Detecting Disagreements in Large-Scale Multi-Agent Teams

Gal A. Kaminka*
The MAVERICK Group
Computer Science Department
Bar Ilan University, Israel
galk@cs.biu.ac.il

October 1, 2008

Abstract

Intermittent sensory, actuation and communication failures may cause agents to fail in maintaining their commitments to others. Thus to collaborate robustly, agents must monitor others to detect coordination failures. Previous work on monitoring has focused mainly on small-scale systems, with only a limited number of agents. However, as the number of monitored agents is scaled up, two issues are raised that challenge previous work. First, agents become physically and logically disconnected from their peers, and thus their ability to monitor each other is reduced. Second, the number of possible coordination failures grows exponentially, with all potential interactions. Thus previous techniques that sift through all possible failure hypotheses cannot be used in large-scale teams. This paper tackles these challenges in the context of detecting disagreements among team-members, a monitoring task that is of particular importance to robust teamwork. First, we present new bounds on the number of agents that must be monitored in a team to guarantee disagreement detection. These bounds significantly reduce the connectivity requirements of the monitoring task in the distributed case. Second, we present YOYO, a highly scalable disagreement-detection algorithm which guarantees sound detection. YOYO's run-time scales linearly in the number of monitored agents, despite the exponential number of hypotheses. It compactly represents all valid hypotheses in single structure, while allowing for a complex hierarchical organizational structure to be considered in the monitoring. Both YOYO and the new bounds are explored analytically and empirically in monitoring problems involving thousands of agents.

*This research was supported in part by Israel Science Foundation grants #1211/04, and #1357/07.

1 Introduction

Agents in realistic environments sometimes fail to maintain their commitments to others. This can occur due to sensor and actuator uncertainties, or (possibly intermittent) communication failures. It may also occur due to the nature of dynamic, complex domains, which can challenge the agent's design in unanticipated environment states, e.g., in industrial systems (e.g., [13]), and virtual environments (e.g., [24, 29, 30, 35]).

Agents must therefore monitor others to ascertain that their commitments are maintained, and to detect coordination failures when they occur (see e.g., [2, 5, 6, 12]; additional works are discussed in Section 2). Indeed, a number of investigations have explored mechanisms for detecting (and responding to) failures in coordination and teamwork [3, 8, 20–22, 27, 37].

Large-scale multi-agent systems—where the number of agents is the principal scale factor—pose a number of challenges to the existing monitoring techniques. First, agents in large-scale systems become physically and logically separated, and thus less able to directly monitor each other (the *limited connectivity* challenge). However, existing approaches often rely on being able to monitor all agents, either by communications or observations. Second, the number of possible failures grows combinatorially in the number of agents, with all possible interactions. Thus approaches that search through failure hypotheses do not scale well. We discuss these challenges and previous work in detail in Section 2.

This paper addresses these challenges in depth, in the context of detecting *disagreements*, a principal failure in multi-agent teamwork. Theoretical and empirical research on teamwork in humans (e.g, [1]) and in synthetic agents [2, 6, 13, 23, 35] stresses agreement as a cornerstone to effective teamwork (although different terms are used 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 make two contributions. First, we tackle the challenge of limited connectivity by providing new bounds on the 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 [21], in a distributed fashion. However, in practice, limited connectivity restricts 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* (i.e., no false positives) and *complete* (no false negatives) detection can be guaranteed in practice even if other agents monitor just one key agent; however, all key-agents must still monitor each other. In addition, we show that monitoring only key agents is an also a sufficient condition in the centralized monitoring case, where a single agent is monitoring all others. Such monitoring is guaranteed to be sound, and detect any disagreement that would have been detected had the centralized monitoring agent monitored all others. Using the techniques presented, a monitoring agent can detect failures in large teams, involving thousands of agents. The assumptions underlying the bounds can often be met in practice, simply by allowing an agent to become a key agent

by broadcasting its state to its peers. In such real-world setting, the bounds can serve to focus such broadcasts and reduce them to a minimum.

Second, 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 highly scalable structure, and efficiently detect situations in which the agents are in a state of disagreement. YOYO can be used to provide sound disagreement detection capabilities. It is an example of a *Socially-Attentive* monitoring algorithm, exploiting knowledge of the social relationships in the monitored team. We present an empirical evaluation of YOYO in monitoring problems involving thousands of agents.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents new bounds on the number of agents that must be monitored. Section 4 presents the YOYO algorithm. Section 5 presents the results from experiments in using YOYO. Section 6 concludes with a discussion of the applicability of the presented contributions, and future directions for this work.

2 Motivation and Background

We use the term *limited connectivity* in a general sense to describe the phenomenon where an agent cannot observe, sense, or communicate with its peers, due to processing and bandwidth limitations. Limited connectivity is only of little concern in small-scale systems. Given a few cycles, agents can typically integrate multiple perceptions, over time, to assess what their peers are up to. However, as the number of agents grows, the ability to integrate such information over time diminishes rapidly [38]. For instance, existing peer-to-peer (P2P) systems include millions of active nodes. Yet not one node is able to communicate directly with all of its peers at once, due to both bandwidth and processing power issues. Even spreading the efforts over time will not be sufficient in practice.

Limited connectivity adversely affects the ability of an agent to monitor its peers and to detect coordination failures. Because of limited connectivity, the monitoring agent is not able to correctly assess the state of its teammates, and thus will necessarily face some uncertainty as to their state, and by implication, as to the existence of a coordination failure. Yet few bounds and few techniques are known for monitoring with limited connectivity.

Most closely related to monitoring with limited connectivity is our own previous work on centralized and distributed coordination failure detection. In [20], we introduced the notion of *key agents*, whose observable behavior is sufficiently unambiguous to an observer such that they can be used to detect failures even under conditions of uncertainty. The same work also showed that in the distributed case, if all agents monitor each other and there are sufficient key agents, failure detection will be guaranteed [20]. Later, the result was extended to show that in fact only the key agents had to be monitored in the distributed case,

thus allowing for reduced connectivity [21]. Our work in this paper lowers this upper-bound further (see Section 3). However, our work here is specific to disagreements.

A second important challenge with large-scale multi-agent systems is raised by the number of monitoring hypotheses that must be processed. As a multi-agent system grows in the number of agents, so does the the number of potential coordination failures it may contain. Suppose each of N agents may be in one of k internal states. Then the number of possible joint states is k^N. In loosely-coupled systems, each agent is essentially independent of its peers, and may select between its k possible states freely. In such systems, the vast majority of joint states—if not all—are considered valid states. However, in a coordinated multi-agent system, the selection of an internal state by an agent is dependent on the selections of its peers. In other words, agents move between joint states together; Only a limited portion of the space of joint states would be valid, from the designer's perspective. Thus most joint states may in fact be invalid from a coordination point of view.

Agreement is a good example of such tightly-coupled coordination. Teamwork literature emphasizes the importance of team-members being in agreement on various features of their state, such as goals, plans, and beliefs[1] [2, 6, 7, 13, 18, 23, 35]. Since the object of the agreement is irrelevant for our purposes in this paper, we will use the term *state* to denote the internal state-feature of the agent which is the object of the agreement (e.g., a belief in a proposition p, a plan p, an intention, etc.). Suppose a team of N agents agrees that their selection of internal state would be synchronous, i.e., for every selected state of one agent, all others must be in some agreed-upon internal state. There would be $O(k)$ valid agreement joint states, and the rest of the k^N joint states would be considered invalid—*coordination failure*—states. Thus large-scale systems where agents coordinate face a large exponential number of possible faults, and only a limited set (by comparison) of valid states.

Despite the much greater number of possible failure states, many of the approaches proposed in the past for coordination failure detection rely on enumerating possible faults. Klein and Dellarocas [3,22] have proposed a centralized approach to detecting failures (which they refer to as *exceptions*). Their work utilizes agent sentinels, which communicate with the agents in the system to identify their state or actions, and report on it to a centralized fault detection system. This fault detection system then matches the reported information against coordination fault-models, for detection. These are generated offline, by manual analysis of domain-independent coordination models. The fault models and their use do not address limited connectivity, ambiguities in agent states, nor failures in the sentinel system. However, they are demonstrated over a larger range of failures (this paper only covers disagreements).

More recently Platon et al. [25, 26] have systematically and methodically examined different types of agent failures (exceptions), as well as resulting co-

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

ordination failures and their causes (e.g., those due to agent death, knowledge inconsistencies, etc.). They propose a number of ways for integrating failure handling capabilities into the agent architecture. We do not investigate these issues in this paper.

A different—distributed—approach is taken by Horling et al. [8, 9]. They present an integrated failure-detection and diagnosis system for a multi-agent system in the context of an intelligent home environment. The system uses the TAEMS domain-independent multi-agent task-decomposition and modeling language to describe the ideal behavior of each agent. The agents are also supplied with additional information about the expected behavior of the environment they inhabit under different conditions, and their role within the multi-agent organization. A distributed diagnosis system, made of diagnosis agents that use fault-models, is used to identify failures in components (such as erroneous repeated requests for resources) and inefficiencies (such as over- or under-coordination). Multiple diagnosis agents may use communications to inform each other of their actions and diagnoses. The fault-models are used in planning monitoring actions, in identifying failures responsible for multiple symptoms, and in guiding recovery actions. Like similar works above, this work did not address connectivity concerns.

A key issue with fault-model approaches is their scalability, given that the number of possible faults in large-scale multi-agent systems is likely to be exponential. Some have addressed this by focusing on general failure conditions. As an example, Wilkins, Lee, and Berry [37] offer an execution monitoring approach which encompasses both coordination and task-execution failures. Their work introduces a taxonomy of generic failure types, which must be adapted and specialized to the domain and task. Agents responsible for monitoring rely on communicated state reports from the monitored agents to identify failures. While experiments with the system were carried out only on relatively small multi-agent systems, the modeling of the failures shows example of how fault-models can be sufficiently non-specific so that they may be reused in larger-scale systems. For instance, the fault models included distance failures (units getting too close), which are triggered when an adversary gets closer to a friendly unit). It does not matter who the adversary or friendly units are, nor their specific location, etc. The use of such general fault-models, however, diminishes from the ability to detect complex or specific failures.

A common theme running through the fault-model approaches above is that they utilize communications or direct observations to acquire knowledge as to the state of monitored agents, and typically require knowledge of all agents, thus ignoring limited connectivity. This is a potentially limiting factor in their use in large-scale networks, where limited connectivity will necessarily lead to uncertainty in monitoring. Moreover, in many domains, even direct monitoring of another agent may involve some uncertainty. In particular, a monitoring agent may entertain several hypotheses as to the true internal state of another agent. However, these approaches often ignore such uncertainty.

In contrast to the fault-model approaches discussed above, we advocate a model-based approach, in which a model of the correct behavior of the agent

is used to detect failures, by noting discrepancies between ideal and actual behavior. Earlier work taking this approach to detect disagreements [20, 21] contrasts with the work in this paper in terms of the run-time complexity of searching through the hypotheses space to determine if a coordination failure occurred. Both earlier investigations relied on a plan recognition algorithm (RESL) which modeled each individual agent separately. While computing the individual hypotheses using RESL can be done in time $\mathcal{O}(NL)$, where L is the size of the state-space of a single agent, and N the number of agents, extracting the hypotheses can take exponential time $\mathcal{O}(L^N)$. However, RESL can be used in principle to detect many kinds of coordination failures, and allows either sound or complete disagreement detection (in the centralized case). In contrast, YOYO, presented in this paper, runs in time $\mathcal{O}(N+L)$, but supports only sound detection in the centralized case.

Poutakidis et al. [27] have utilized Petri-net representation of interaction protocols to centrally detect interaction failures, where agents fail to follow the protocol in their conversation with others. This work does not address limited connectivity, in that it assumes all messages from all agents are observable. Although Poutakidis et al. allow for multiple monitoring hypotheses to co-exist, they do not provide a method for selecting hypotheses such that soundness or completeness is guaranteed. In contrast, our work in this paper addresses both centralized and distributed monitoring settings, addresses limited connectivity, and provides guarantees on the detection results.

$RESC_{team}$ [34] is a multi-agent plan-recognition scheme which implicitly uses coherence as a key constraint in representation. $RESC_{team}$ represents only a single coherent hypothesis, 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.

YOYO is a variant of YOYO*, a probabilistic team plan-recognition algorithm, used for overhearing [19]. In contrast to YOYO*, YOYO is symbolic, has better run-time complexity, and targets detection. However, it fails at tasks in which the previous algorithm can excel (e.g., in overhearing).

Once a failure is detected, it needs to be diagnosed and resolved. Kalech and Kaminka [15] have addressed model-based diagnosis of coordination failures. Roos et al. [31, 32] have addressed model-based diagnosis of non-coordination failures in multi-agent systems. Horling et al. [8] use the fault-detecting causal model for diagnosis and subsequent recovery actions. Recently, there is also work on diagnosis in large-scale systems [14].

3 Monitoring Graphs and Limited Connectivity

A key question is how to guarantee failure-handling results while limiting the number of agents that must be monitored. The approach we take to this involves the construction and analysis of monitoring graphs, which represent information about which agent can monitor whom. We show that for disagreement detection, one can set conditions on the structure of the graph which, when

satisfied, guarantee that detection is *complete* and/or *sound*, under conditions of uncertainty. Complete detection guarantees all failures will be detected (i.e., no false negatives). Sound detection guarantees only failures will be detected (i.e., no false positives). We separate discussion of centralized and distributed monitoring settings.

We begin by formalizing the notion of a monitoring graph (Definition 1). We will use this construct throughout the paper.

Definition 1. 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 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 state different from what has been agreed upon with its peers, it would go undetected. Thus a connectivity lower-bound for detecting disagreements (indeed, any kind of coordination failure) is that the monitoring graph must be connected.

However, connectivity by itself is insufficient. Uncertainty can also have significant impact on the results of monitoring. When an agent A monitors an agent B, it may often entertain multiple hypotheses as to the state of 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 from B's observable behavior (i.e., via plan recognition). 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 use the shorthand $M(A, B)$ to denote the hypotheses set of B's currently selected state. In this work, we assume observer independence: $M(A, B) = M(C, B)$ for any agents A, C.

To see the impact of uncertainty, suppose an agent A has selected state P_1, and is monitoring another agent B that has selected state P_2. A disagreement exists here since agent B should have selected P_1. However, since the internal state of B may not be known to A with certainty, A may have several interpretations of B's chosen state. The set of these interpretations may contain P_1, in which case A may come to incorrectly believe that B is also executing P_1, and that therefore no disagreement has occurred. Indeed, if the set of hypotheses includes both P_1 and P_2, then A may or may not detect the failure, depending on the choice it makes.

This problem is exacerbated when monitoring a team containing multiple agents. A team-monitoring hypotheses set for a given team T with n is the

cross-product of the individual hypotheses sets:

$$M(A,T) = M(A,a_1) \times M(A,A_2) \times \ldots \times M(A,a_n)$$

Suppose A is monitoring itself (as a member of the team, executing P), and agents B, C. If $M(A,B/P) = \{P,Q\}$ and $M(A,C/R) = \{P,R\}$, then four monitoring hypotheses exist overall for the team T containing agents A, B, C:

$$M(A,T) = \{(P,P,P),(P,P,R),(P,Q,P),(P,Q,R)\}$$

One hypothesis implies no failure exists. Others differ in how many disagreements there are.

3.1 Centralized Disagreement Detection

In general, as discussed above, the condition of monitoring graph connectivity is necessary, but insufficient, to guarantee failure detection results. The challenge is to find an upper-bound, a sufficient condition on the connectivity of the monitoring graph, which would provide a method for systematically choosing hypotheses such that a guarantee exists for the results.

Kaminka and Tambe have shown that it is possible to use a ranking heuristic, *maximum coherence*, to select hypotheses such that it is possible to guarantee certain aspects of the monitoring results [21]. Informally, the coherence value of an hypothesis is a measure of the number of disagreements it implies. Formally, coherence is defined in [21] as the ratio of the number of agents to the number of different states in the team-monitoring hypothesis. Thus (P,P,P) has coherence of

$$\frac{|\{A,B,C\}|}{|\{P\}|} = \frac{3}{1} = 3$$

while (P,Q,P) has coherence of

$$\frac{|\{A,B,C\}|}{|\{P,Q\}|} = \frac{3}{2} = 1.5$$

and (P,Q,R) has coherence of

$$\frac{|\{A,B,C\}|}{|\{P,Q,R\}|} = \frac{3}{3} = 1$$

Systematic selection of hypotheses which have maximum coherence is guaranteed to result in sound detection [21, Theorem 1]: If a maximum-coherence hypothesis indicates a failure, then a failure has indeed occurred. However, some failures may go unnoticed if, due to uncertainty, a maximum-coherence hypothesis exists which indicates no disagreement. In contrast, selection of a coherence-minimizing hypothesis is guaranteed to provide complete detection, where no failure will go unnoticed (but there may be false detections). Unfortunately, no coherence-based heuristic exists that guarantees both sound and complete detection in the centralized monitoring case [21, Theorem 3].

To provide this guarantee the hypothesis set $M(A, B/P)$ for given agents A, B must be *complete*, as defined below (Definition 2). Monitoring completeness is commonly assumed (in its individual form) in plan-recognition work, (e.g., [4,11,34]), and generally holds in many applications. It means that when A monitors B, the set $M(A, B/P)$ includes the correct hypothesis P, but will typically include other matching hypotheses besides P.

Definition 2 (Monitoring Completeness)**.** Given a monitoring agent A, and a monitored agent B, we say that A's monitoring of B is *complete* if for any state P that may be selected by B, $P \in M(A, B/P)$. If A is monitoring a team of agents $B_1, \ldots, B_n$, we say that A's *team-monitoring* hypotheses set $M(A, T)$ is *complete* if A's monitoring of each of $B_1, \ldots, B_n$ is complete.

Kaminka and Tambe show that if a single centralized monitoring agent monitors all others and monitoring is complete, it can guarantee either sound or complete detection of disagreements, but not both [21, Theorem 3]. They also found that if certain *key agents* exist, then it may be possible to reduce the monitoring requirements in the system.

Key agents have the property that their behavior, when they select one of two specific states (P_1, P_2), is sufficiently unambiguous, such that any agent that monitors them cannot confuse P_1 and P_2. In other words, key agents, when executing either P_1 or P_2, never have both P_1 *and* P_2 in the hypothesis set of any agent observing them.

As a result, any observer that is executing one of P_1, P_2 can identify with certainty whether a disagreement exists between it and the key agents. We repeat here the formal definition of key agents from [21]:

Definition 3 (Key Agents)**.** Let P_1, P_2 be two agent states. 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: (i) P_1, P_2 are *observably-different*; (ii) B is a *key agent* in $\{P_1, P_2\}$. We assume symmetry so that if two states are not observably different to A, then they are observably the same:

$$M(A, B/P_1) \cap M(A, B/P_2) \neq \emptyset \Rightarrow M(A, B/P_1) \cap M(A, B/P_2) \supseteq \{P_1, P_2\}.$$

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. This is done by preferring *maximally-coherent* hypotheses as to the state of the monitored agent. Maximally-coherent hypotheses are optimistic—they are hypotheses that minimize the number of disagreements between the two agents. The use of such hypotheses leads to sound disagreement detection [20,21, Theorem 1].

An agent A_1 (selecting state P_1) will detect a disagreement with a team-member A_2 (selecting a different state P_2) if A_2 is a key agent for the plans P_1, P_2 [21, Lemma 1]. A_1 knows that it has selected P_1. If A_2 has selected 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 selected P_1. A_1 can now alert its teammate, diagnose the failure, etc.

We will now show that when key agents exist in a team, it is sufficient for a single agent to monitor them to guarantee sound disagreement detection in the centralized case. More accurately, any disagreement that the agent would have been detected when monitoring all agents (and itself)—as previous work [21] suggests—would be detected if the agent monitors only key agents (and itself).

Theorem 1. *Given a team T (of which some members are key agents), and a single agent $A \in T$, if A monitors only the key agents of T and itself, such that (i) monitoring is complete; and (ii) hypotheses are selected based on maximal coherence, then A would detect any disagreement that would have been detected had it monitored all agents.*

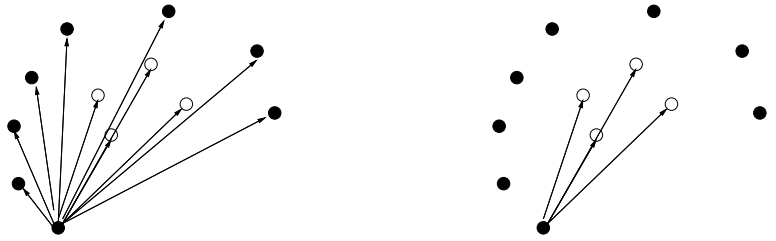
Proof. We will show that whenever A detects a disagreement when monitoring *all* agents, it will detect the same disagreement when monitoring only the key agents in T. There are two cases. In the first case, agent A detects a disagreement between itself and another agent. In the second case, A detects a disagreement between two other agents.

Case 1. Suppose that A has selected P, and has detected a disagreement with another agent B (executing a different plan Q). Assume for contradiction that B is *not* a key agent in P, Q. Under previous work, A would have monitored B (since it would have monitored all agents). Because A is using maximal coherence, the only condition underwhich it would detect a disagreement is if $P \notin M(A, B/Q)$. But this means that $M(A, B/Q) \cap M(A, B/P) = \emptyset$, because of the symmetry assumption in the definition of observably-different plans. Thus, B is in fact a key-agent in P, Q, contradicting the assumption in this case.

Case 2. Suppose that A has selected plan P, and is monitoring two other agents B, C, who have selected plans Q, Z, respectively. Assume for contradiction that A has detected a disagreement between B, and C, but not with itself (otherwise, it would have been handled as in Case 1 above). Thus $M(A, B/Q) \cap M(A, C/Z) \supseteq \{P\}$. This contradicts the assumption that A detected a disagreement between B and C, since under maximal coherence, A would have selected P as an individual hypothesis for both. A would have therefore detect no disagreement between B and C, contradicting the given condition that a disagreement was detected. Thus this case is impossible. □

The intuition for this proof is as follows: If A has detected a disagreement with B, then A's model of B did not include P (A's plan). Because of symmetry, we assume that if two plans are not observably different, then they are observably the same. In other words, if P, Q are not observably-different (by B), then $M(A, B/Q) \supseteq \{P, Q\}$, and A would therefore have not been able to detect the disagreement.

The assumption of symmetry in the definition of observably-different plans is a strong assumption, and critical to the proof. With it, two plans are either observably-different, or are observably the same when executed by the agent. Without it, it would have been possible for A to detect a disagreement with B even if B is not a key agent: For instance, if $M(A, B/P) = \{P, R\}$, $M(A, B/Q) = \{Q, R\}$ then B is not a key agent in P, Q, and yet A



(a) Centralized monitoring, sound or complete, but not both, in [21].

(b) Centralized monitoring, sound, in this paper.

Figure 1: Illustration of centralized monitoring graphs. Non-filled dots indicate key agents.

would have been able to detect a disagreement with it if A selects P and B selects Q. This raises new questions as to upper bounds when this assumption is removed, but we leave those for future investigation. Note also, that if all agents are key-agents, then the centralized monitoring agent will end up monitoring all team-members even with the new bound.

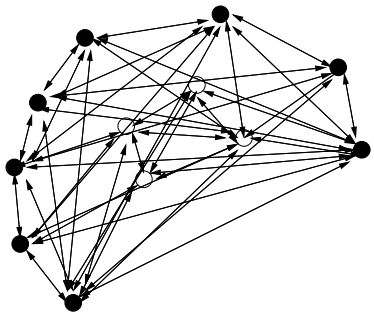
Figure 1 illustrates the significance of this new upper bound for centralized monitoring. Figure 1-a shows an agent monitoring all others. Figure 1-b shows the agent monitoring only key agents. Under the new upper-bound shown above, the agent is still guaranteed to detect all failures it would have detected (using maximal coherence) when monitoring all agents.

3.2 Distributed Disagreement Detection

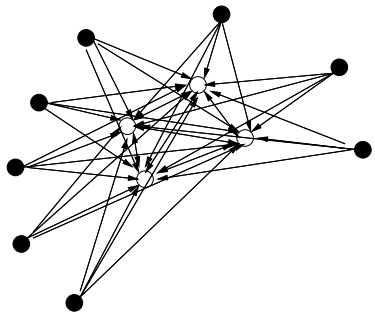
We now consider the case of distributed monitoring settings, where team-members monitor each other. It is easy to see that if the graph is connected, and each agent knows *exactly* the selection of its monitored peer, then sound and complete detection is possible, in a distributed fashion, with very limited connectivity: Each agent A monitors at least one other agent B (or is monitored by another agent B). If A selects an internal state different from B, then at least one of them would detect the disagreement immediately. If A monitors B—and knows with certainty B's state—then a simple comparison with A's selected state is all that is needed. Sound and complete detection means that at least one team-members will detect a disagreement if one occurs, and no false detections will take place.

A challenge is raised when the state of agents is not known to their monitors with certainty. This occurs necessarily under conditions of limited connectivity: Since monitors cannot perceive, sense, or communicate with all the monitored agents, and do not have shared memory access to the monitored agents, they necessarily have some uncertainty about their actions.

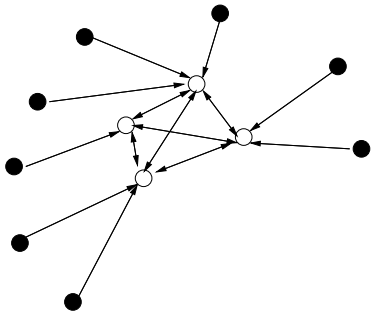
Fortunately, it is possible to show that under some conditions, having all



(a) Distributed monitoring, in [20].



(b) Distributed monitoring, in [21].



(c) Distributed monitoring, in this paper.

Figure 2: Illustration of distributed monitoring graphs. Non-filled dots indicate key agents. All cases allow for sound and complete disagreement detection.

agents take part in monitoring (distributed monitoring) allows for complete and sound detection of coordination failures (in particular, disagreements). Intuitively, this happens when there is always a key agent to be found, for any selected state. This condition is defined below:

Definition 4 (Observably-partitioned state-space)**.** A state-space P is said to be *observably-partitioned* if for any two states $P_i, P_j \in P$ there exists a key-agent A_{ij}. The set of these A_{ij} agents is called the *key agents set of* P.

Kaminka and Tambe [20] have shown that if at least a single key agent exists for every pair of team plans (i.e., the team employs an *observably-partitioned state-space*), and if all team-members monitor *all* agents, then detection is not only sound, but also complete (see Figure 2-a for illustration). Later on [21, Theorem 4], the result was improved somewhat: All agents must monitor the key agents only—all of them—and the key agents must monitor each other (Figure 2-b).

The condition of an observably-partitioned state-space is often easy to satisfy. For instance, 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. Often, in fact, 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. We leave further exploration of such dynamic creation of key-agents to future work.

However, the requirement that *all* key-agents be monitored inhibits deployment of scaled-up applications. 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 not all key agents, will be accessible for monitoring.

The theorem below takes an additional step by providing more relaxed conditions on the connected nature of the monitoring graph, in particular with respect to the connectivity of the nodes representing key agents. These conditions are: (i) every non-key agent selecting a state P_0 monitors *a single* key agent for each possible pair of states involving P_0 (i.e., for each pair of states, where one of the states is P_0); and (ii) the monitoring sub-graph for all key agents for a given pair of states forms a clique (i.e., key agents are fully connected between themselves). This case is illustrated in Figure 2-c.

Theorem 2 (Clique Key-Agent Monitoring)**.** *Let T be a team, employing an observably-partitioned state-space P, where: (i) Each team-member $A \in T$, selecting a state P_1, who is not a key agent for P_1, P_2 monitors* one *key agent for P_1, P_2; (ii) all key agents for a pair of states X, Z monitor all other key agents for X, Z (forming a bidirectional clique in the underlying monitoring graph); and (iii) all monitoring carried out is complete, and uses maximal-coherence. Then disagreement detection in T is sound and complete.*

To prove this theorem, we utilize two lemmas. The first has been presented and proved in [21, Lemma 1], and we repeat it here for clarity:

Lemma 1 (Lemma 1, [21])**.** *Let A_1, A_2 be agents who are monitoring each other using the maximal coherence heuristic. Suppose A_1, A_2 are executing P_1, P_2, respectively, where $P_1 \neq P_2$. Then A_1 would detect a disagreement with A_2 if A_2 is a key agent in P_1, P_2.*

Proof. See [21]. □

The second lemma (Lemma 2) is a weaker version of the theorem. Here, all agents in team T are key agents (each, for at least one pair of states). Under the other conditions of the theorem, we show that disagreement detection is sound and complete.

Lemma 2. *Let T be a team of agents, employing an observably-partitioned state-space P. If every agent $t \in T$ is a key agent for some $p_1, p_2 \in P$, then Theorem 2 holds.*

In other words, assume: (i) Each team-member $A \in T$, selecting a state $p_1 \in P$, who is not *a key agent for $p_1, p_2 \in P$ monitors* one *key agent for p_1, p_2; (ii) all key agents for any pair $p_i, p_j \in P$ monitor all other key agents for p_i, p_j (forming a bidirectional clique in the underlying monitoring graph); (iii) the team utilizes an observably-partitioned state-space P; (iv) all monitoring carried out is complete, and uses maximal-coherence; and (v) Every agent $t \in T$ is a key agent (there exist some pair of states $p_i, p_j \in P$ such that t is a key agent for p_i, p_j. Then disagreement detection in T is sound and complete.*

Proof. First, since all monitoring is complete and is done using maximal-coherence, we know monitoring results are sound [21, Theorem 1]. We will show that the monitoring results are complete. To do this, we show that if a disagreement exists, it would be detected.

Assume for contradiction that a disagreement exists, and that it was not detected by any agent. 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 selecting the same state. The assumption (for contradiction) that a disagreement exists means that $k \geq 2$. Without loss of generality, let us arbitrarily denote the states in these partitions $p_1, \ldots, p_k$, and name the partitions $P_1, \ldots, P_k$, respectively.

Let us pick any partition, and arbitrarily denote it X, and the state selected in it x. Since G_T is connected, the partitions form a connected graph, though not necessarily all partitions are connected to all others. Therefore, X must be connected to a set of partitions $Q_1, \ldots, Q_m$, where $q_i \in P_1, \ldots P_k$, and $1 \leq m < k$. We denote the states of the partitions $Q_1, \ldots, Q_m$ by $q_1, \ldots, q_m$, respectively.

We will first show that any agent $a \in X$ (any agent selecting state x) is not a key agent for x, q_i, $1 \leq i \leq m$. To see this, assume for contradiction that a is key for x, q_i. Pick an arbitrary agent $b \in Q_i$. There are two cases:

b is a key agent for x, q_i, just like a. In this case a is monitoring b (because all key agents for x, q_i monitor each other), and would detect a disagreement with b (Lemma 1 above). Contradiction.

b is not a key agent for x, q_i. Therefore, it must be monitoring a key agent r for x, q_i (as required in the conditions of the lemma). Because all key agents for x, q_i monitor each other, a is also monitoring r. Since no disagreement is detected, r could not have selected state q_i nor x, and must have therefore selected a different state y, where $y \neq x, q_i$. Now,

- r cannot be a key agent for y, x, or otherwise a would have detected a disagreement. Thus $M(a, r/x) \cap M(a, r/y) \supseteq y, x$.
- r cannot be a key agent for y, q_i, or otherwise q would have detected a disagreement. This implies that $M(b, r/q_i) \cap M(b, r/y) \supseteq y, q_i$.

But then, based on observer-independence, it follows that $M(a, r/x) \cap M(a, r/q_i) = y \neq \emptyset$. This means that r is not a key agent for x, q_i. Contradiction.

The above leads the the conclusion that a cannot be a key agent for x, q_i,$1 \leq i \leq m$.

However, a condition of the lemma is that all agents in T are key-agents. Thus a must be a key agent for some pair of states $z, w \in P$. It must be that $\exists i, j$, s.t. $z = q_i$, and $w = q_j$, since the partitions for Z, W (Q_i, Q_j) must be connected to X. Therefore, a is a key agent for q_i, q_j. But a is not a key agent for x, q_i, and is not a key agent for x, q_j, as we have seen above. Therefore:

- a cannot be a key agent for x, q_i. Thus $M(b, a/x) \cap M(b, a/q_i) \supseteq q_i, x$, for any observing agent b.
- a cannot be a key agent for x, q_j. Thus $M(b, a/x) \cap M(b, a/q_j) \supseteq q_j, x$, for any observing agent b.

But then, based on observer-independence, it follows that $M(b, a/q_j) \cap M(b, a/q_i) = x \neq \emptyset$. This means that a is not a key agent for q_i, q_j (z, w). Contradiction.

Since in all possible cases the assumption that a disagreement exists but was not detected leads to contradiction, necessarily all agents are in agreement, that is $k = 1$. 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 2 in place, we can now prove Theorem 2 by induction on the number of agents in T[2].

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 other words, 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 1).

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:

[2]The proof was developed jointly with Michael Bowling [17].

- **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 1). 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 executing P_2. And since it is 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 1).

- **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 theorem holds for N agents based on Lemma 2.

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 ([21, Theorem 1]). □

This theorem allows teams to overcome significant connectivity limitations, without sacrificing detection quality. It 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 states).

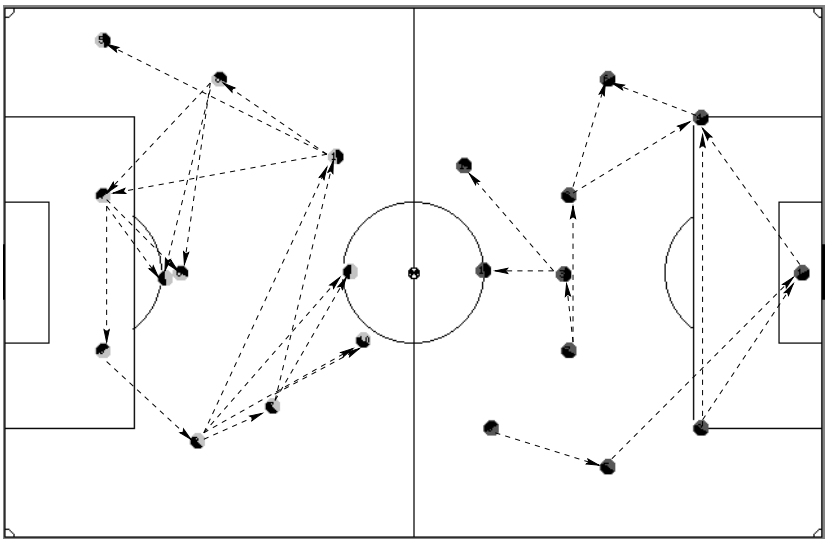


Figure 3: Monitoring graphs in a RoboCup simulation-league game situation.

The upper-bound the theorem provides is more general than may seem at first glance. First, 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 (possibly unreliable) 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. For instance, 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%) of a game. Typically, each RoboCup player can only see 2–3 key agents, this means that key agents cannot typically monitor all others. To illustrate, Figure 3 shows the monitoring graph of two teams overlaid on a screen-shot of an actual game situation. For both teams, the bound presented in this paper does not apply to the monitoring graph, thus sound and complete disagreement detection is not guaranteed. This empiric constraint raises the bar on the challenge to find a lower bound on the number of agents that must be monitored to guarantee detection.

It remains an open question whether a lower bound than that which is de-

scribed above may be possible. 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), i.e., when the monitoring graph forms a spanning tree. This would equate the upper and lower bounds on connectivity. In practice, it would translate to guaranteeing failure detection in over 70% of the thousands of RoboCup monitoring cases we have examined. Below, we present this formally as a conjecture.

Conjecture 1 (Spanning-Tree Key-Agent Monitoring)**.** *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.*

By allowing for limited connectivity, the agent can reduce the number of its monitoring targets. However, for each one of those agents that it does monitor, it must keep track of (possibly multiple) hypothesized states. These are used to form joint-state hypotheses, from which the maximal coherent hypotheses are selected. The next section details an efficient mechanism for the generation and selection of such hypotheses.

4 Efficient Disagreement Detection

Disagreement detection involves a key step of representing the state of monitored agents, such that the state of different agents can be contrasted to detect disagreements. This section presents a method for doing this process efficiently. Section 4.1 describes a general hierarchical representation of monitored agents' states, and a basic inference algorithm which uses the representation for observation-based and communication-based monitoring. Section 4.2 then presents YOYO, a novel algorithm for highly-scalable disagreement detection.

4.1 Representation

Much of contemporary theoretical and empirical work on teamwork (collaboration), both in synthetic agents and in humans, has emphasized agreement on a hierarchical recipe, or plan, as a key to effective teamwork (see, for instance, [6, 7, 13, 28, 35]). 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 (recipe), and a team decomposition

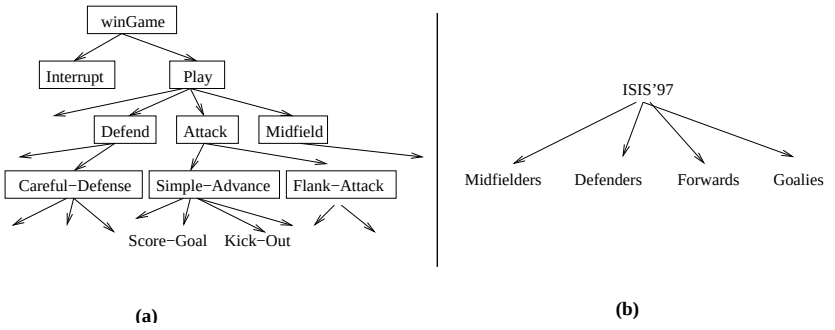


Figure 4: Plan-hierarchy (a) and team-hierarchy (b) in the RoboCup domain. Boxed plans denote (sub)team plans, which must be agreed-upon by (sub)team-members. Individual agents—each a subteam—are not shown.

hierarchy. These have been fully described in [19, 28], 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 as multiple instance 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 descendant. A vertex with no children edges denotes an atomic step.

For example, Figure 4-a presents a portion of the plan-hierarchy used to monitor the ISIS'97 RoboCup Simulation team [24]. 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 leaf): WINGAME — PLAY — ATTACK — SIMPLE-ADVANCE — SCORE-GOAL.

In order to detect disagreements, the monitoring agent must first know which plans are ideally to be agreed upon. We assume that such *team plans* are known, e.g., because they are marked in advance by the designer [13, 18, 28, 35]. In Figure 4-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.).

To maintain knowledge of which teams are assigned to which plans, there are pointers from each plan to nodes in a *team hierarchy* (Figure 4-b), used to track which sub-team is associated with each plan step (and vice versa). The team hierarchy is a tree 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 they exist. For instance, Figure 4-b presents the team-hierarchy of the ISIS'97 RoboCup team [24], composed of a root node for the entire team, and four nodes for its four subteams. Within subteams, members do not have different roles, and choose sub-plans for individual execution, with no social constraints on their selection (not shown in Figure 4-b).

4.2 Efficiently Detecting Disagreements

The monitoring agent uses the plan hierarchy to maintain its information on the state of the monitored agent. Such information can come from communications [3,19], e.g., where agents announce their initiation or termination of selected plan steps; or it may come from plan-recognition inference based on observations of the other agent's actions [10, 21]. An algorithm for such inference, called RESL, has been previously described in [21] 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 a plan, 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 matching, otherwise it is tagged as failing to match (not matching). And an untagged child with a successfully-tagged parent is tagged 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.

After hypothesizing the state of each agent based on received or observed information, the monitoring agent 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.

This method has been successfully used in monitoring agents deployed in ModSAF [36], RoboCup [24], and civilian evacuation simulation [28].

However, a difficulty emerges in applying this technique in monitoring a large number of agents. While only a single maximally-coherent hypothesis is needed, the total number of joint-state hypotheses combinatorially explodes in the number of agents. If we denote the size of the plan library by L and the number of agents by N, the number of joint-state hypotheses is $O(L^N)$ in the worst case. Moreover, the space requirements for reasoning also pose some difficulty, as an array of plan-libraries (one for each agent) requires $O(NL)$ space.

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 time and space *linear in the number of agents*. 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. As discussed in Section 2, YOYO is based upon earlier work on the YOYO* probabilistic plan-recognition algorithm [19], but differs from it in several important ways.

YOYO represents 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 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, as provided in the team-hierarchy described above.

YOYO maintains pointers from each node in the team hierarchy to plan-step 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 4 will have pointers from

the ISIS'97 node in the team hierarchy (Figure 4-b) to the PLAY plan in the plan-hierarchy (Figure 4-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 that 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.

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 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 ← team(X), P ← X
11:   while parent(P) ≠ null do
12:     propagate any tags to parent(P) (match or not match)
13:     if team(parent(P)) = parent_team(T) then
14:       propagate tags down to untagged plans
15:       create pointer from parent_team(T) to parent(P)
16:       T ← parent_team(T)
17:     P ← 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.
```

For example, suppose that the players are known to be executing the IN-

TERRUPT plan (see Figure 4). 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 other plans (in particular INTERRUPT) as not matching. 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 then create a pointer from the root node in the team hierarchy, representing the ISIS'97 team (the root in Figure 4-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 16–17) 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, and since they are both tagged matching and non-matching, a disagreement will be detected.

YOYO uses minor enhancements of the data-structures as used by RESL. 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[3], 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.

YOYO's first part matches plans against all observations (lines 1–8), and thus takes $O(NL)$, where N is the number of agents (generating observations), and L the size of the plan-hierarchy (i.e., number of nodes). 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 $O(NL)$ 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 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 in H, at most, to represent the additional agent. In contrast, RESL uses an additional copy of the entire plan hierarchy for every

[3]The time is taken to be local to the agent running YOYO. No need for a global clock.

additional agent.

However, 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. Thus the bounds presented in the previous section play a significant role in reducing overall run-time, regardless of whether RESL or YOYO are used.

5 Experiments

To evaluate the efficacy of the YOYO algorithm, we empirically compare the run-time performance of RESL and YOYO on identical monitoring problems, as the number of monitored agents is scaled up. Trials were carried out in several different simulated domains:

ModSAF. The ModSAF environment is a commercially-developed high-fidelity simulation system, which was used as the environment for coordination-failure experiments in [20, 21]. In the original system, teams of 3–6 simulated helicopters utilized (individually) a plan-hierarchy (as described above) to execute training exercises. In this paper, we re-created the principal components of the plan-hierarchy, encompassing four sub-hierarchies, with ambiguity between them (i.e., more than one path through the plan-hierarchy would match a given observation).

RoboCup. The ISIS'97 and ISIS'98 RoboCup Soccer Simulation teams [24] employed a plan-hierarchy used by the agents to play virtual soccer. In this paper, we re-created again important (and ambiguous) portions of their plan-hierarchies.

RoboCup Simple. We revisited the plan-hierarchy for the RoboCup domain above, but removed any ambiguity. Thus observations lead to selecting a single correct hypothesis, with no need for reasoning about multiple hypotheses. The motivation for using this domain is to show the effect of ambiguity on run-time in both cases. 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. We recorded RESL and YOYO's execution time (matching observations, inference, and disagreement detection). For accuracy, we recorded actual CPU time, rather than wall-clock time. 30 trials were done for each fixed number of agents. The number of agents was varied between 2 and 3000, in skips of 10.

Figures 5 and 6 show the results of these experiments in the ModSAF and RoboCup domains, respectively (see below for the results in the *RoboCup Simple* domain). The X axis in both figures denotes the number of agents monitored (from 2 to 3000). The Y axis shows the average run-time in CPU seconds, for

running the disagreement-detection algorithm on a single trial. Figures 5-a and 6-a show the run-time of YOYO and RESL on a scale appropriate for RESL's performance (hundreds of seconds). Since it is difficult to see the results for YOYO on such scale, Figures 5-b and 6-b show the same figures, on a scale appropriate for YOYO—approximately 3 orders of magnitude smaller.

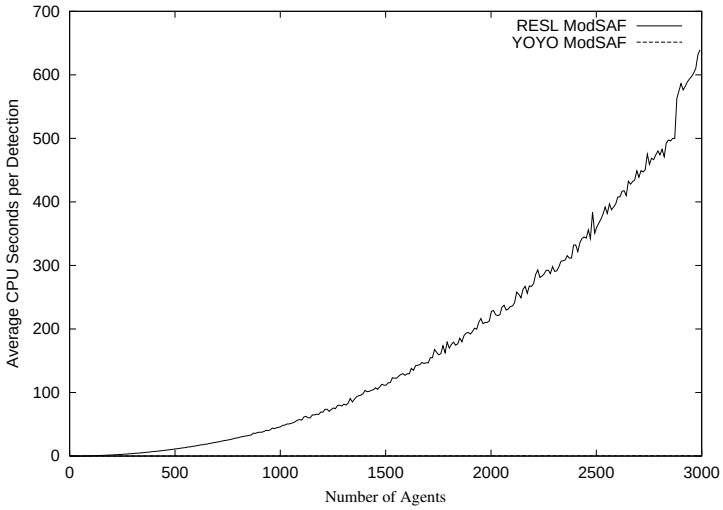
The *RoboCup Simple* domain shows different results. Here, there is no uncertainty in the domain, as observations correspond uniquely to plans within the plan-hierarchy. As a result, much of RESL's machinery for tracking multiple hypotheses is not used. Figure 7 shows the run-time results in this domain; each point is an average of 300 runs. The axis are the same as in the previous figures (though we skipped 50 agents at a time in varying the number of agents). However, note here that due to the lack of ambiguity, RESL's run-time is almost comparable to YOYO's (though YOYO is still faster). The small difference between RESL and YOYO in this domain is likely due to the overhead in RESL's maintenance of multiple plan-hierarchy structures: Where RESL updates N structures (each with a single hypothesis), YOYO updates one.

The difference in performance between YOYO and RESL in the different domains is not just a function of YOYO's maintenance of a single structure (rather than the N structures maintained by RESL). It is also due to the fact that it considers only coherent hypotheses, of which only a linear number (in the size of the plan hierarchy) exist.

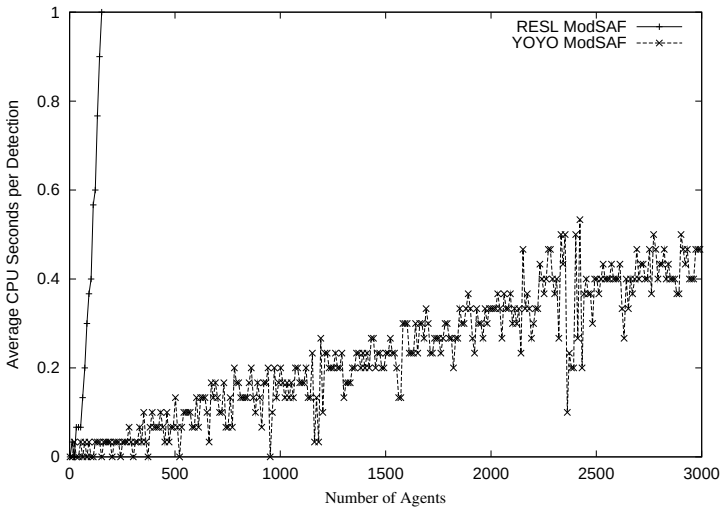
6 Discussion and Future Work

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, the challenges of limited connectivity and an exponential number of potential failures are raised. This paper has addressed these challenges, in the context of a critical monitoring task—detection of disagreements between teammates. First, the paper has shown that by carefully constructing the monitoring graphs of the system—who is monitoring whom—one can make guarantees on the quality of disagreement detection, while reducing the connectivity requirements on the agents. Second, we have presented YOYO, a highly-scalable algorithm that provides disagreement detection even in large-scale teams.

Our work in this paper is only one example in a growing body of literature that considers large-scale systems [33]. There are challenges that are independent of those discussed above: Reducing the load on the monitoring agent [5], formal methods and data-structures for representing large-scale organizations [16], planning and execution with large-scale systems [38], etc. Our work in this paper complements these. For instance, Durfee [5] discussed decision-theoretic and heuristic methods for reducing the amount of knowledge that agents consider in coordinating. Thus, while Durfee's work focuses on reducing computational loads in monitoring each single self-interested agent, our work focuses on reducing the number of monitored agents, and on savings

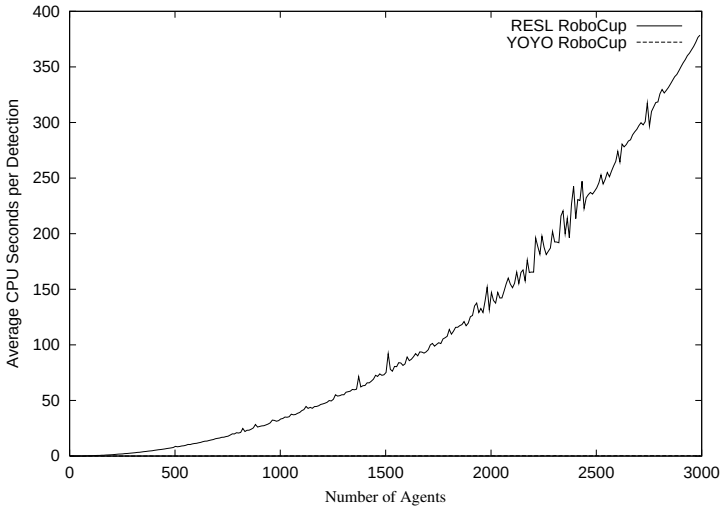


(a) RESL and YOYO (Y axis in range of 0–700 seconds).

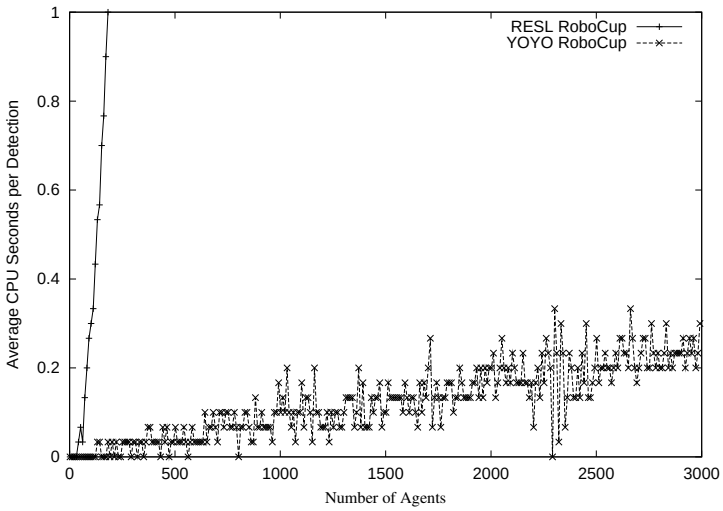


(b) RESL and YOYO (Y-axis limited to 1 second).

Figure 5: RESL and YOYO run-times in the ModSAF domain. The X axis marks the number of agents monitored; the Y axis denotes computation time in CPU seconds. *Note that the range of the Y axis is different in (a) and (b).*



(a) RESL and YOYO (Y axis in range of 0–400 seconds).



(b) RESL and YOYO (Y-axis limited to 1 second).

Figure 6: RESL and YOYO run-times in the RoboCup domain. The X axis marks the number of agents monitored; the Y axis denotes computation time in CPU seconds. *Note that the range of the Y axis is different in (a) and (b).*

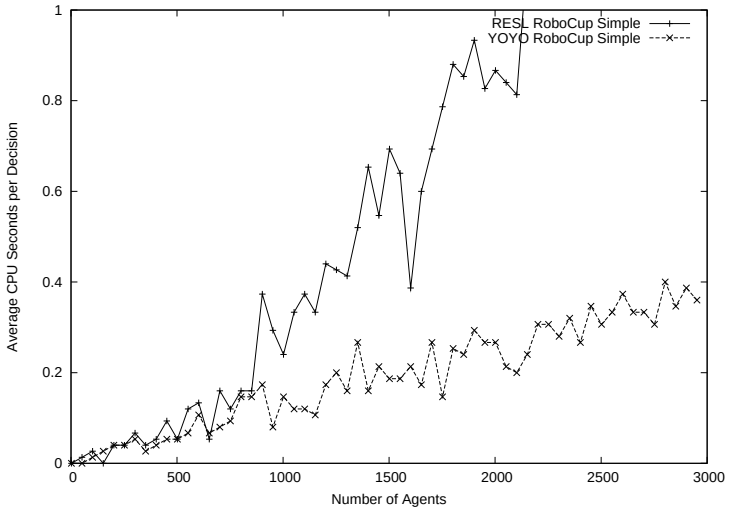


Figure 7: RESL and YOYO run-times in the RoboCup Simple domain. The X axis marks the number of agents monitored; the Y axis denotes computation time in CPU seconds.

possible only when monitoring teams together.

Recent work on model-based diagnosis has also begun to address limited connectivity, though indirectly, and only to a limited extent. Work by Roos et al. [31,32] has examined the use of model-based diagnosis by agents diagnosing a distributed system. While the methods describe do not address coordination failures, they are certainly relevant in terms of discussing the type of connectivity assumptions required for the diagnosis to work. Our recent work [14,15] on the use of model-based diagnosis of disagreements also limits connectivity: A key focus is on using only a handful of agents to represent all others in the diagnosis process, thus limiting run-time and communication load.

Key issues and assumptions remain open for future investigation. With respect to the monitoring bounds presented in Section 3, there remain questions as to the satisfaction of the assumptions in practice. We argue for both sides: On one hand, any agent broadcasting its state to its peers is a key agent. And indeed many real-world systems utilize communications to alleviate uncertainty; any such system can now utilize the bounds to manage key agents better, by focusing such broadcasts where absolutely required. Moreover, the conditions discussed in the paper only constrain the types of graphs that will guarantee complete and sound detection: They apply for both static and dynamic monitoring graphs. Nevertheless, it is certainly the case that we should continue to seek improved bounds, and/or weakened assumptions.

YOYO also leaves open questions. YOYO works with a known set of team/subteam plans; and its run-time complexity involves the size of the plan-hierarchy. It thus expects a closed system, with a manageable plan-hierarchy.

But as systems grow in the number of agents, they may require working with an open plan-hierarchy, that can change as agents are added or removed from the system. Moreover, if the scale-up is not only in the number of agents, but also in the size of the plan-hierarchy, then this will affect the scalability of the system. Finally, 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 directly identify what agents are involved, or the extent of the disagreement. Thus for diagnosis tasks [14, 15], YOYO has to be augmented by mechanisms that allow the monitoring agent to reconstruct the hypotheses underlying the disagreement. We hope to address these open questions in future work.

Acknowledgments

This work is based in part on a 2002 paper by the author and Michael Bowling [17]. We are indebted to Michael Bowling for his help in proving Theorem 2, for which he deserves joint credit. We also owe Michael many thanks for discussions and comments on earlier drafts of this work, including finding flaws with earlier attempted proofs. Milind Tambe and David V. Pynadath helped with initial versions of the YOYO algorithm. We also thank Meir Kalech, Dorit Avrahami-Zilberbrand, and Michael Lindner for useful discussions and corrections. As always, thanks to K. Ushi.

References

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

[2] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 35, 1991.

[3] C. Dellarocas and M. 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.

[4] M. Devaney and A. 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.

[5] E. 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.

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

[7] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 417–445. MIT Press, Cambridge, MA, 1990.

[8] B. Horling, B. Benyo, and V. Lesser. Using self-diagnosis to adapt organizational structures. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 529–536, 2001.

[9] B. Horling, V. R. Lesser, R. Vincent, A. Bazzan, and P. Xuan. Diagnosis as an integral part of multi-agent adaptability. Technical Report CMPSCI Technical Report 1999-03, University of Massachusetts/Amherst, 1999.

[10] M. J. Huber and E. 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] S. S. Intille and A. 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, 1999.

[12] N. R. Jennings. Commitments and conventions: the foundations of coordination in multi-agent systems. *Knowledge Engineering Review*, 8(3):223–250, 1993.

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

[14] M. Kalech and G. A. Kaminka. Diagnosing a team of agents: Scaling-up. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*, 2005.

[15] M. Kalech and G. A. Kaminka. Towards model-based diagnosis of coordination failures. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005.

[16] M. Kalech, M. Lindner, and G. A. Kaminka. Matrix-based representation for coordination fault detection: A formal approach. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

[17] G. A. Kaminka and M. Bowling. Towards robust teams with many agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, 2002.

[18] G. A. Kaminka and I. Frenkel. Flexible teamwork in behavior-based robots. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005.

[19] G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan recognition approach. *Journal of Artificial Intelligence Research*, 17:83–135, 2002.

[20] G. A. Kaminka and M. Tambe. I'm OK, You're OK, We're OK: Experiments in distributed and centralized social monitoring and diagnosis. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, pages 213–220, Seattle, WA, 1999. ACM Press. A slightly different version appears in proceedings of the IJCAI-99 workshop on team behavior and plan recognition.

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

[22] M. Klein and C. Dellarocas. Exception handling in agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*. ACM Press, 1999.

[23] S. Kumar, P. R. Cohen, and H. 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.

[24] S. C. Marsella, J. Adibi, Y. Al-Onaizan, G. A. Kaminka, I. Muslea, M. Tallis, and M. Tambe. On being a teammate: Experiences acquired in the design of robocup teams. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1–2), 2001.

[25] E. Platon. *Modeling Exception Management in Multi-Agent Systems*. PhD thesis, Laboratoire d'informatique de Paris 6, Universté et Marie Curie, 2007.

[26] E. Platon, N. Sabouret, and S. Honiden. An architecture for exception management in multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, 2(3):267–289, 2008.

[27] D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, 2002.

[28] D. V. Pynadath and M. Tambe. Automated teamwork among heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems*, 7:71–100, 2003.

[29] J. Rickel and W. L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13:343–382, 1999.

[30] J. Rickel and W. L. Johnson. Virtual humans for team training in virtual reality. In *Proceedings of the Ninth International Conference on Artificial Intelligence in Education*, pages 578–585. IOS Press, 1999.

[31] N. Roos, A. t. Teije, A. Bos, and C. Witteveen. An analysis of multi-agent diagnosis. *in Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-02)*, 2002.

[32] N. Roos, A. t. Teije, and C. Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. *in Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-03)*, pages 655–661, 2003.

[33] P. Scerri, R. Vincent, and R. Mailler, editors. *Challenges of Large Scale Coordination*. Springer, 2005.

[34] M. Tambe. Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.

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

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

[37] D. E. Wilkins, T. Lee, and P. Berry. Interactive execution monitoring of agent teams. *Journal of Artificial Intelligence Research*, 18:217–261, 2003.

[38] Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara. An integrated token-based approach to scalable coordination. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*, 2005.