Social Comparison for Failure Detection and Recovery

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

Abstract. Plan execution monitoring in dynamic and uncertain domains is an important and difficult problem. Multi-agent environments exacerbate this problem, given that interacting and coordinated activities of multiple agents are to be monitored. Previous approaches to this problem do not detect certain classes of failures, are inflexible, and are hard to scale up. We present a novel approach, SOCFAD, to failure detection and recovery in multi-agent settings. SOCFAD is inspired by *Social Comparison Theory* from social psychology and includes the following key novel concepts: (a) utilizing other agents in the environment as information sources for failure detection, (b) a detection and repair method for previously undetectable failures using abductive inference based on other agents' beliefs, and (c) a decision-theoretic approach to selecting the information acquisition medium. An analysis of SOCFAD is presented, showing that the new method is complementary to previous approaches in terms of classes of failures detected.

1 Introduction

Agent behavior monitoring in complex dynamic environments is an important and well known problem, e.g., [3], [10]. This problem is exacerbated in multi-agent environments due to the added requirements for communication and coordination. The complexity and unpredictability of such dynamic environments causes an explosion of state space complexity, which inhibits the ability of any designer, human or machine (i.e., planners), to enumerate the correct response in each possible state. The agents are therefore presented with countless opportunities for failure, which could not have been anticipated. For instance, it is generally difficult to predict when sensors will return unreliable answers, communication messages get lost, etc.

The agents must therefore be responsible for autonomously detecting the failures, and for recovering from them. To this end, an agent must have information about the ideal behavior expected of it. This ideal can be compared to the agent's actual behavior to detect discrepancies indicating possible failures. Previous approaches to this problem (e.g., [3], [10], [15]) have focused on the designer or planner supplying the agent with redundant information, either in the form of explicitly specified execution-monitoring conditions, or a model of the agent itself which may be used for comparison. Indeed, monitoring explicit conditions on the agent's behavior have proved useful to us in initial stages of failure detection.

However, both of these approaches suffer from limitations which render them insufficient for failure detection in general:

1. *Information failures.* Both approaches fail where relevant information is unexpectedly unavailable. For instance, if a condition monitor depended on a sensor to provide verification, a failure of the sensor will render the monitor useless.
2. *Inflexibility.* Monitoring conditions in agent behavior can be too rigid in highly dynamic environments, as agents in complex environments must often adjust their behavior flexibly to respond to the actual circumstances they are in.
3. *Difficulty in scaling up.* Both approaches mandate that the designer supply redundant information, which entails further work for the designer, and encounters difficulties in scaling up to more complex domains. Model-based approaches require the designer to specify the agent design twice, in a sense: Once in designing the agent, and again in designing a self-model for simulation and comparison.

We propose a complementary novel approach to failure detection and recovery, which is unique to multi-agent settings. This approach, SOCFAD (Social Comparison for FAilure Detection), is inspired by ideas from *Social Comparison Theory* [9]. The key idea in SOCFAD is that agents use other agents as sources of information on the situation and the ideal behavior. The agents compare their own behavior, beliefs, goals, and plans to those of other agents, in order to detect failures and correct their behavior. The agents do not necessarily adapt the other agents' beliefs, but can reason about the differences in belief and behavior, and draw useful conclusions regarding the correctness of their own actions. This approach alleviates the problems described above:

1. It allows agents to overcome information failures, as relevant information may be inferred from other agents' behavior and used to replace or complement the agent's own erroneous perceptions.
2. It allows for flexibility in detecting failures, since the flexible, dynamic, behavior of other agents' is used as an ideal for comparison.
3. It doesn't require the designer to provide the agent with redundant information about itself (in the form of a model or conditions), utilizing instead other agents as sources of information.

One key general heuristic used in SOCFAD is application in a team context. In particular, teamwork or collaboration is ubiquitous in multi-agent domains. An important issue in SOCFAD is that the agents being compared should be *socially similar* to yield meaningful differences. By constraining SOCFAD to use team-members for comparison, we narrow down the search for socially-similar agents. Furthermore, by exploiting agent modeling (plan-recognition) techniques to infer team members' goals, SOCFAD enables efficient comparison without significant communication overhead. We also allow the agent to explicitly reason about *social roles* and *status*, so that it can compare itself only to agents that can provide it with meaningful information.

SOCFAD is implemented and discussed within the context of IFDARS (Integrated Failure Detection And Recovery System), a system provided to our agents for the purpose of failure detection and recovery. IFDARS integrates different failure detection and recovery techniques within a unified framework, allowing evidence from different failure detection modules to be combined and reasoned about explicitly.

An additional novelty in IFDARS is that it brings forth an assumption that is implicitly made with the other approaches: The model (or condition) provided by the designer is always correct (the *model-correctness assumption*). However, in social comparison, other agents act as the knowledge sources, and cannot be assumed to be correct at all times. In detecting failures by social comparison, the agents must reason not only about the actual differences found, but also about the possibility that the agent itself is not at fault, but its social role models. By making this assumption explicit, IFDARS recovery modules can utilize different information sources and parameterized biases to reason about the differences in a general way.

2 SOCFAD and IFDARS: Motivation

The motivation for our approach comes from our application domain which involves developing automated pilot agents for participation in synthetic multi-agent battlefield simulation environments [12]. The environment was commercially developed for military training, and is highly dynamic, complex, and rich in detail. In addition to the unpredictability of the environment, communications and sensors are unreliable, mission and task specifications may be incomplete, etc. These qualities present the agents with never-ending opportunities for failure, as anticipation of all possible internal and external states is impossible for the designer. Two examples may serve to illustrate: In the first, a team of three helicopters takes off from the home base and heads out towards their battle position. While two of the agents follow the mission plan, a single agent hovers in place at the starting position indefinitely, due to an unanticipated miscommunication of the mission specification. In the second example, a similar team of three agents arrives at a specified landmark position. One of the team-members, whose role is that of a scout, is to continue forward towards the enemy, identifying and verifying its position. The scout's team-mates are to wait for its return in the specified position, and indeed one agent correctly lands and waits. Due to unanticipated sensory failure, the remaining agent, which is also supposed to wait, does not detect the landmark marking the waiting point. Instead of waiting behind, it continues to fly forward with the scout, following it into the battlefield.

We have collected dozens of such failure reports over a period of a few months. While it is generally easy for the human designer to correct these failures once they occur, it is generally hard to anticipate them in advance. The failures occur despite significant development and maintenance effort -- given the complexity of the dynamic environment, predicting all possible states and all possible interactions is impossible.

These failures are not negligible. Rather, they are very obvious failures, usually due to unanticipated (by the human designer) circumstances, and generally catastrophic, completely prohibiting the agent in question from participating in the simulation. In the first example above, not only is the single agent stuck behind unable to participate in the simulation, but the remaining agents are unable to carry out the mission by themselves.

An underlying quality of many of these failures is that they are not specific to military procedures. Indeed, the domain experts expect some level of common sense handling of failures even in the most structured and strict military procedure. By exercising social common sense, an agent may at least detect that something may be wrong, even if it does not have knowledge of the military domains. Social clues, such as (in the examples above) noticing that team-mates are leaving while the agent is hovering in place, or that a team-member has landed while the team was flying in formation, would have been sufficient to infer that something may be wrong.

3 IFDARS

The *Integrated Failure Detection And Recovery System* (IFDARS) integrates different failure detection and recovery techniques, and allows for evidence from multiple sources to be combined and reasoned about explicitly (Figure 1).

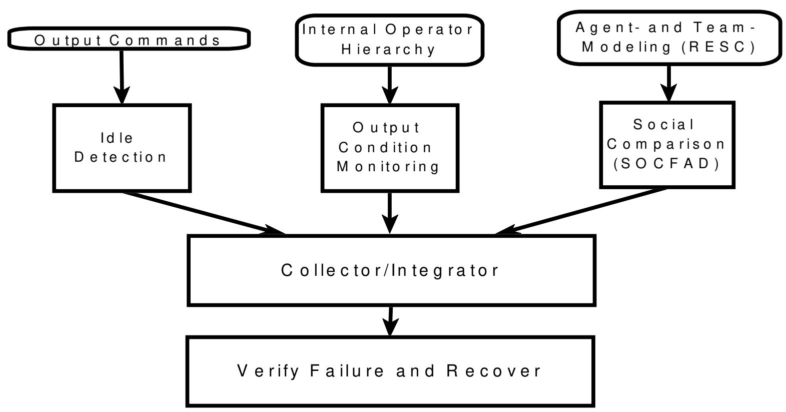

Figure 1. IFDARS Structure.

IFDARS uses three different failure-detection modules which all interface in a unified manner by generating '*interesting events*'--indications of possible failures (false positives are allowed). Events have *specificity*--they may be *generic*, i.e., general indications, not localized to a specific fault; or they may be *specific*--indicating for instance that the aircraft may have a problem with its speed. To allow the system to reason about specific failure-detection modules, all events are tagged by the failure-detection module that generated them. Events have *weight* which indicates how important they are, and *certainty* that in fact a failure took place. Events may also be generated as a response to other events. For example, a continuing repetition of events is itself a reason to suspect a failure may be underway, and so if a failure repeats itself too often, an event describing this repetition is generated.

The events are collected together from the different detection modules in the collector/integrator component. Given a set of events $E_1,...,E_k$, with corresponding weights $W_1,...,W_k$ and certainties $C_1,....,C_k$, for specificity i, the alarm level A_i is calculated as follows:

$$A_i = \sum_{i=1}^{k} W_i C_i$$

Once alarm levels are raised above threshold, the system reasons about the possible failures, verifying and possibly recovering from the failures. The recovery process lowers the alarm levels appropriately.

The three current failure-detection modules in IFDARS are: (a) a social comparison module, implementing SOCFAD, (b) a condition-monitoring module, and (c) an activity measurement module. The three modules utilize different input sources for detecting failures. The condition monitoring module monitors the currently running reactive plans, via designer-supplied conditions. The activity measurement module attempts to detect when the agent is unreasonably idle (i.e., stuck). The social comparison module is the basis for SOCFAD. We have found all three modules to be useful in detecting failures in the agent's behavior, but as condition-monitoring approaches and activity measurement monitoring are already common techniques in failure detection, we will focus on the social comparison process in the next section.

4 Social Comparison for Failure Detection: SOCFAD

SOCFAD is inspired by Social Comparison Theory [4], a theory from social psychology, developed to explain cognitive processes in groups of humans. Newell [9] presents the first three axioms of this theory as follows (pg. 497):

1. Every agent has a drive to evaluate its opinions and abilities.
2. If the agent can't evaluate its opinions and abilities objectively, then it compares them against the opinions and abilities of others.
3. Comparing against others decreases as the difference with others increases.

The numerous reports of failures we have collected demonstrate the very real need of agents in dynamic, unpredictable domains to evaluate themselves by monitoring their execution. This empirically verifies the importance of the first axiom. Approaches emphasizing the designer as a source of information against which to compare the agent's performance fit naturally under the title of objective sources for the agent's self-evaluation. SOCFAD focuses on the remaining parts of the axioms - allowing the agent to compare its own abilities and opinions (i.e., behavior, beliefs, operators, and goals) to those of others (second axiom), and considering the weight put on the results of such comparison (third axiom).

Although Social Comparison Theory is descriptive, we have begun to operationalize it for monitoring (see Algorithm 1). The abstract version of our algorithm accepts inputs representing the states of agents being compared - their beliefs, goals, behavior, etc. The agents' states are then compared by *Find-Difference* to detect possible failures, and a social similarity metric is used in the function *Similarity* to produce a level of certainty in the detected failure.

```
Social-Failure-Detect(myself, other-agents)   {
        1.Difference ← Find-Difference(my-self, other-agents)
        2.If Difference = NIL then goto 5
        3.Failure-Certainty ← Similarity(Difference)
        4.If Failure-Certainty > 0 then return Difference as  a detected
          failure, with certainty Failure-Certainty.
        5.No failure was detected.  Return NIL.
}
```

Algorithm 1. Social Failure Detection (Abstract Version).

The interesting issues in this algorithm are hidden in the two functions *Find-Difference* and *Similarity*. Different capabilities and performance result by changing the information being compared by *Find-Difference*, (e.g., internal beliefs and goals vs. observable behavior). In *Find-Difference*, it is useful to (i) limit agent states compared for efficiency, and (ii) use information that captures the control processes of the agents. Agent's plan hierarchies usually satisfy both constraints, but potentially other aspects of states could be used. The *Similarity* function reasons about the social similarity of agents being compared, and translates the differences to a certainty that indeed a failure has occurred. These algorithms will be incrementally developed through the rest of this section.

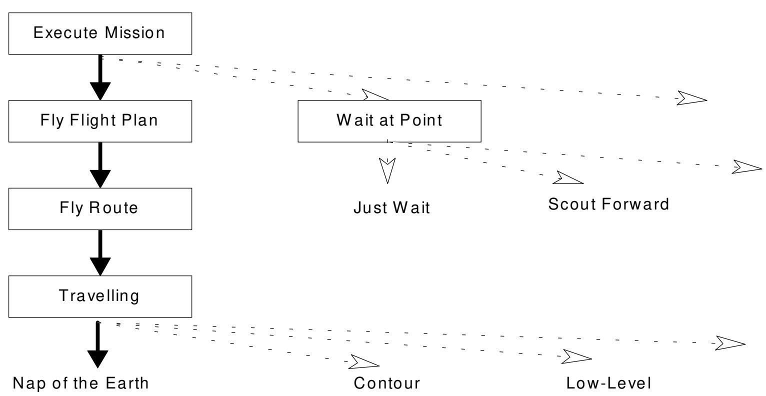

Figure 2. An Example Operator Hierarchy

Our agents' design is based on reactive plans (operators) ([5], [9], [11]), which form a hierarchy that controls each agent (Figure 2). The design implements the *Joint Intention Framework* [7]. Following this framework, operators may be team operators (shared by the team) or individual (specific to one agent). Boxed operator names signify team operators, which achieve and maintain joint goals, while the other operators are individual. Team operators require coordination with the other members of the team as part of their application ([13], [14]). Figure 2 presents a small portion of the hierarchy. The filled arrows signify the operator hierarchy currently in control, while dotted arrows point to alternative operators which may be used. In the figure, the agent is currently executing the execute-mission team operator as its highest-level team plan, and has chosen to execute the fly-flight-plan operator, for flying the agent team through the different locations specified in its mission.

Operator hierarchies form the basic structure of our agent's reasoning process, and were natural objects for comparison. To operationalize SOCFAD we require a way of acquiring knowledge of the operator hierarchies of other agents (so that we have something to compare against), a definition for *Find-Difference* (a procedure for comparing hierarchies), and a definition for *Similarity* as well.

In theory, knowledge of other agents can be communicated. However, such communication is often highly impractical given significant communication costs, risk in communicating in hostile environments, and unreliability in dynamic and uncertain settings. Instead our implementation of SOCFAD relies on agent modeling (plan recognition) techniques that infer an agent's beliefs, goals, and plans from its observable behavior and surrounding.

We use the $\text{RESC}_{\text{team}}$ [13] method in modeling other agents, but different techniques may be used interchangeably, as long as they provide the needed information and representation. $\text{RESC}_{\text{team}}$ will be briefly described here (see [13] for more detail). $\text{RESC}_{\text{team}}$ represents other agents' plans by building additional operator hierarchies in the agent's memory which correspond to the other agents' inferred reactive plans currently executed. Thus, the monitoring agent has unified access not only to its own original operator hierarchy, but also to the inferred operator hierarchies of other team members (Figure 3).

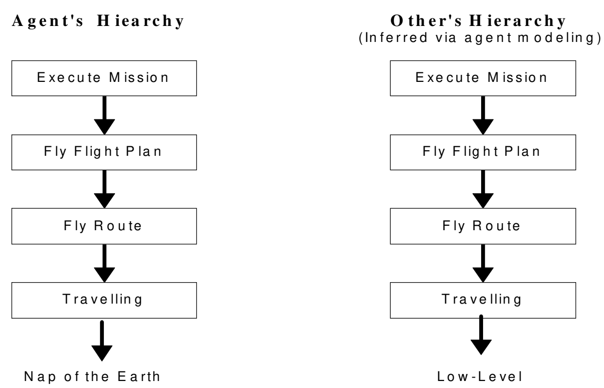

Figure 3. An Example of Two Hierarchies in the Agent's Memory.

Based on the representation of the other agents' plans by operator hierarchies, the *Find-Difference* function can be easily implemented (Algorithm 2) as the simple process of comparing the chosen operators in equal depths of the hierarchies. Hierarchies of different lengths are also considered different.

```
Find-Difference(my-operator-hierarchy, other-hierarchies) {
    1.Depth ← 0
    2.Compare operators in hierarchies at depth Depth
    3.If a difference is found, return it.
    4.Else, are the operators leaves of the hierarchies?
        4.1 No. Increase Depth.  Goto 2.
        4.2 Yes. Return NIL.
}
```

Algorithm 2. Find-Difference.

As implied by the third axiom of social comparison theory, differences with other agents are meaningful only to the extent that the other agents are *socially similar*. Other agents may not be executing plans that are relevant to the agent's goals, and therefore may not be able to contribute relevant information towards the monitoring of the agents own plans and goals. Worse yet, other agents may intentionally want to use deception in order to influence the agent's decision making to advance their own agendas.

Fortunately, a team context provides an initial solution. Team members tend to work on joint goals and sub-plans related to the one the agent should be executing, and can be assumed to be non-hostile (therefore not intentionally deceiving the agent in question). The comparison process in Find-Difference therefore considers team members only.

4.1 Team Operator Differences

Our agents use the Joint Intentions framework [7] as the basis for their coordination of team activities. In this framework, explicit team operators form the basis for teamwork, requiring mutual belief (MB) on the part of the team members as a condition for the establishment, and termination (based on achievement, unachieveability, or irrelevancy), of explicit team operators. Team operators must therefore be identical for all team members. A difference in team operators is therefore a certain sign of failure, regardless of its cause.

In one example above, one agent has failed to detect a key landmark position and continued execution of the "fly-flight-plan" team operator. However, its teammates correctly detected the landmark and terminated execution of that operator. They then switched to executing "wait-at-point" team operator, in which two agents are to land while the *scout* is to go forward and scout the enemy position. Through agent modeling, the miscoordinating agent infers the operators the other agents are executing. It realizes that they could potentially be executing the "wait-at-point" operator and detects a discrepancy with its own team operator of "fly flight plan". At this point it does not know which side is correct – either itself is at fault or its teammates. Regardless, the agent can conclude that a failure has occurred with the team and the coordination among its members.

The purpose of utilizing the joint intentions framework is to benefit from the domain-independent guarantees it provides for team coordination. As the agents are designed to follow the framework, it would appear at first that the above failures of miscoordination cannot occur. However, given the well recognized difficulty of establishing mutual belief *in practice*, differences in team operators do occur. In the example motivating this discussion, since the landmark that was to signal termination of the "fly-flight-plan" operator is in the external environment, it was assumed to be visible to all agents. Thus, mutual belief that the landmark was detected was assumed by the agent that successfully detected it, and it correctly abandoned the team operator (this assumption is motivated by the inefficiency of continuous communications). The second agent, which had missed detection of the landmark was true to the joint intentions framework as well: It didn't abandon one team operator without establishing mutual belief that it was achieved, unachievable, or irrelevant. The key point is that while both agents have correctly followed the joint intentions framework, a failure in sensing, coupled with a practical assumption about establishment of mutual belief caused the joint-goals of both agents to differ. And no matter which agent is right, a failure has certainly occurred, since the team is no longer coordinated.

```
Similarity (Operator-Difference) {
        If Operator-Difference is between team operators then
                return maximum certainty
}
```

Algorithm 3a. Similarity, Version 1.

To operationalize this discussion, we can now define an initial version of the Similarity function used in the social failure detection algorithm (Algorithm 1). The key idea is that at the team level, agents have identical team operators, and so are maximally socially similar.

4.2 Individual Operator Differences

The previous section discussed differences between agents that are maximally similar--agents that have joint goals and together form a team. However, in service of team operators different agents may work on different individual operators. These individual operators do not necessarily carry with them the responsibilities for mutual belief that team operators do, and so differences in individual operators are not sure signs of failure, but at best indications of the possibility.

We therefore require additional information about the agents causing the difference which can help in determining whether the difference is justified or not. For instance, agents working towards similar goals have similar *social roles:* For example, in a soccer game there are field players and a goalie which have different roles within the team. Agents with similar roles would serve as better sources of information for plan-execution monitoring than other agents. Related to the social role is *social status*, which may also justify differences in individual operators among team members. For instance, in the military domain agents of different ranks may follow different individual operators to guide their behavior.

The example where a failing agent was stuck in place while its team-members have taken off and were flying away serves to illustrate this distinct type of discrepancy. Here, a comparison of the agent's own chosen method-of-flight operator to the methods of flight chosen by its comrades indicates to the agent that it is not acting like the rest of the team - that in fact a failure may have occurred (see leaf operators in Figure 3).

We have provided our agent with the means to explicitly utilize the social similarity of team-members in their reasoning. The agent explicitly considers the parameter of the social role of other agents within the team in filtering and assigning weights to the information inferred about them. For example, if the agent is an *attacker*, which is one of the roles in a team in our domain, it will assign more weight to other agents which are *attackers*. For efficiency, the agent may completely ignore agents which it decides, based on their role, are not relevant as information sources.

Even after filtering irrelevant differences with agents of differing social roles, there remain individual differences which are justifiable and do not constitute a failure, simply because agents may not necessarily find themselves in identical external and internal states. For instance, in the real world, no two agents can share the exact same physical space. We therefore require more techniques which can raise our confidence that indeed a failure has occurred, when a discrepancy in individual operators is found.

The above discussion brings us to an updated version of the Similarity function, incorporating the heuristics discussed above: social role and social status. The exact definition of role and status, and the weights by which they modify the certainty (3.1-3.2 in Algorithm 3b below) are domain dependent, as are the default certainties that a failure has occurred.

```
Similarity (Operator-Difference) {
    1. Certainty ← Default /* or a-priori certainty */
    2. If Operator-Difference is between team operators then
         Certainty←Maximum Certainty /* From version 1 */
    3. Else /* difference between individual operators */
         3.1 If Operator-Difference is between agents with same Role, then increase
              Certainty, else decrease it.
         3.2 If Operator-Difference is between agents with same Status, then
              increase Certainty, else decrease it.
    4. Return Certainty.
}
```

Algorithm 3b. Similarity, Version 2.

4.3 Towards Recovery Based on Social Comparison

In general, to recover from a failure, a process of diagnosis is required. Here again social comparison raises novel issues. First, it does not make the *model-correctness assumption* made in previous approaches. Second, it allows the process of diagnosis and recovery to utilize social sources of information which were not utilized before.

Model-correctness assumption. In recovering from a failure detected by social comparison, the agent must reason explicitly about the differences in beliefs that exist between itself and the other members of the team. From the fact that other agents are executing a different plan, the agent can infer by abduction that the preconditions necessary for selection and execution of that plan were satisfied by the other agents. It can then reason about the relevance of these preconditions to its own selected plan. For instance, in the example of the agent's failure to detect a key landmark, it appears that the other agents are carrying out the wait-at-point operator (one agent lands, while the other one which is known to be the scout goes forward). Once this discrepancy is noted ("I am executing fly-flight-plan, they are executing wait-at-point"), the agents makes an abductive inference that the other agents believe that the team has indeed reached the landmark. However, the agent does not necessarily adapt the team-members' view - it does not assume the model (other agents) to be correct.

Socially-based recovery. As the model-correctness assumption is made explicit, social information sources can be utilized for diagnosis and recovery. If the agent believes it is at fault, it can alter its own beliefs by adopting the preconditions which it inferred are satisfied for the other agents' operators. In particular, team operator's preconditions require mutual belief, and so by adopting them the agent allows the correct team plan to be selected, therefore synchronizing itself with the rest of the team. For example, the agent in the landmark example fixed its own beliefs regarding the landmark based on this abduction. This fulfills the preconditions of its own "wait-at-point" operator, which is now selected and allows the agent to recover gracefully from the failure.

5 Results and Evaluation

Our agent, including IFDARS, is implemented completely in Soar [9]. Approximately 1200 rules are used in the implementation of the agent, which includes the military procedures, as well as the teamwork and agent-modeling capabilities. Additional 60 rules implement IFDARS, forming an add-on layer on top of the procedures making up the agent.

The social comparison approach to failure detection complements the condition-monitoring detection methods, being able to detect different types of failures. In general, the condition-monitoring approaches cannot detect failures where a feature of the environment is not detected, and are limited in their abilities to detect failures where the inputs to the agent (as perceived by the sensors) are incorrect. Model-based approaches in particular use the agent's own inputs to generate an ideal output which is compared to the actual output to detect problems in the process converting inputs to outputs. However, failures may occur in the inputs to the agent due to sensory problems, resulting either in incorrect readings or in missing perceptions. A model-based approach cannot detect these failures as it uses the erroneous inputs. However, the social comparison approach can detect such failures and correct them as demonstrated in the example of the undetected landmark.

In contrast, social comparison methods will encounter problems in single-agent situations, or if all team members encounter identical failures simultaneously (which we hypothesize to occur very infrequently in complex multi-agent settings). There a process of comparison would not generate any differences if the execution of the plan is incorrect, as the agents would all display the same incorrect behavior.

Here a model-based or condition-monitoring approach is very suitable for detecting failures.

Our explicit choice to prefer agent-modeling to communications for acquiring the information for comparison from the other agent stems from practical constraints common to many multi-agent domains. However, in general, using the following decision tree, an agent can decide whether to use agent modeling for acquiring the information, or to have (by request or by design) the other agents communicate back their beliefs, plans and goals:

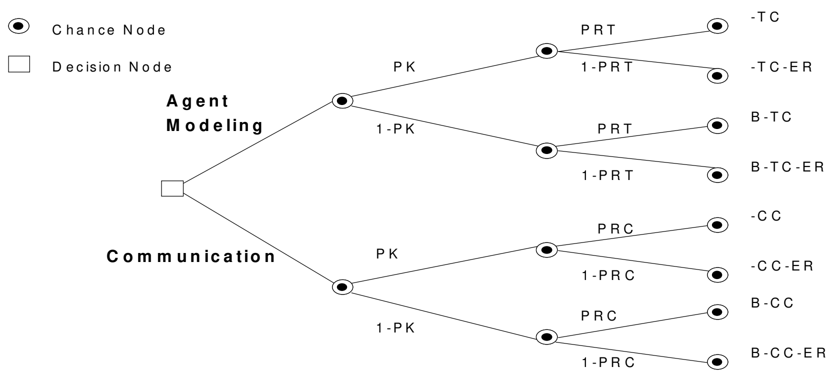

Figure 4. Decision Tree for Information-Acquiring Method.

The purpose of acquiring information about the other agents (by either modeling or communications) is to detect possible failures. In the decision tree above, the agent seeks to maximize its expected utility from the process of acquisition. Thus, a bonus B is rewarded in the decision tree above if the agent indeed acquires new information. This bonus is not awarded if the information does not differ in any way from what the agent already knew. TC and CC are the costs for modeling and communicating, respectively (CC is the total cost incurred by sender and receiver). PK is the probability that the information is already known (so no difference is detected), PRT the probability that the modeling process was reliable, and PRC the probability that communications were reliable (for both modeling and communication, reliability implies that the information was acquired correctly). ER is the penalty for making a mistake - for example for incorrect inference made by modeling, or for receiving an unreliable message.

By following the tree, it is clear that the agent should rely on agent modeling rather on communications from other agents whenever $TC+(1-PRT)*ER < CC+(1-PRC)*ER$. In our domain, the cost of communications is very high, as the agents operate in a hostile environment and expose themselves by communicating with each other. On the other hand, the cost of agent modeling is relatively low, being mostly a computational cost rather than a survival risk. In addition, in a team context, modeling is often reliable, in contrast to communications, which are unreliable quite often. Our estimation of reliability and the cost of error make agent modeling an attractive choice for acquiring knowledge of others. For simplicity, one may choose to assume reliable communications and agent-modeling, and then the agent's choice is dictated solely by the cost of modeling vs. communications.

6 Related Work

Social comparison is related to work on multi-agent coordination and teamwork, although in general, social comparison generalizes to also detect failures in execution of individual operators, which are outside the scope of coordination. Particularly relevant are *observation-based* methods, which utilize agent modeling rather than communications for coordination (e.g., [6], [14]). Work on teamwork [14] concentrates on maintaining identical joint goals to prevent miscoordination, while the focus of SOCFAD is on detecting when the goals do differ. Indeed, social comparison can be useful for recovering from failures in teamwork. The recovery from the undetected landmark failure mentioned earlier can be construed as an example of active coordination on the part of the team. Huber and Durfee [6] do not assume joint goals but instead look at coordination as emergent from opportunistic agents, which coordinate with others when it suits their individual goals. As these agents do not have team goals, they cannot assume maximal social similarity at the team coordination level, and so would not be able to detect team failure. Also, while Huber and Durfee demonstrate the benefits of using plan-recognition rather than explicit communications in a dynamic domain, they do not discuss the qualities of the domain which make plan-recognition beneficial. The decision-tree provided in the previous section presents a first step towards this direction.

Atkins et al. [2] attacks a similar problem of detecting states for which the agent does not have a plan ready. They offer a classification of these states, and provide planning algorithms that build tests for these states. However, their approach considers only the individual agents and not teams. It also suffers from the same limitations as condition monitoring approaches in not being able to detect modeled states which have not been sensed correctly. For instance, their approach cannot detect states which were not planned-for by the planner, but are still "safe" [2] such as the example of the undetected landmark.

Social comparison is also related to imitation [2]. In fact, imitation can be shown to be a special case of the general social comparison algorithm (Algorithm 1). By choosing to compare itself against the observable behavior of other agents, rather than their internal goals, the social comparison approach leads to imitation. In the example of the agent failing to detect a landmark and land, a simple imitation of the scout would be clearly inadequate. Alternatively, imitation of the other *attacker* would lead to failure later on as the agent is still executing the wrong team-operator and follows the wrong sequence of actions.

To illustrate this point further, consider a similar case, where the failing agent is actually the scout that is supposed to go forward. Upon reaching the landmark, its two team-members land waiting for it to go forward. Since it didn't detect the landmark the agent is still executing the flying-in-formation plan. If it were to imitate its team-mates, it would simply land or hover near them while they are waiting for it to go forward. Instead, with SOCFAD the agent would compare the plan that it is executing with those of its team-mates, and realize that they are now executing a different plan, based on detecting a landmark which it has failed to detect. It could thus recover from such an error.

Mataric [8] used socially similar agents (*next of kin*) to investigate generation of group behavior from local interactions, while the focus of SOCFAD is on failure detection. By restricting group members to be socially similar, Mataric showed little communication is necessary as the agents can make correct predictions on the behavior of their peers, and this allows coherent group behavior to emerge. Although SOCFAD emphasizes the importance of social similarity for the individual, we do not assume it. In fact, a core issue in SOCFAD is the search for socially similar agents which can be used for comparison among all agents.

7 Summary and Future Work

This paper presents a novel approach to failure detection, an important problem plaguing multi-agent systems in large-scale, dynamic, complex domains. Existing approaches often face difficulty in addressing this problem in such domains. The key novelties of our approach are: (a) a new failure detection method, utilizing other agents in the environment as information sources for comparison, (b) a general heuristic for team-based comparison, (c) a detection and repair method for (previously undetectable) information failures using abductive inference based on other agents' beliefs, and (d) a decision-theoretic approach to selecting the information acquisition medium.

Several issues are open for future work. One important issue is in techniques and biases useful for deciding which side is correct where a difference is encountered with another agent, but no information is known to support either side. Previous approaches have arbitrarily chosen to bias their decision by making the model correctness assumption implicitly. IFDARS allows to explicitly handle this state by other biases and heuristics to be used. A simple techniques that may be used is to follow the majority, so that if a majority of agents agree with one agent, its beliefs and behavior is taken to be correct. Such a technique has clear limitations, but initial experiments show it to be quite useful. Another technique is to bias the agent detecting the failure towards accepting responsibility for the failure (low self-confidence) or for rejecting it, possibly attributing it to the other agent. This bias can be easily parameterized, and can result in very different behaviors on the part of the failure-detecting agent. An additional option enabled by IFDARS is to utilize evidence supplied by other failure-detection modules to provide additional evidence.

Another important issue left for future work is the integration of learning into the detection and recovery process, whereby the agent should be able to learn not only how to respond to detected failures, but also the settings in which they are likely to arise, how to prevent them from happening, etc. A key object for learning is social similarity, where the agent would learn which agents are socially similar, or otherwise serve as good source of information for failure-detection purposes (good role-models).

References

1. Atkins, E. M.; Durfee, E. H.; and Shin, K. G. 1996. Detecting and reacting to unplanned-for world states, in *Proceedings of the AAAI-96 Fall symposium on Plan Execution*. pp. 1-7.
2. Bakker, P.; and Kuniyoshi, Y. 1996. Robot see, robot do: An overview of robot imitation. *AISB Workshop on Learning in Robots and Animals*, Brighton, UK.
3. Doyle R. J., Atkinson D. J., Doshi R. S., Generating perception requests and expectations to verify the execution of plans, in *Proceedings of AAAI-86*, Philadelphia, PA (1986).
4. Festinger, L. 1954. A theory of social comparison processes. *Human Relations*, 7, pp. 117-140.
5. Firby, J. 1987. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87).*
6. Huber, M. J.; and Durfee, E. H. 1996. An Initial Assessment of Plan-Recognition-Based Coordination for Multi-Agent Teams. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*. Kyoto, Japan. pp. 126-133.
7. Levesque, H. J.; Cohen, P. R.; Nunes, J. 1990. On acting together, in *Proceedings of the National Conference on Artificial Intelligence (AAAI-1990)*, Menlo Park, California, AAAI Press.
8. Mataric, M. J. 1993. Kin Recognition, Similarity, and Group Behavior. In *Proceedings of the Fifteenth Annual Cognitive Science Society Conference.* Boulder, Colorado. Pp. 705-710.
9. Newell A., 1990. *Unified Theories of Cognition*. Harvard University Press.
10. Reece, G. A.; and Tate, A. Synthesizing protection monitors from causal structure, in *Proceedings of AIPS-94*, Chicago, Illinois (1994).
11. Rao, A. S.; Lucas, A.; Morley, D., Selvestrel, M.; and Murray, G. 1993. Agent-oriented architecture for air-combat simulation. Technical Report: Technical Note 42, The Australian Artificial Intelligence Institute.
12. Tambe, M.; Johnson W. L.; Jones, R.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent Agents for interactive simulation environments. *AI Magazine*, 16(1) (Spring).
13. Tambe, M. 1996. Tracking Dynamic Team Activity, in *Proceedings of the National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon.
14. Tambe, M. 1997. Agent Architectures for Flexible, Practical Teamwork, in *Proceedings of the National Conference on Artificial Intelligence*, Providence, Rhode Island (To appear).
15. Williams, B. C.; and Nayak, P. P. 1996. A Model-Based Approach to Reactive Self-Configuring Systems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon.