Effective, Quantitative, Obscured Observation-Based Fault Detection in Multi-Agent Systems

(Extended Abstract)

Michael Q. Lindner
Bar-Ilan University, Ramat-Gan, Israel
lindnerm@gmail.com

Noa Agmon
Bar-Ilan University, Ramat-Gan, Israel
agmon@cs.biu.ac.il

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems—*Cooperation and coordination*

General Terms

Algorithms, Reliability, Experimentation

Keywords

Agent Cooperation::Teamwork, coalition formation, coordination; fault detection

1. INTRODUCTION

A key challenge in multi agent systems is verifying that all agents obey the system rules at any moment [10, 6]. In behavior-based systems, each agent is in some 'state' at any moment. The system designer defines what states each agent might take, according to its teammates states (*plans* [9, 1] or *policies* [8]). Being a distributed system, no single agent has knowledge of the others. Monitoring for *fault detection* must rely on gathering information by communication or observation. This information is not always accurate [9, 5]. There are various types of approaches for that issue. Some researches define a-priory the possible failures, and identify them at run time [7, 4].

Others, which we adopt, prefer to define the allowed behavior, and identify exceptions from it [6, 8], in the following way: (a) Define a *policy* of the state-combinations agents are allowed to take; (b) at run time, each agent observes its teammate, deducing their possible states; (c) then, it compares them to those allowed by the policy and see if: (1) all the possible states are allowed (no fault); (2) none of them is allowed (fault); (3) some are allowed and some are not (possible fault).

Since the overall number of *joint states* in the system is m^n (number of *agents* powered by the number of *states* each agent might take), the naïve comparison has exponential complexity in both space and time. Various researches suggests ways for reducing that. In [8], a binary matrix based policy is used. Each matrix (*supercombination* or *s-comb*) represents the states each agent is allowed to take; a policy is defined using a *rule* of few s-combs—the union of all the combinations allowed by each of the s-combs. At run time, the possible-states observation is also given as a binary matrix. For example, assume a system of three agents $\{a_1, a_2, a_3\}$ and four states $\{s_1, s_2, s_3, s_4\}$. The policy rule φ defines the allowed states, and the *observation matrix* W—the possible states at a given moment.

$$\varphi = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \sqcup \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \quad W = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

That means, for example, that if agent a_1 takes state s_4, a_2 takes s_2 and a_3 takes s_1, their combinations is allowed. Since this combination is also found in W, it means that the agents might be in a legal state. However, W also express the combination of a_1 in s_4, a_2 in s_4 and a_3 in s_1, which is illegal (i.e., not defined by φ). Therefore, we cannot say whether there is a fault or not. Lindner et al. [8] present an algorithm that tests whether the system is assured to be faulty or not in a linear time and space complexity of $O(nm\ell)$ (where ℓ is the number of s-combs in the policy).

2. MOTIVATION

While existing researches suggest efficient algorithms for finding whether a system is faulty or is assured to be clean of faults, none provides means for detecting the probability of a fault where neither of those two unequivocal results are found. In real world system, dramatically different decisions might be taken for different values of that probability.

Calculate that probability requires somewhat different input in the first place. Rather than a set of 'possible states' per agent (say, as a binary matrix W), the observation now must include the *probability* of each agent to be in each state. In this text we assume that these probabilities are provided by a third party (e.g., by fusing the results of few inaccurate sensors).

A naïve calculation of the fault probability is the summary of the probability of each of the allowed combinations (φ_{legal}), where each of them is the product of each agent a_i's probability (expressed as the function θ_{a_i}) to be in the state dictated by this combination $\gamma(a_i)$. Mathematically:

$$\sum_{\gamma \in \varphi_{\text{legal}}} \prod_{i=1}^{n} \theta_{a_i}(\gamma(a_i)). \tag{1}$$

Obviously, the time complexity of implementing this equation is equivalent to the number of legal combinations, which might be m^n in the worst case.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

3. EFFECTIVE PROBABILITY CALCULATION

In order to effectively calculate the probability of a fault in the system, we suggest a different algorithm, based on the s-combs policy definition. First, we will provide a way to calculate the probability of a single s-comb policy.

3.1 Single S-Comb Calculation

Assume a system with a policy φ composed of a single s-comb, C. At run-time, an agent gets an *obscured observation* matrix W, representing the probability of each agent to be in each state at that time. The policy allows each agent a_i to be in *any* of the states marked '1' in row C_i. Let matrix H be the result of an element-wise product of C and W: $H = C \wedge W$. In this matrix, each element h_{ij} represents the probability of agent a_i to be in state s_j if this state is legal, or 0 if it is not. The probability of agent a_i to be in *some* legal state is therefore the summary of all the elements in row H_i. The product of all agents' probabilities provides the overall probability that the system has no fault:

$$\prod_{i=1}^{n} \sum_{j=1}^{m} h_{ij} \tag{2}$$

This calculation takes only $O(nm)$ time and space.

3.2 Multi S-Comb Calculation

Calculation of a multi s-comb policy must consider combinations that are defined by more than one s-comb in the policy rule, making them non-independent. In probability theory, the probability of several non-independent events' union is given by the inclusion-exclusion principle [2]. In our case, 'events' are s-combs, and we get:

$$P\left(\bigcup_{k=1}^{\ell} R^k\right) = \sum_{k=1}^{\ell} \left[(-1)^{k-1} \sum_{\substack{C \subseteq \{1,\ldots,\ell\} \\ |C|=k}} \left(\prod_{i=1}^{n} \sum_{j=1}^{m} H_{ij}^{C} \right) \right] \tag{3}$$

where H^C is the matrix $W \wedge \bigwedge_{c \in C} R^c$ (element-wise product of the ijth elements of the matrix W and all the matrices R^c). The time complexity of this calculation is linear in the number of agents and states, but is exponential in the number of s-combs in the policy—$O(nm\ell 2^{\ell})$; its space complexity is linear in all three parameters.

Fig. 1 shows the empiric calculation time of identical systems using the Naïve Algorithm vs. our S-Comb one. For the same number of s-combs, increase in the number of agents or states results in minor linear change of the S-Combs run time, but in exponential growth in the Naïve run time. Having more '1' elements in the policy s-combs ('density', [5]) results in more combinations defined by the same ℓ matrices of $n \times m$. That changes nothing in the S-Comb run time, but highly increases the Naïve run time. Only increasing ℓ affects the S-Comb curve mush more than the Naïve. In fact, actual run time might be much less than exponential in many cases, by some tweaking of the calculations.

4. PLANS

Some researches [9, 1] suggest policies that dictates different allowed states over time, step by step—*plans*. In our research, we extend the s-comb policies to allow definition of such plans, where each *stage* of the plan is defined by a single s-comb. We then suggest a continuous algorithm that

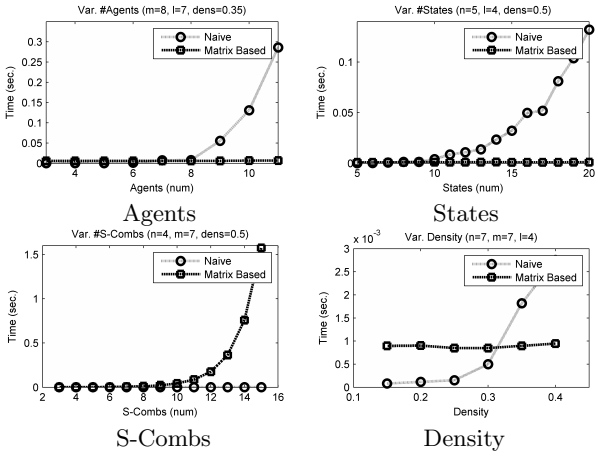

Figure 1: Runtime: Matrix-Based vs. Naïve

calculates the probability of no faults in the system, i.e., that all the agents are aligned to the same plan and stage. It hass time complexity of $O(mnd)$ and space complexity to $O(n(d+m))$, where d is the number of all the stages (in all plans) of the system.

5. REFERENCES

[1] B. Banerjee, L. Kraemer, and J. Lyle. Multi-agent plan recognition: Formalization and algorithms. In *AAAI'10*, pages –1–1, 2010.

[2] L. Comtet. *Advanced Combinatorics: The Art of Finite and Infinite Expansions*. Reidel, 1974.

[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, 2000.

[4] 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, January 1999.

[5] M. Kalech. Diagnosis of coordination failures: a matrix-based approach. *Autonomous Agents and Multi-Agent Systems*, 24(1):69–103, 2012.

[6] G. A. Kaminka and M. Tambe. Robust multi-agent teams via socially-attentive monitoring. *JAIR*, 12:105–147, 2000.

[7] M. Klein and C. Dellarocas. Exception handling in agent systems. In *Proceeding of the Third International Conference on Autonomous Agents*, pages 62–68, May 1999.

[8] M. Lindner, M. Kalech, and G. A. Kaminka. A representation for coordination fault detection in large-scale multi-agent systems. *AMAI*, 56(2):153–186, 2009.

[9] G. Sukthankar and K. Sycara. Hypothesis pruning and ranking for large plan recognition problems. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 998–1003. AAAI Press, 2008.

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