This paper appears in the Proceedings of the Ninth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'15), November 2-6, 2015, Larnaca, Cyprus, and in Lecture Notes in Artificial Intelligence (Springer). The final publication is available at http://dx.doi.org.

Using Contextual Knowledge for Trust Strategy Selection

Larry Whitsel[1] and Roy M. Turner[2]

[1] University of Maine at Augusta
Bangor, ME 04401 USA
larry.whitsel@maine.edu
[2] School of Computing and Information Science
University of Maine, Orono, ME 04469
rturner@maine.edu

Abstract. In open multiagent systems in which some of the agents may be self-interested, it is vital that an agent be able to make trust decisions about its peers to determine which of them may be untrustworthy and how to behave in response. The agent's context, including the environment, observed and inferred actions and motives of other agents, and properties of the MAS as a whole, is critical to making an informed trust decision and especially to choosing a strategy for taking actions. However, most prior work in the area has ignored context or only treated it implicitly. In this paper, we present an implemented approach that explicitly represents the agent's context, informed by known contexts, and that uses that contextual knowledge to select the best strategy, even in the presence of untrustworthy agents.

Keywords: Multiagent systems, trust decisions, self-interested agents, context representation, miscreant agents

1 Introduction

An open multiagent system (OMAS) is a multiagent system in which agents can come and go and may not all be under the control of the same entity. An agent in an OMAS that contains self-interested agents must be able to identify others in the society that are likely to act in a way inimical to its own interests, to the society's interests, or both. We refer to these as *miscreant agents*. Once a miscreant has been identified, the agent must then decide which actions to take when interacting with it to protect and advance its own interests.

The problem of identifying a miscreant agent is essentially a *trust decision*: for each other agent in the system, can we trust that agent? We use *trust* here the general sense of Castelfranchi and Tan [3] to mean both that the agent is trustworthy in the normal sense and that it is capable of performing whatever action we are concerned with.

Unfortunately, trust decisions are made difficult by several factors. Self-interested agents will usually have utility functions for their actions that are

hidden or otherwise unpredictable, and so an agent will need to observe their behavior over time to detect patterns indicating their trustworthiness. In addition, an agent's view of the current situation will be incomplete, since in most domains the environment will be only partially-observable. Consequently, the agent will need to infer what its observations may have missed, including others' actions and their effects on others.

Determining how to behave in the presence of miscreants is also difficult. We call an agent's mapping (implicit or explicit) from particular configurations of the situation to actions its *strategy*. An agent will have multiple possible strategies to choose from, but not all will be appropriate for the context. In particular, the strategy in use will depend on the presence and kind of miscreant behavior.

Consequently, knowledge about the current context must be used to determine which strategy to select at any given time. As the context changes—for example, as miscreants come and go in the OMAS—the agent should use its contextual knowledge to select strategies to automatically tune its behavior in response.

Research on trust decisions has generally been concerned with identifying what we call miscreants by focusing primarily on the other agent. For example, socio-cognitive approaches [5], or that of Mao [10], used game theoretic or psychological attribution theory to either respond directly to or to infer an agent's motives. Reputation-based approaches (see, e.g., [12]) decide to trust an agent based on its past observed behavior. Machine learning approaches (e.g., [7]) have attempted to learn strategies based on a set of interactions with another agent.

Few if any approaches, however, have focused on using the context, including the environment, the goals of the agent (and the MAS), and observed properties of other agents, including their reputation, to infer the presence of miscreants and how to respond to them. Yet doing so allows better assessment of the true situation, and it can allow previously-successful strategies to be immediately identified and used.

In our approach, we use explicitly-represented contextual knowledge to help an agent identify and respond to miscreant agents. Known contexts—classes of situations—are represented in knowledge structures called *contextual schemas* (c-schemas) [17]. Features of the current situation, including environmental features, current goals, and possible indications of miscreant behavior, are used to select one or more matching c-schemas, essentially identifying the current context. The c-schema(s) then suggest appropriate strategies for the situation.

The approach is called CATS (Context-based Agent Trust System), and it has been implemented and tested in a toy domain (Liar's Dice) that is a reasonable surrogate for many real-world open multiagent system domains [18,19]. In this paper, we focus on how contextual knowledge is represented and used in CATS.

In the remainder of the paper, we will first discuss context and its representation, then how an agent using the CATS approach assesses and manages its context. This makes use of inferences about properties of agents based directly on their own actions as well as on their interactions with others (society-level

analysis of motives [18]). We then briefly touch upon our evaluation of CATS via experiments in a simulated MAS.

2 Context and context representation

We view a *context* as a kind of situation, where a situation is itself the sum of all observed and unobserved features of the environment, the agent, other agents, etc. Contexts are classes of situations that have some implication for how an agent should behave [17]. For example, the context of an agent operating in an OMAS with a pair of collusive miscreant agents would likely encompass many different situations: playing bridge, participating in an auction, and so forth, where such agents were present. The utility of recognizing a particular situation as an instance of a known context is that it allows the agent to focus on the salient features and to select appropriate behavior.

Contexts are related to one another. One context can be a specialization of another, for example, or it can be a blend of several other known contexts. For example, an agent engaged in an Internet bandwidth auction where there are collusive agents present could consider its context as composed of other contexts such as as participating in an auction, negotiating about bandwidth, and operating in the presence of collusive agents. In our approach, if there is something about the composite context that has important implications for the agent's behavior, then it will be remembered; otherwise, the agent will blend the components when it is encountered again.

As we have argued elsewhere (e.g., [17]), an agent should explicitly represent contexts it knows about rather than spreading contextual knowledge across its knowledge base (e.g., as rule antecedents or plan preconditions). Context representations can be a means of bundling facts and assertions about the world (cf. [6]), allowing all relevant information to be retrieved at once about the context, as well as facilitating knowledge acquisition and learning. In addition, explicit representations allow an agent to commit to what its context is, then automatically behave appropriately until the context changes, thus saving reasoning effort.

We represent contexts as *contextual schemas* (c-schemas) [17]. C-schemas both limit the scope of reasoning and bundle together related knowledge about (in CATS) trust decisions and appropriate behavior. Previous work on trust has used the agent's context only to limit scope, e.g., turning a dynamic situation into a static one by limiting the agent's predictions or decisions to one case and ignoring the rest. In contrast, we allow multiple contexts to be recognized at once and represented by multiple c-schemas as well as make use of context as a bundling mechanism by adding additional knowledge to our assessment of other agents' actions to help fill in missing knowledge.

CATS relies on a library of c-schemas from which the appropriate one(s) can be found based on the situation. In other work [9,17], c-schemas have been organized in a dynamic conceptual memory (e.g., [8]). Here, we make no commitment

```
(defrule in-wary
  (is-me ?me ?cf)
  (game-round (bidder-is ?x)(certainty ?cf0))
  (or (and
        (claim (claim-type ally)(source ?x)(target ?y)(certainty ?cf1))
        (claim (claim-type enemy)(source ?me)(target ?y)(certainty ?cf1)))
      (system-state (state-name global-alert)(certainty ?cf1))
      (claim (claim-type enemy)(source ?me)(target ?x)(certainty ?cf1))
      (or (belief-strength (belief trustworthy) (source ?x)
          (strength very-low)(certainty ?cf1))
          (belief-strength (belief trustworthy) (source ?x)
          (strength low)(certainty ?cf1)))
      (claim (claim-type cheating)(target ?x)(certainty ?cf1))
      (claim (claim-type enemy)(source ?x)(target ?me)(certainty ?cf1))
      (belief-strength (belief bid-aggressive) (source ?x) (target ?me)
          (strength very-high)(certainty ?cf1))
      (belief-strength (belief bid-aggressive) (source ?x) (target ?me)
          (strength high)(certainty ?cf1)))
 =>
  (assert (in-context wary ?cf1)))
```

Fig. 1. A rule from the contextual schema `in-wary`.

as to how c-schemas are stored; we simply assume that they can be retrieved as needed based on the situation's features.

We are not concerned in CATS with many of the kinds of information usually stored in c-schemas (e.g., context-dependent semantics, event-handling information, etc.). Instead, we are more concerned with strategies, that is, mappings from situations to actions. C-schemas in this project are associated with a strategy that is appropriate for all situations that are instances of the context represented. Thus, contexts in this project are delimited based on changes in strategy.

At the present time, to populate our agent's c-schema library, we rely on the system designer. In an ideal system, the agent would learn its own c-schemas, either by modifying those initially given to it or creating them de novo. This would require the agent to track the success of strategies in different situations and group together similar situations that share strategies successfully.

Showing a complete c-schema would require more space than can be allotted in this paper, but an example of one production rule that acts as descriptive knowledge for a c-schema called `in-wary` is shown in Figure 1. In this case our agent is making use of an embedded CLIPS [13] rule-based system to implement its context manager.

3 Context assessment and management

Representing context is half the problem. The other is identifying the context the agent is in and managing the corresponding representations.

In CATS, we break this into four processes or modules: tracking the situation to produce inferences useful for recognizing the context (Track); society-level analysis of motives to produce additional interagent-based inferences (Analyze); recognizing the context and managing the contextual knowledge (Contextualize); and selection and execution of context-appropriate strategies by the agent (Execute).

Figure 2 shows an overview of our approach, the major modules discussed below, and some of the data paths between them.

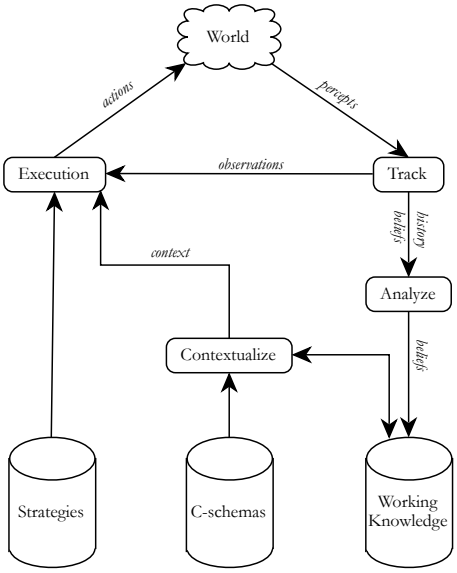

Fig. 2. Overview of the reasoner's architecture

3.1 Tracking the situation

The Track module is primarily concerned with making non-society-level inferences about the situation that can help identify the context. It uses the agent's percepts and the history of those percepts over time to produce beliefs about the current situation that can then be used by the other CATS modules. This is reminiscent of some robotics software architectures (e.g., [2]), where raw, objective sensor data is aggregated and processed into subjective, symbolic, agent-centric inferences for use by (e.g.) situation assessment.

We are primarily interested here in how an agent can differentiate between agents it can trust and those who are miscreants. Thus, Track is concerned with establishing the values of *trust-warranting properties* (e.g. [1]) of others. For this, we look to humans for inspiration.

Humans, being social animals, are greatly concerned with recognizing deception by others; it is even possible that we have evolved specialized "cheater-detection" circuits in our brains [4]. However, even for humans, detecting deception is a difficult task. It is even more difficult in environments with electronic-mediated communications [15], which of course has implications for trust decisions by artificial agents.

Humans use the perceived intent of others' actions to help decide about trusting them. To do this, we rely on a combination of observable trust-warranting properties ("manifesta"), a history of previous interactions, and beliefs we have about the motivations of others [1,3]. From these, we develop beliefs about other, hidden trust-warranting properties ("krypta" [1]). Humans also use others' reputations to make trust decisions, as do some agent-based systems (for a review, see [14]). Others' reputations can be maintained by an agent itself or by relying on a (hopefully trusted) third party.

Both of these sources of trust-warranting information are mirrored in CATS' Track module. One source of information is properties of the problem domain that are easily observable, for example, board positions or moves in a game domain or bidders and values of bids in an auction. The value of this kind of immediately-available information has low *durability*, that is, it decays rapidly over time, and it is highly domain-specific.

Another kind of information available has to do with properties of the OMAS itself, such as protocols used, organizational structures in effect, and rules of behavior. These tend to be more durable. For example, the speed at which system rules or protocols change is constrained by the need to disseminate them to all participating agents. Although knowledge of how another agent interacts with the system itself may not be readily generalizable to other OMASs, it can give significant indications of the agent's intentions and trustworthiness.

Track produces beliefs about the world and about trust-warranting properties for each other agent in the society. We represent beliefs as tuples $< P, O, V, C >$, where P is the property that is the subject of the belief, O is the object(s) or agent(s), V is the value of the property for O, and C is a confidence value in the range $[-1, 1]$. O can consist of more than one object. For example, the belief that agent A has goal G might be represented as $< \texttt{hasGoal}, (A, G), 0.735 >$.

Track adjusts its beliefs, including confidence in existing beliefs, as the situation changes. It takes what may be called a "frequentist" approach to belief confidence revision: the more observations that are made that support a belief, the more confident Track becomes about that belief. Knowledge about the durability of a property, as discussed above, factors into this as well.

Interagent interactions are another source of information important for making trust decisions. This is the purview of the Analyze module and its society-level analysis of motives.

3.2 Society-level analysis of motives

An important source of trust-warranting information is how another agent interacts with others in the OMAS. By observing this, an agent can infer possibly-

hidden motives. This is called *society-level analysis of motives* (SLAM, introduced earlier in relation to collusion detection [18]), which is done by the Analyze module.

Analyze depends on the beliefs created by Track about domain, system, and agent properties. It uses knowledge about how agent actions and their results can affect others to create beliefs about relationships between agents. It also takes into consideration exogenous events and actions such as messages between the agents, system-wide information messages, agent entry or exit, and so forth. The beliefs are then used by Contextualize as it determines what the current context is. Analyze also makes use of contextual information from Contextualize to change which knowledge it uses to create beliefs. This circularity is unavoidable, since the meaning of agent interactions is itself context-dependent.

Analyze examines each action taken by another agent to determine what it might mean to other agents, including itself. For example, an agent may take an action that indicates that it is unusually aggressive, that it is similar to another, or that provides disproportionate benefits to some other agent (possibly indicating collusion).

Occasionally, any agent's actions may seem to indicate miscreant behavior, for example, by going against our agent's interests or disproportionately benefiting some other agent. However, it is the *pattern* of an agent's behavior over time that is most important in making a trust decision, as well as any sudden unexpected changes in the pattern, for example, if agent A always takes actions that benefit B more than itself, or if an aggressive agent suddenly begins to act benignly toward another.

As an example, suppose our agent is engaged in a series of "English", or outcry, auctions with other agents. Suppose that one of the other agents always bids aggressively, yet never wins the bid, thus bidding up the final price. Its actions have a cost to itself, i.e., the actual cost of participating and the potential cost incurred if it should win an auction. However, the benefits, except if it wins, go to another agent in the system: the seller. Our Analyze module would regard this as grounds for the hypothesis that there is an unsanctioned coalition (i.e., collusion) between that agent and the seller, even though no explicit collusive actions or messages have been observed. It would consequently treat the other agents as miscreants.

Analyze accrues evidence about factors indicating relationships by keeping a weighted count of the occurrences (or non-occurrences) of the factors, then, during each of its decision cycles, calculating the mean and standard deviation for all observations. Bayesian inference is then used to infer a new belief state from the prior one, and the probabilities are used to assign a symbolic value to the degree of belief, from `very-weak` to `very-strong`. Rules are then used to combine these individual beliefs into beliefs about relationships.

For example, in our auction example, we inferred that one of the agents bid aggressively, had actions that disproportionately benefited the auctioneer, and seldom won a bid. The combination of these factors would allow Analyze to believe the hypothesis that the agent was a shill for the auctioneer and, hence,

a miscreant, with a confidence value derived from the probabilities determined for each factor involved.

Once Analyze has inferred an agent's motives and its relationships to others, it remembers these as a *durable hypotheses* about the agent. A durable hypothesis is different from other beliefs in that it persists from one decision cycle to another, whereas others are generated from historical and perceptual information at the start of each decision cycle. The duration of hypotheses will vary based on the stream of more transient beliefs and on a decay process that the Analysis module uses to reduce the confidence in the hypothesis over time. We have found it useful to have multiple classes of durable hypotheses, each with its own decay rate. The more significant a hypothesis is to our assessment of the situation, the longer we wish to consider it, but still at some point the agent forgets an old hypothesis if beliefs cease to support it.

Based on its durable hypotheses and other beliefs, the Contextualize module then can determine the current context.

3.3 Context management

The Contextualize, or context management, module uses the beliefs and durable hypotheses from Track and Analyze to determine what the current context is and to manage the application of associated contextual knowledge.

As an example of context management, consider this scenario. On a cold winter day, you are sitting at your desk at work. A man enters the building wearing a ski mask. Suddenly, you hear the sound of sirens outside, and the man steps quickly into another room and closes the door. Here, the data is straightforward. We have two human actors, three locations (outside, the entryway, and the other room), and at least three objects (the desk, the ski mask, and the siren). Why does the siren beginning to sound result in a change in your perception of the other person's trustworthiness?

In our approach, the explanation is that you perceive a change in the context. Prior to the sound of the siren, the observation that a man had arrived wearing a ski mask was evaluated in the context of cold weather; our agent would have formed the hypothesis that the other person had donned the mask as a response to local conditions. When the siren sounds and the other agent takes an action to conceal himself, the agent's idea of what the context is and what the man's motivations are would change, and its trust in the agent should drop. Whereas our prior context-motivated strategy may have caused us to continue working on a paper, after the context change, a new "act warily" strategy may suggest the action "call police".

Contextualize has four major tasks. First, it must find contextual schemas matching the current situation. Second, it must *activate* the appropriate c-schemas from this set and deactivate any currently-active c-schemas that no longer match the situation. Third, it must select a strategy appropriate for the current context and make that available to the Execute module. And fourth, it must update the knowledge used by the other modules to reflect what is known or predicted about the context based on the knowledge from active c-schemas.

C-schema retrieval. Contextualize has to find c-schemas that match the current situation. In previous work, we relied on a content-addressable "dynamic" schema memory for this (e.g., [8,9]) of the kind sometimes used in case-based reasoning. However, we do not require such a mechanism. As long as an appropriate set of c-schemas can be found that match the current situation, CATS is agnostic as to the mechanism.

In our approach, we refer (after [11]) to c-schema retrieval as *evoking* a set of c-schemas based on the situation's features. In CATS, the primary features used are the beliefs about the world and interagent interactions produced by Track and Analyze. At the evoking stage, we are not overly concerned with the degree of match between a contextual schema and the situation; it is enough that some beliefs "bring to mind" (evoke) the c-schema with some level of confidence (based on the belief's confidences).

Contextualize continually watches the situation to detect context changes that should bring to mind new c-schemas; or, to better phrase it, it is constantly looking for c-schemas that now match the situation, which could indicate a change in context. If it detects beliefs predicted by a c-schema not currently in use, than that c-schema is considered. In addition, c-schemas in CATS explicitly list the "boundary conditions" for entering or leaving the represented context. This is used by Contextualize as well to change the set of c-schemas under consideration.

There may be situations in which no specific c-schema can be identified. For this eventuality, CATS has a "default" c-schema that is always applicable, albeit with low confidence.[3] Thus, when no other, more-specific context is recognized, this c-schema would provide a default strategy; otherwise, it is "overruled" by more specific c-schemas during activation/deactivation.

C-schema activation/deactivation. Retrieving contexts is only part of the problem of context assessment, since not all will be good matches. For example, being involved in an Internet auction for bandwidth might remind an agent of auctions, auctions involving some of the agents that are present now, and Internet auctions; only the latter two would be good matches, since the first is just a generalization of the latter.

In other work, we use a differential diagnosis process to determine which of the evoked c-schemas should be used as the representation of the current context [17]. In this work, we are more focused on representing and using contextual knowledge, and so we use a correspondingly simpler diagnostic process. Contextualize uses a version of MYCIN's certainty factor combination technique [16] to assess the match between the observed/inferred situation and what is predicted by the agent's c-schema. This includes the boundary rules mentioned previously that c-schemas may contain that, when matched to the situation, change Contextualize's belief in what the current context is. The c-schema or c-schemas that this process selects as matching the current situation are then *activated* to represent the current context.

[3] In other projects, the corresponding c-schema would automatically be retrieved from memory when no more-specialized ones were found.

Some c-schemas that had matched the situation may be found to be no longer appropriate as the situation changes. Contextualize will notice this during its diagnostic process. In addition, it can use any boundary rules in the old c-schema that suggest the context is no longer in effect. Contextualize then deactivates such c-schemas.

Strategy selection. Once it has assessed the context, Contextualize must find the best strategy for the current context and pass that along to Execute. This is done using knowledge provided by the c-schemas representing the current context, i.e., by using their suggestions for strategies appropriate for the current context.

Currently the results of our contextual reasoning is the selection of a single strategy from those suggested by the active c-schemas. This may be a weakness if a truly novel context occurs in the society, since our a priori c-schemas may not respond well to the unanticipated context. In the future, we will look at how to mitigate this problem by combining strategies from multiple c-schemas, each of which may capture some aspect of the novel context, to create a new strategy that will work better than any existing one.

Once Execute receives a strategy suggestion from Contextualize, it can use this strategy until the situation changes enough for Contextualize to send it a new strategy.

Propagate contextual knowledge. In addition to suggesting a strategy to Execute, Contextualize is also responsible for propagating other contextual knowledge to the agent's modules. C-schemas provide the agent with declarative and procedural knowledge. This includes knowledge the other modules use to do their jobs, for example, to create beliefs about the state of the world and about agent motives. When the context changes, such knowledge from active c-schemas needs to be activated, and knowledge unique to exiting c-schemas should be deactivated. If there are conflicts between knowledge from different c-schemas, then Contextualize chooses from among them based on its confidence in the c-schema as a fit for the situation.[4]

There are several kinds of knowledge that can be activated from c-schemas. For example, predictions about the world (e.g., a particular miscreant is present) may need to be activated by establishing a belief or a durable hypothesis. Existing beliefs may also be revised by changing their confidence values. Diagnostic rules (e.g., boundary rules from a c-schema) may be added to aid future context recognition. Other rules may be activated to be used by Track and Analyze to help them make decisions that are appropriate to the group. This will cause them to make inferences that are automatically appropriate for the context.

To illustrate how this might work, consider an agent that has been a long term member of an OMAS. Also existing in the same system is Agent X, for whom we have a long history of observations about its action. We have labeled Agent X as "trustworthy", and therefore use a cooperative strategy when dealing with it. During the course of observing activity in the system we notice that Agent X is treated well by Agent Z, but that Agent X makes inaccurate, negative reputation

[4] Other work in our lab looks at how to merge such conflicting contextual knowledge.

reports about Agent Z, in effect slandering Agent Z. Our agent would generate a number of hypotheses about this situation, e.g.: (1) Agent X seeks to undermine Agent Z; (2) Agent Z's reputation is lower than it should be; and (3) Agent X is not as trustworthy as we thought. These hypotheses, together or separately, should cause Contextualize to recognize a new context and recommend a new strategy the agent should use. As a result of context recognition, the agent should revise beliefs it holds to reflect the new hypotheses and beliefs about the effects of Agent X's actions. First, it might add a new assumption about the relationship of Agents X and Z. This would entail asserting a new durable hypothesis. Second, it should lower its confidence in any beliefs about the reputation of Agent Z, since it suspects that reputation has been compromised by Agent X's slander. Since beliefs are generated anew at the start of each decision cycle, simply updating the facts for the current cycle is not sufficient. Instead, Contextualize needs to modify Track's and Analyze's working knowledge so that they make the right decisions. Finally, it should discard its old durable hypothesis that Agent X is a trustworthy agent and prevent it from arising again by modifying the knowledge upon which Analyze makes its decisions, for example, by asserting that Agent X tends to slander Agent Z.

4 Empirical evaluation

To determine if adding context-based reasoning provides an improvement over general-purpose strategies, we constructed an agent using the techniques described in this paper. The agent interacted with other agents using our Liar's Dice (a poker-like dice game) test framework [19]. Each of the other agents employed a single general-purpose strategy (although possibly different for each agent), while our context-based agent chose which of the strategies to use based on its contextual reasoning. We tested the agent in several different test scenarios with both mixed-strategy and homogeneous-strategy societies. In some tests we introduced miscreant agents that employed a collusive strategy, while other cases did not employ miscreants.

For each test scenario we conducted a session of 36,000 games. With this number of games, there is a > 0.99 probability that the rarest roll (five sixes) will occur in the session. We can therefore expect that all possible hands will be experienced.

There were 25 distinct cases based on: whether the CATS agent was present or replaced by an agent following a random (*coinflip*) strategy; the presence of collusive agents (i.e., miscreants); and other context changes, such as the entry of a non-collusive agent or a system message announcing a change of state.

We measured the *success* of each agent, meaning the improved performance in the games, as well as its *correctness*, the percentage of correct decisions. We were also concerned with the *strength* of the strategy used by the agent, measured as $(w-l)/(w+l)$, where w and l are the number of wins and losses, respectively.

The results showed a statistically significant ($p < 0.05$) improvement in both correctness and success for our context-aware agent compared to other agents

in the society when the test scenario included a miscreant agent.[5] In all mixed-strategy experiments, the CATS agent displayed the highest level of correctness among all agents. In all but one mixed-strategy experiment, the context-aware agent was also the most successful agent in the society. In homogeneous-strategy society experiments, the context-aware agent performed well with or without the presence of miscreant agents, with two exceptions. Without miscreants present, the context-aware agent was equivalent to other members of the homogeneous society of stochastic agents, and it was inferior to other agents in the the case of a homogeneous society composed of altruistic agents. Barring these two special cases, the CATS agent showed better success and correctness in all cases, with statistically significant improvement when miscreant agents were part of the society. The improvement in strength of CATS over the second best strategy was > 2.6 for all, while the improvement in correctness over the second best strategy was > 3.32.

In experiments in which we inserted miscreant behaviors, the CATS agent maintained or slightly increased its correctness, while the correctness of other dynamic strategies (i.e., those that changed their actions based on conditions) were reduced by significant amounts ($p \leq 0.05$). Since the CATS agent uses only the strategies available to the other agents in the society, this indicates that its correctness is the result of its ability to better recognize the situation occurring in the society and respond to it by choosing a better strategy. This argument is further supported by the experiments in which no miscreant behavior was present. In these experiments, the CATS agent continued to exhibit comparable levels of correctness, but, since there were no miscreants to affect the other agents, their performance was improved. In the absence of miscreant behavior, but in the presence of context change, our approach continued to outperform other strategies, but the effect was not statistically significant. in terms of strength, the CATS agent was the same or slightly better than the others in the presence of miscreants ($p < 0.05$), but in experiments in which there were no miscreants, there was no significant difference in performance.

The results show that a context-aware agent enjoys an advantage over other agents in environments that include context change, and that the agent's ability to detect miscreant behaviors provides a statistically significant improvement in performance. The CATS agent's ability to detect miscreants allows it to choose the best strategy, rather than being locked into a single strategy, as were the other agents.

We believe that the size of the effect of context-based reasoning will vary based on the domain and on the amount of tuning put into the creation of the c-schemas. In the future we expect to be able to confirm and quantify this relationship.

[5] See [19] for complete details, including the statistical analysis.

5 Conclusion

We have described how a context-aware agent employing CATS can make better trust decisions, resulting in better performance with respect to its goals. In our approach, an agent not only tracks environmental factors and the actions of other agents, it also performs a society-level analysis of motives to uncover hidden relationships and motives agents may have. All of these are indicators of what the current context really is. Once the context is assessed, then contextual schemas representing that context provide suggestions for an appropriate strategy, e.g., to deal with miscreants, as well as other knowledge the agent can use in the context. We have implemented and tested this approach in a toy domain that has many characteristics of real-world open multiagent system domains. Experiments support the conclusion that context-aware strategy selection is beneficial in an OMAS. The approach was effective in two ways: (1) allowing an agent to make a correct decision about whether or not to trust another agent; and (2) maintaining good overall performance with respect to accomplishing the agent's own goals.

The results of these experiments are very encouraging and provide several further paths of inquiry, some of which we have mentioned in this paper.

References

1. Bacharach, M., Gambetta, D.: Trust as type detection. In: Trust and Deception in Virtual Societies, pp. 1–26. Springer (2001)
2. Blidberg, D.R., Chappell, S.G.: Guidance and control architecture for the EAVE vehicle. IEEE Journal of Oceanic Engineering OE–11(4), 449–461 (1986)
3. Castelfranchi, C., Tan, Y.: Introduction: Why trust and deception are essential for virtual societies. Trust and Deception in Virtual Societies (pp. xvii–xxxi). Dordrecht: Kluwer (2001)
4. Cosmides, L., Tooby, J., Fiddick, L., Bryant, G.: Detecting cheaters. Trends in Cognitive Science 9, 505–506 (2005)
5. Falcone, R., Castelfranchi, C.: Social trust: A cognitive approach. In: Castelfranchi, C., Tan, Y. (eds.) Trust and Deception in Virtual Societies, pp. 55–90. Kluwer Academic Publishers (2001)
6. Guha, R.: Contexts: A Formalization and Some Applications. 1991. Ph.D. thesis, PhD thesis, Stanford University, 1991. Also technical report STAN-CS-91-1399-Thesis, and MCC Technical Report Number ACT-CYC-423-91 (1991)
7. Iszuierdo, L., Izquierdo, S.: Dynamics of the Bush–Mosteller learning algorithm in 2x2 games. In: Weber, C., Elshaw, M., Mayer, N. (eds.) Reinforcement Learning: Theory and Applications, p. 424. I-Tech Education and Publishing, Vienna (2008)
8. Kolodner, J.L.: Retrieval and Organizational Strategies in Conceptual Memory. Lawrence Erlbaum Associates, Hillsdale, New Jersey (1984)
9. Lawton, J.H., Turner, R.M., Turner, E.H.: A unified long-term memory system. In: Proceedings of the International Conference on Case-Based Reasoning (ICCBR'99). Monastery Seeon, Munich, Germany (July 1999)
10. Mao, W.: Modeling social causality and social judgment in multi-agent interactions. Ph.D. thesis, University of Southern California (2006)

11. Miller, R.A., Pople, H.E., Myers, J.D.: INTERNIST–1, an experimental computer-based diagnostic consultant for general internal medicine. New England Journal of Medicine 307, 468–476 (1982)
12. Mui, L., Mohtashemi, M., Halberstadt, A.: Notions of reputation in multi-agents systems: a review. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1. pp. 280–287. AAMAS '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/544741.544807
13. Riley, G.: CLIPS – a tool for building expert systems. On the World Wide Web at http://www.ghg.net/clips/CLIPS.html, accessed December 16, 2004; last updated December 1, 2004. (2004)
14. Sabater, J., Sierra, C.: Review on computational trust and reputation models. Artificial Intelligence Review 24(1), 33–60 (2005)
15. Santos Jr, E., Johnson Jr, G.: Toward detecting deception in intelligent systems. In: Defense and Security. pp. 130–141. International Society for Optics and Photonics (2004)
16. Shortliffe, E.H.: Computer-based Medical Consultations: MYCIN. Elsevier, New York (1976)
17. Turner, R.M.: Context-mediated behavior. In: Brézillon, P., Gonzalez, A. (eds.) Context in Computing: A Cross-Disciplinary Approach for Modeling the Real World Through Contextual Reasoning, chap. 32, pp. 523–540. Springer (2014)
18. Whitsel, L., Turner, R.M.: A context-based approach to detecting miscreant behavior and collusion in open multiagent systems. In: Proceedings of the Seventh International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'11). Karlruhe, Germany (September 2011)
19. Whitsel, L.T.: A Context-Based Approach to Detecting Miscreant Agent Behavior in Open Multagent Systems. Ph.D. thesis, School of Computing and Information Science, University of Maine, University of Maine, Orono, ME 04469 USA (December 2013)