. Suppose an agent A is monitoring an agent B. If $M(A, B/P_1) \cap M(A, B/P_2) = \emptyset$ for any agent A, we say that B has observably-different roles in P_1 and P_2, and call B a *key agent* for P_1, P_2. We assume symmetry so that if two plans are not observably different, then $M(A, B/P_1) \cap M(A, B/P_2) \supseteq \{P_1, P_2\}$.

For instance, both attackers and the scout have observably-different roles in the plans executed in Example 1: The attackers land in WAIT-FOR-SCOUT, but fly in JOIN-SCOUT. The scout flies in WAIT-FOR-SCOUT, but lands in JOIN-SCOUT. Note that observably-different behavior does not imply complete disambiguation. For instance, if the scout is observed to be flying, that does not allow the hypothesis that it is executing JOIN-SCOUT, but permits two other hypotheses: WAIT-FOR-SCOUT and FLY.

The key-agent is the basis for the conditions under which a team-member A_1 will detect a disagreement with a team-member A_2 using a maximal-coherence hypothesis. A_1 (executing a team-plan P_1) will detect a disagreement with a team-member A_2 (executing different team-plan P_2) if A_2 is a key agent for the plans P_1, P_2 [14, Lemma 1]. A_1 knows that it is executing P_1. If A_2 is executing P_2, and is a key-agent in P_1 and P_2, then A_1 is guaranteed to notice that a disagreement exists between itself and A_2, since A_2 is acting observably different than it would if it had been executing P_1. A_1 can now alert its teammate, diagnose the failure, etc.

Bounding the Number of Observed Agents. As the previous section demonstrates, agents who wish to detect disagreement must focus their attention on the key agents in a team. Indeed, in centralized monitoring settings, a team-member which is monitoring itself and its teammates using maximal-coherence can focus its attention only on key-agents, since it cannot expect to detect disagreements with non-key agents, as their behavior is ambiguous. Such monitoring is guaranteed to be sound for centralized settings [14, Theorem 1]. However, in such centralized settings, all key agents must be monitored.

We now consider the case of distributed monitoring settings, where team-members monitor each other. Previous work has shown that if at least a single key agent exists for every pair of plans (i.e., the team employs an *observably-partitioned set of team plans*), and if all team-members monitor *all* key agents, then detection is not only sound, but also complete [14, Theorem 4]: At least one team-member will detect a disagreement if one occurs, and no false detections will take place. This result is of particular interest to building practical robust teams, and fortunately the conditions for it are often easy to satisfy: Teams are very often composed such that not all agents have the same role in the same plan, and in general, roles do have observable differences between them (e.g., Examples 1 & 2). Often, the set $M(A, B/P)$ can be computed offline, in advance; this allows the designer to identify the key agents in a team prior to deployment. Furthermore, any agent can become a key-agent simply by communicating its state to the monitoring agent and therefore eliminating ambiguity; thus a team can use highly-focused communications to guarantee detection.

However, the requirement that *all* key-agents be monitored prohibits deployment of scaled-up applications: First, as the size of the team grows, limited connectivity becomes more common, since agents become more physically and logically distributed. Thus not all agents, and in particular key agents, are going to be visible. Second, the run-time of monitoring, even using the efficient YOYO algorithm presented earlier, is dominated by the need to process observations of each agent. Thus reducing the number of observed agents can improve monitoring run-time in practice.

To formally address this challenge, we define the monitoring graph of a team as follows:

Definition 5. A *monitoring graph* of a team T is a directed (possibly cyclic) graph in which nodes correspond to team-members of T, and edges correspond to monitoring conditions: If an agent A is able to monitor an agent B (either visually or by communicating with it), then an edge (A, B) exists in the graph. We say that the monitoring graph is connected, if its underlying undirected graph is connected.

If the monitoring graph of a team is not connected, then there is an agent which is not monitored by any agent, and is not monitoring any agent. Obviously, a disagreement can go undetected in such a team: If the isolated agent chooses a plan different from its peers, it would go undetected. However, the fact that a monitoring graph is connected does not guarantee detection, since an agent that is monitoring a non-key agent would not necessarily detect a disagreement with the non-key agent.

Previous work has shown that if all agents monitor all key agents, then sound and complete detection is guaranteed [14]. However, in the case of Examples 1 & 2, this translates to an unrealistic requirement that all key agents are monitored at all times, by everyone in the team. This is difficult to guarantee in practice, for instance due communication range restrictions or limited ability to observe all key agents. Certainly, for the case of Example 2, our experiments shown that on average, a player can see at most 2–3 team-members at any given time.

The theorem below takes a step towards addressing this issue by providing more relaxed conditions on the connected nature of the monitoring graph, in particular with respect to the connectiveness of the nodes representing key agents. These conditions are: (i) every non-key agent executing a plan P monitors *a single key agent* for each possible pair of plans involving P (i.e., for each pair of plans, where one of the plans is P); and (ii) the monitoring graph for all key agents in a given pair of plans is a clique (i.e., key agents are fully connected between themselves).

Theorem 6. *Let T be a team in which: (i) Each team-member A, executing a plan P_1, who is not a key agent for P_1, P_2 monitors a key agent for P_1, P_2; (ii) all key agents for a pair of plans X, Z monitor all other key agents for X, Z (forming a bidirectional clique in the underlying monitoring graph); (iii) the team employs an observably-partitioned set of plans; and (iv) all monitoring carried out is complete, and uses maximal-coherence. Then disagreement detection in T is sound and complete.*

Proof. By induction on the number of agents in T. The full proof stretches over a number of pages, and is therefore presented in the appendix. □

This theorem allows teams to overcome significant connectivity limitations, without sacrificing detection quality. The theorem translates into significant freedom for the designer or the agents in choosing whom (if any) to monitor; when a monitored agent is unobservable, an agent may choose to monitor another: Non-key agents need monitor only a single key agent, rather than all key agents (for every pair of plans). The upper-bound the theorem provides is more general than may seem at first glance. First, while we refer to *plans* in the theorem, there is in fact nothing in the result that uses the representation we discuss in the previous section. The theorem holds for any state feature of interest—beliefs about a shared environment, goals, etc.; it is up to the designer to pick a monitoring technique that acquires the needed information for constructing the monitoring hypotheses. Second, the theorem does not depend at all on the method by which monitoring occurs, whether by communications or by observations. Thus the connectivity of a monitoring graph does not have to be maintained visually. Some or all of the edges in the monitoring graph may actually correspond to communication links between agents.

Though this theorem represents a significant advance in lowering the bound on the number of agents that must be monitored, all key agents must still monitor each other. This is a critical constraint in practice: We have reconstructed the visual monitoring graph in *thousands* of RoboCup game situations, to find that even with this new bound, sound and complete disagreement detection would have been possible without communications only in small percentage (approximately 5%). Typically, each RoboCup player can only see 2–3 key agents. To illustrate, Figure 3 shows the monitoring graph of two teams overlayed on a screen-shot of an actual game situation. For both teams, the monitoring graph does not guarantee sound and complete disagreement detection under the known bound.

This empirical constraint raises the bar on the challenge to find a lower-bound on the number of agents that must be monitored to guarantee detection. We have strong evidence that leads us to believe that in fact a much lower bound than that which is described above may be possible, but as of now, it remains unproven. We believe that it may be possible to guarantee sound and complete detection in all cases where each key agent is either monitored or is monitoring a single other key agent (rather than all of them). If so, this would translate to guaranteeing failure detection in over 70% of the

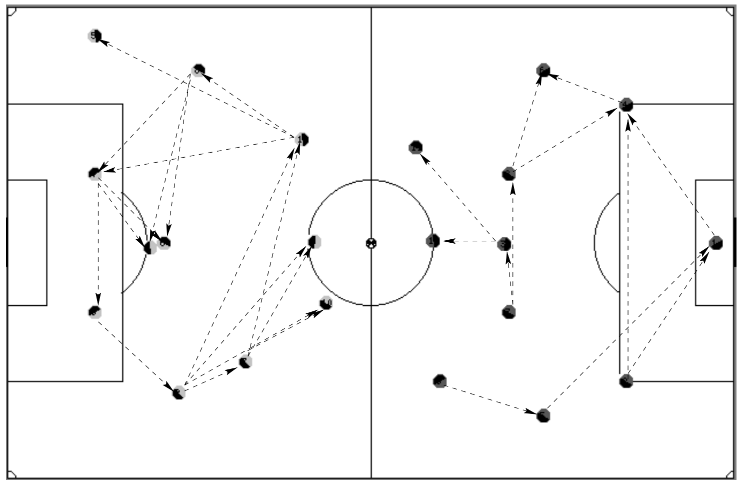


Figure 3: Monitoring graphs in actual game situation.

thousands of RoboCup monitoring cases we have examined. We present this formally as a conjecture.

Conjecture 7. *Let T be a team in which: (i) Each team-member A executing a plan P_1, who is not a key-agent in P_1, P_2 (where $P_1 \neq P_2$) monitors a key agent in P_1, P_2; (ii) every key-agent for a pair of plans X, Z monitors or is monitored by one other key-agent in X, Z (if more than one exists); (iii) the team employs an observably-partitioned set of plans; and (iv) all monitoring carried out is complete, and uses maximal-coherence. Then disagreement detection in T is sound and complete.*

6 Related Work

Most closely related to YOYO is our own earlier work on the YOYO* visualization algorithm [13], deployed in different domains. YOYO differs from its predecessor in several important ways: (i) YOYO utilizes a non-probabilistic representation scheme, while YOYO* utilizes a probabilistic representation; (ii) YOYO is focused on disagreement detection, and in fact does quite poorly in visualization tasks on which YOYO* performs well; and (iii) YOYO includes explicit checks for failure detection. Also, while YOYO* has revealed a potential tradeoff between expressivity and scalability in visualization algorithms, YOYO provides the same detection results as the full array approach.

$RESC_{team}$ [19] is a multi-agent plan-recognition scheme which implicitly uses coherence as a key constraint in representation. $RESC_{team}$ represents only a single coherent hypotheses, while YOYO represents all coherent hypotheses. However, $RESC_{team}$ can reason about the assignment of agents to roles/subteams, while YOYO assumes this knowledge is given a-priori. Intille and Bobick [11] rely entirely on coordination constraints among football players to recognize team-tactics. Devaney and Ram [6] use pattern-matching to recognize team-tactics in military maneuvers. Huber

[10] proposes using automatically built plan-recognition Bayesian networks to allow agents to coordinate without communications. In contrast to our work, all of these approaches utilize probabilistic representations, do not detect failures, and have not explicitly addressed scalability.

Durfee [7] discussed decision-theoretic and heuristic methods for reducing the amount of knowledge that agents consider in coordinating. The methods include pruning nested (recursive) models, using communications to alleviate uncertainty, using hierarchies and abstractions, etc. Our work complements Durfee's in several way: (i) we focus on monitoring in teams of cooperating (rather than self-interested) agents, allowing exploitation of socially-attentive means; (ii) we provide bounds on the number of agents that must be monitored; (iii) Durfee's work focuses on reducing computational loads in monitoring each single agent, while our work on YOYO demonstrates significant savings that are achieved when considering all monitored agents together.

Dellarocas and Klein [5] present complementary failure detection techniques for contract-net protocol interactions, using a centralized monitoring scheme that monitors all agents using pre-built fault-models. While their techniques complement ours, they do not address limited connectivity, or scalability concerns.

7 Discussion and Future Work

Multi-agent literature has often emphasized that an agent must monitor other agents in order to carry out its tasks. However, as the numbers of agents in deployed teams is scaled up, two key challenges are raised: (i) the number of hypotheses a monitoring agent has of its peers increases exponentially in the number of agents, and thus large teams cannot be monitored effectively; and (ii) agents become more physically and logically separated, and so are less visible (less connected) to the monitoring agent. Thus not all agents can be monitored.

This paper has begun to address these challenges, in the context of a critical monitoring task—detection of critical disagreements between teammates. We have presented YOYO, a linear-time sound disagreement detection algorithm, exploiting the organizational structure of the monitored team to monitor any number of agents in a single structure. We addressed the challenge of limited connectivity of agents by showing new bounds on the number of agents that must be monitored in a team to guarantee sound and complete disagreement detection in distributed and centralized monitoring settings. This new result in the distributed case shows that in fact high-quality detection can be guaranteed despite situations where many agents do not monitor others, or are not monitored by others. Our empirical and analytical evaluation demonstrates that the results indeed constitute significant advancement to the practice of monitoring large teams.

Acknowledgement. We thank Milind Tambe, Jeff Rickel, George Bekey, Victor Lesser, David Pynadath, Orna Raz and Dan Peleg for useful discussions. As always, thanks to K. Ushi.

References

[1] Marc S. Atkin, Gary W. King, and David L. Westbrook. Hierarchical agent control: A framework for defining agent behavior. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, 2001.

[2] Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, 1998.

[3] John J. Burns, Eduardo Salas, and Janis A. Cannon-Bowers. Team training, mental models, and the team model trainer. In *Advancements in Integrated Delivery Technologies*, Denver, CO, 1993.

[4] Philip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 35, 1991.

[5] Chrysanthos Dellarocas and Mark Klein. An experimental evaluation of domain-independent fault-handling services in open multi-agent systems. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-00)*, pages 95–102, Boston, MA, 2000. IEEE Computer Society.

[6] Mark Devaney and Ashwin Ram. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 942–947, Madison, WI, 1998.

[7] Edmund H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pages 406–413, 1995.

[8] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.

[9] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *distributed computing*, 37(3):549–587, 1990.

[10] Marcus J. Huber and Edmund H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pages 163–170, 1995.

[11] Stephen S. Intille and Aaron F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 518–525. AAAI Press, July 1999.

[12] Nicholas R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[13] Gal A. Kaminka, David V. Pynadath, and Milind Tambe. Monitoring deployed agent teams. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, 2001.

[14] Gal A. Kaminka and Milind Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[15] Sanjeev Kumar, Philip R. Cohen, and Hector J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-00)*, pages 159–166, Boston, MA, 2000. IEEE Computer Society.

[16] Neal Lesh, Charles Rich, and Candace L. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modelling (UM-99)*, Banff, Canada, 1999.

[17] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

[18] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[19] Milind Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.

[20] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[21] Milind Tambe, W. Lewis Johnson, Randy Jones, Frank Koss, John E. Laird, Paul S. Rosenbloom, and Karl Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

[22] Milind Tambe, Gal A. Kaminka, Stacy C. Marsella, Ion Muslea, and Taylor Raines. Two fielded teams and two experts: A robocup challenge response from the trenches. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 1, pages 276–281, August 1999.

[23] Milind Tambe, David V. Pynadath, Nicholas Chauvat, Abhimanyu Das, and Gal A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-00)*, pages 301–308, Boston, MA, 2000.

A Proofs

This appendix presents the full proof of Theorem 6. This proof relies on a lemma and a couple of theorems from [14], and they are therefore reprinted here to provide the reader with the necessary context.

Theorem 8. *[14, Theorem 1] Let a monitoring agent A monitor a simple team T. If A's team-modeling of T is complete, and A uses a maximally team-coherent hypothesis for detection, then the teamwork failure detection results are sound.*

Proof. We will show that if no failure has occurred, none will be detected, and thus that any failure that is detected is in fact a failure. Let $a_1, \ldots, a_n$ be the agent members of T. Each agent a_i is executing some plan P_i $(1 \leq i \leq n)$. Thus collectively, the group is executing $(P_1, \ldots, P_n)$. If no failure has occurred, then all the agents are executing the same plan P_0, i.e., $\forall i, P_i = P_0$. Since A's team-modeling is complete, the correct hypothesis $(P_0, \ldots, P_0)$ is going to be in the set of team-modeling hypotheses H. Since it is a maximally team-coherent hypothesis, either it will be selected, or that a different hypothesis *of the same coherence level* will be selected. Any hypothesis with the same coherence level as the correct one implies no failure is detected. Thus the detection procedure is sound. □

Lemma 9. *[14, Lemma 1] Suppose an agent A_1 is monitoring a teammate A_2 using the maximally team-coherent heuristic (and under the assumption that team-modeling is complete). A_1, executing P_1, would detect a disagreement with an agent A_2 executing a different plan P_2, if A_2 has an observably different role in P_1 and P_2 (i.e., A_2 is a key agent in P_1, P_2).*

Proof. A_1 knows that it is executing P_1, and is monitoring A_2, who has an observably different role in P_1 and P_2. Since A_2 is executing P_2, and following the observably different role, $P_1 \notin M(A_1, A_2/P_2)$. Therefore from the perspective of A_1, it cannot be the case that it assigns P_1 in any *agent-modeling* hypothesis, and therefore any *team-modeling* hypothesis that A_1 has will have A_1 executing P_1, and A_2 executing some plan other than P_1. In other words, from A_1's perspective there is no team-coherent hypothesis, and so a difference would be detected between A_1 and A_2. □

Lemma 10. *If an agent K is not key agent for P, Q, and it is not a key agent for P, R, then it is not a key agent for Q, R.*

Proof. K is not a key agent for P, Q. Therefore, it is not observably different in P, Q. Therefore it is observably the same in P, Q. Therefore

$$M(A, K/P) \cap M(A, K/Q) \supseteq \{P, Q\}$$

By the same reasoning, since K is not a key agent in P, R,

$$M(A, K/P) \cap M(A, K/R) \supseteq \{P, R\}$$

Therefore,

$$M(A, K/Q) \cap M(A, K/R) \supseteq \{P\} \neq \phi$$

Thus K is not a key agent in Q, R. □

Based on Theorem 8 and Lemma 9, we can now proceed with the main theorem.

Theorem. *(Number 6, Page 14). Let T be a team in which: (i) Each team-member A, executing a plan P_1, who is not a key agent for P_1, P_2 monitors a key agent for P_1, P_2; (ii) all key agents for a pair of plans X, Z monitor all other key agents for X, Z (forming a bidirectional clique in the underlying monitoring graph); (iii) the team employs an observably-partitioned set of plans; and (iv) all monitoring carried out is complete, and uses maximal-coherence. Then disagreement detection in T is sound and complete.*

Lemma 11. *If every agent t there exist some pair of plans X, Z such that is t a key agent for X, Z, then Theorem 6 holds.*

Proof. First, since all monitoring is complete and is done using maximal-coherence, we know monitoring results are sound (Theorem 8). We will show that the monitoring results are complete. We consider the monitoring graph G_T of the team T, and partition it into k partitions, such that each partition holds the vertices corresponding to agents executing the same plan. We will show that necessarily, $k = 1$, i.e., all agents are executing the same plan.

Assume for contradiction that $k > 1$. Let us pick any partition, and mark it after the plan selected by the agents in it, X. Since G_T is connected, the partitions for a connected graph, though not necessarily all partitions are connected to all others. Therefore, X must be connected to partitions $Q_1, \ldots, Q_{m>=1}$. We will first show that $\forall i, 1 \leq i \leq m, \forall x \in X$, x is not a key agent for X, Q_i.

Assume for contradiction that there exists an agent $x_0 \in X$ which is key for X, Q_i, and no disagreement is detected by any agent. Pick an agent $q \in Q_i$. There are two cases:

- **q is a key agent for X, Q_i.** In this case x_o is monitoring q (because all key agents for X, Q_i monitor each other), and would detect a disagreement with q. Contradiction.

- **q is not a key agent for X, Q_i.** Therefore, it must be monitoring a key agent r for X, Q_i. Because all key agents for X, Q_i monitor each other, x_o is also monitoring r. Since no disagreement is detected, r cannot be executing Q_i or X, and must therefore be executing some other plan $y \neq X \neq Q_i$, such that r must not be a key agent for y, X, nor is it be a key agent for y, Q_i. But based on Lemma 10, it follows that r is not a key agent for X, Q_i. Contradiction.

Therefore, $\forall i, 1 \leq i \leq m, \forall x \in X$, x is not a key agent for X, Q_i. Pick an agent $x_1 \in X$. x_1 cannot be a key agent for any pair of plans X, Q_i, and yet it must be a key agent for some pair Z, Y. It must be that $\exists i, j,$ s.$t.$ $Z = Q_i$, and $Y = Q_j$, since the partitions for Z, Y must be connected to X. Therefore, x_1 is a key agent for Q_i, Q_j. But x_1 is not a key agent for X, Q_i, and is not a key agent for X, Q_j, and therefore (Lemman 10), it is not a key agent for Q_i, Q_j. Contradiction.

Since in all possible cases the assumption $k > 1$ leads to contradiction, necessarily $k = 1$, that is all agents are in agreement. Thus it cannot be the case that two or more agents are in disagreement, and none detects a failure. Therefore monitoring is complete, and since it must be sound (see beginning of proof), the theorem holds if every agent is a key agent. □

With Lemma 11 in place, we can now prove Theorem 6.

Proof. We will first show disagreement detection completeness by induction on the number of agents N. The idea here is to show that if any two agents A_1, A_2 have selected two different plans P_1, P_2, where $P_1 \neq P_2$, then a member of the team T will detect the failure. In otherwords, to show completeness we need to show that if a disagreement occurs, it will be detected.

Induction base: Obviously if there is only one agent no disagreement can occur, so we begin with the case of two agents, A_1, A_2, who have selected plans P_1, P_2 respectively, where $P_1 \neq P_2$, and are therefore in disagreement. We know that at least a single key agent exists for P_1, P_2, because the team employs an observably-partitioned set of plans. Without loss of generality, assume the key agent is A_2. Then A_1 is monitoring it, and since A_2 is key agent in P_1, P_2 then A_1 will detect the disagreement (Lemma 9).

Induction hypothesis: Assume the theorem holds for a team with up to $N-1$ agents. We will show that it holds for a team with N agents. There are two cases:

- **Case 1: T has an agent t which is non-key for all pairs of plans X, Y.** We examine G_N, the directed monitoring graph of T (see Definition 5). Since the monitoring graphs of all key agents are connected, and since all non-key agents are monitoring key agents, it follows that G_N is connected. We examine the incoming and outgoing edges of the vertex representing t. Since G_N is connected, t has incoming monitoring edges (t is monitored by other agents), or t has outgoing monitoring edges (t monitors other agents), or both.

  Let us now remove t from the graph. Since t is a non-key agent in all pairs of plans X, Y, it follows that removing it results in a reduced graph G_{N-1} which satisfies the conditions of the theorem, for a reduced team of only $N-1$ agents: (i) All other non-key agents are continuing to monitor key agents (we have not modified monitoring edges from these other non-key agents to key-agents); (ii) all key agents continue to monitor all other key agents; (iii) the team still employs an observably-partitioned set of plans—since the removal of t did not change the set of key agents nor the set of plans; and obviously (iv) monitoring is still complete and uses maximal coherence). We are now left with a team of $N-1$ agents.

  If A_1, A_2 are within the $N-1$ agents left, then the disagreement would be detected (based on the induction hypothesis), and so we are done. If no disagreement is detected, then it follows from the induction hypothesis that no disagreement exists among the $N-1$ agents, i.e., one of A_1, A_2 is included within the $N-1$ agents, and the other is t. Without loss of generality, assume $t = A_2$. Then A_1, executing P_1 is one of the $N-1$ agents in the reduced team, and since they are not in disagreement with A_1, they must all be executing P_1. Let us now re-create the original graph G_N, reintroducing t into the team by putting back the original incoming and outgoing edges. Since $t = A_2$, it is P_2. And since it not a key-agent, it must be monitoring a key-agent for P_1, P_2. However, this key agent must be executing P_1. Therefore, t would have detected a disagreement (Lemma 9).

- **Case 2: T does not have an agent that is non-key agent for all pairs of plan.** Thus every agent is a key agent in some pair of plans X, Y. Then the theorm holds for N agents based on Lemma 11.

In all possible cases, a failure is detected if a disagreement exists, thus failure detection is complete. Since monitoring is complete, failure detection is also sound (Theorem 8). □