International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:7, 2009

Target Tracking in Sensor Networks: A Distributed Constraint Satisfaction Approach

R.Mostafaei, A.Habiboghli, and M.R.Meybodi

***Abstract*—**In distributed resource allocation a set of agents must assign their resources to a set of tasks. This problem arises in many real-world domains such as distributed sensor networks, disaster rescue, hospital scheduling and others. Despite the variety of approaches proposed for distributed resource allocation, a systematic formalization of the problem, explaining the different sources of difficulties, and a formal explanation of the strengths and limitations of key approaches is missing. We take a step towards this goal by using a formalization of distributed resource allocation that represents both dynamic and distributed aspects of the problem. In this paper we present a new idea for target tracking in sensor networks and compare it with previous approaches. The central contribution of the paper is a generalized mapping from distributed resource allocation to DDCSP. This mapping is proven to correctly perform resource allocation problems of specific difficulty. This theoretical result is verified in practice by a simulation on a real-world distributed sensor network.

***Keywords*—**Distributed CSP, Target Tracking, Sensor Network

I. Introduction

DISTRIBUTED resource allocation is a general problem in which a set of agents must intelligently assign their resources to a set of tasks such that all tasks are performed with respect to certain criteria. This problem arises in many real-world domains such as distributed sensor networks [3], disaster rescue [2], hospital scheduling [1], and others. However, despite the variety of approaches proposed for distributed resource allocation, a systematic formalization of the problem, explaining the different sources of difficulties, and a formal explanation of the strengths and limitations of key approaches is missing.

We use a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem [6]. These two aspects present some key difficulties. First, a distributed situation results in agents obtaining only local information, but facing global *ambiguity* — an agent may know the results of its local operations but it may not know the global task and hence may not know what operations others should perform. Second, the situation is dynamic so a solution to the resource allocation problem at one time may become unsuccessful when the underlying tasks have changed. So the agents must continuously monitor the quality of the solution and must have a way to express such changes in the problem. Given these parameters of ambiguity and dynamism, we will remember four classes of difficulties of the problem [6]. In order to address the resource allocation problem, the paper also defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DDCSP).

In this paper, for solving some of problems discussed above, we propose a new DCSP based approach. Since this approach reduces the number of operations of each agent, we improve previous approaches and furthermore, increase the number of targets to be tracked; and our proposed approach is more fault tolerant than others. The central contribution of the paper is as follows: In section II domains and motivations are discussed, in section III we briefly explain distributed constraint satisfaction problems, in section IV generalized mapping of DCSP to sensor networks are introduced, in section V we explain our proposed approach for target tracking and section VI contain our experimental results and finally conclusion is introduced in last section.

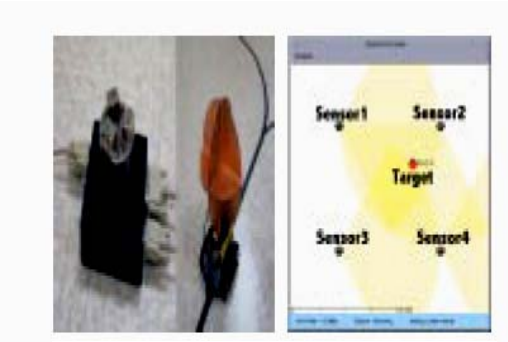
II. Domains and Motivations

Among the domains that motivate this work, the first is a distributed sensor domain. This domain consists of multiple stationary sensors, each controlled by an independent agent, and targets moving through their sensing range (Figure 1.a and Figure 1.b illustrate the real hardware and simulator screen, respectively). Each sensor is equipped with a Doppler radar with together three sectors. An agent may activate at most one sector of a sensor at a given time or switch the sensor off. While all of the sensor agents must act as a team to cooperatively track the targets, there are some key difficulties in such tracking.

First, in order for a target to be tracked accurately, at least three agents must concurrently activate overlapping sectors. For example, in Figure 2 which corresponds to Figure 1.b, if an agent A1 detects a target 1 in its sector 0, it must coordinate with neighboring agents, A2 and A4 say, so that they activate their respective sectors that overlap with A1's sector 0. Activating a sector is an agent's operation. Since there are three sectors of 120 degrees, each agent has three operations. Since target 1 exists in the range of a sector for all agents, any combination of operations from three agents or all four agents can achieve the task of tracking target 1.

Second, there is ambiguity in selecting a sector to find a target.

R.Mostafaei Author, is with the Computer Engineering Department, Islamic Azad University Khoy,Iran (e-mail mostafaevn@iaukhoy.ac.ir)
A.Habiboghli Author, is with the Computer Engineering Department, Islamic Azad University Khoy,Iran (e-mail: habiboghli@iaukhoy.ac.ir)
M.R.Meybodi Author is with the Computer Engineering Department, Amirkabir University of Technology, Iran, (e-mail: meybodi@ce.aut.ac.ir).



(a) Sensor (left) and target (right) (b) simulator (top-down view)
Fig.1. A distributed sensor domain

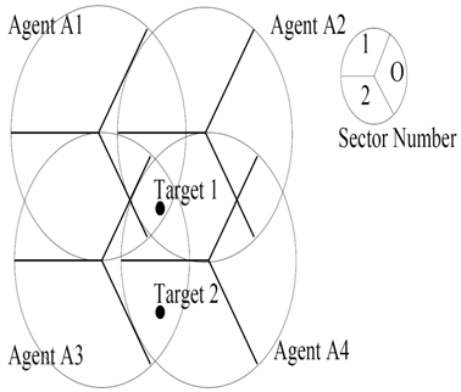


Fig.2. Each sensor (agent) has three sectors.

Since each sensor agent can detect only the distance and speed of a target, an agent that detects a target cannot tell other agents which sectors to activate. Assume that there is only target 1in Figure 2 and agent A1 detects the target first. A1 can tell A4 to activate sector 1.However, A1 cannot tell A2 which of the two sectors (sector 1 or sector 2) to activate since it only knows that there is a target in its sector 0. That is, agents don't know which task is to be performed. Identifying a task to perform depends on the result of other related agents' operations.

If there are multiple targets, a sensor agent may be required to activate more than one sector at the same time. For instance, in Figure 2, A4 needs to decide whether to perform either a task for target 1 or a task for target 2. Since at most one sector can be activated at a given time, A4 should decide which task to perform. Thus, the relationship among tasks to perform will affect the difficulty of the resource allocation problem.

Third, the situation is dynamic as targets move through the sensing range. The dynamic property of the domain makes problems even harder. Since target moves over time, after agents activate overlapping sectors and track a target, they may have to find different overlapping sectors.

The second domain which motivates our work is Robocup Rescue [2] for disaster rescue after an earthquake. Here, multiple Fire engines, ambulances and police cars must collaborate to save civilians from trapped; burning buildings and no centralized control is available to allocate all of the resources. For instance, an ambulance must collaborate with a fire engine and have a fire extinguished before it can rescue a civilian. The tasks are dynamic, e.g., fires grow or shrink and also ambiguous e.g., a fire engine could receive a report of a fire in an area, but not a specific building on fire. This domain thus presents another example of a distributed resource allocation problem with many similarities together with the distributed sensor network problem.

The above applications illustrate the difficulty of resource allocation among distributed agents in dynamic environment. Lack of formalism for dynamic distributed resource allocation problem can lead to ad-hoc methods which cannot be easily reused.

III. Dynamic DCSP

A Constraint Satisfaction Problem (CSP) is commonly defined by a set of variables, each associated with a finite domain, and a set of constraints on the values of the variables. A solution is the value assignment for the variables which satisfies all the constraints. A distributed CSP is a CSP in which variables and constraints are distributed among multiple agents. Each variable belongs to an agent. A constraint defined only on the variable belonging to a single agent is called a local constraint. In contrast, an external constraint involves variables of different agents. Solving a DCSP requires that agents not only solve their local constraints, but also communicate with other agents to satisfy external constraints.

DCSP assumes that the set of constraints are fixed in advance [5]. This assumption is problematic when we attempt to apply DCSP to domains where features of the environment are not known in advance and must be sensed at run-time. For example, in distributed sensor networks, agents do not know where the targets will appear. This makes it difficult to specify the DCSP constraints in advance. Rather, we desire agents to sense the environment and then activate or deactivate constraints depending on the result of the sensing action. We formalize this idea next.

We take the definition of DCSP one step further by defining Dynamic DCSP (DDCSP). DDCSP allows constraints to be conditional on some predicate P. More specifically, a dynamic constraint is given by a tuple (P, C), where P is an arbitrary predicate that is continuously evaluated by an agent and C is a familiar constraint in DCSP. When P is true, C must be satisfied in any DCSP solution. When P is false, C may be violated. An important consequence of dynamic DCSP is that agents no longer terminate when they reach a stable state. They must continue to monitor P, waiting to see if it changes. If its value changes, they may be required to search for a new solution. Note that a solution when P is true is also a solution when P is false, so

the deletion of a constraint does not require any extra computation. However, the converse does not hold. When a constraint is added to the problem, agents may be forced to compute a new solution. In this work, we only need to address a restricted form of DDCSP i.e. it is only necessary that local constraints be dynamic.

AWC [4] is a sound and complete algorithm for solving DCSPs. An agent with local variable Ai, chooses a value vi for Ai and sends this value to agents with whom it has external constraints. It then waits for and responds to messages. When the agent receives a variable value (Aj=vj) from another agent, this value is stored in an AgentView.

Therefore, an AgentView is a set of pairs {(Aj,vj),(Ak,vk),…}. Intuitively, the AgentView stores the current value of non-local variables. A subset of an AgentView is a NoGood if an agent cannot find a value for its local variable that satisfies all constraints. For example, an agent with variable Ai may find that the set {(Aj,vj),(Ak,vk)}is a NoGood because, given these values for Aj and Ak it cannot find a value for Ai that satisfies all of its constraints. This means that these value assignments cannot be part of any solution. In this case, the agent will request that the others change their variable value and a search for a solution continues. To guarantee completeness, a discovered NoGood is stored so that that assignment is not considered in the future.

The most straightforward way to attempt to deal with dynamism in DCSP is to consider AWC as a subroutine that is invoked anew every time a constraint is added. Unfortunately, in many domains such as ours, where the problem is dynamic but does not change drastically, starting from scratch may be prohibitively inefficient. Another option, and the one that we adopt, is for agents to continue their computation even if local constraints change asynchronously. The potential problem with this approach is that when constraints are removed, a stored NoGood may now become part of a solution. We solve this problem by requiring agents to store their own variable values as part of non-empty NoGoods. For example, if an agent with variable Ai finds that a value vi does not satisfy all constraints given the AgentView {(Aj,vj),(Ak,vk)}, it will store the set {(Ai,vi),(Aj,vj),(Ak,vk)} as a NoGood. With this modification to AWC, NoGoods remain "no good" even as local constraints change. Let us call this modified algorithm Locally-Dynamic AWC (LD-AWC) and the modified NoGoods "LD-NoGoods" in order to distinguish them from the original AWC NoGoods.

Lemma I: LD-AWC is sound and complete.

The soundness of LD-AWC follows from the soundness of AWC. The completeness of AWC is guaranteed by the recording of NoGoods. A NoGood logically represents a set of assignments that leads to a contradiction. We need to show that this invariant is maintained in LD-NoGoods. An LD-NoGood is a superset of some non-empty AWC NoGood and since every superset of an AWC NoGood is no good, the invariant is true when a LD-NoGood is first recorded. The only problem that remains is the possibility that an LD-NoGood may later become good due to the dynamism of local constraints. A LD-NoGood contains a specific value of the local variable that is no good but never contains a local variable exclusively. Therefore, it logically holds information about external constraints only. Since external constraints are not allowed to be dynamic in LD-AWC, LD-NoGoods remain valid even in the face of dynamic local constraints. Thus the completeness of LD-AWC is guaranteed.

IV. Generalized Mapping

In this section, we map the Class 3 Resource Allocation Problem, which subsumes Class 1 and 2, onto DDCSP [6]. Our goal is to provide a general mapping so that any resource allocation problem can be solved in a distributed manner by a set of agents by applying this mapping. Our mapping of the Resource Allocation Problem is motivated by the following idea. The goal in DCSP is for agents to choose values for their variable so all constraints are satisfied. Similarly, the goal in resource allocation is for the agents to choose operations so all tasks are performed. Therefore, in our first attempt we map variables to agents and values of variables to operations of agents. So for example, if an agent Ai has three operations it can perform, $(\{O_1^i, O_2^i, O_3^i\})$ then the variable corresponding to this agent will have three values in its domain. However, this simple mapping attempt fails because an operation of an agent may not always succeed. Therefore, in our second attempt, we define two values for every operation, one for success and the other for failure. In our example, this would result in six values.

It turns out that even this mapping is inadequate for the Class 2 and 3 Resource Allocation Problem. This is because an operation can be required for more than one task. We desire agents to be able to not only choose which operation to perform, but also to choose for which task they will perform the operation. For example in Figure 2, Agent A3 is required to active the same sector for both targets 1 and 2. We want A3 to be able to distinguish between the two targets, so that it does not unnecessarily require A2 to activate sector 2 when target 2 is present. So, for each of the values defined so far, we will define new values corresponding to each task that an operation may serve.

V. Proposed Idea in Target Tracking

As discussed in section II, if each sensor is controlled by many agents, for tracking one target, all of related sensors should be cooperate and coordinated. This point causes many problems. For example, each agent may know the results of its local operations but it may not know the global task and may not know what operations others should perform. Since the situation is dynamic, so a solution to the resource allocation problem at one time may become unsuccessful when the underlying tasks have changed because the targets are mobile, however, fault tolerance degree of this system is low. For more clarity, suppose the sensor network of figure2, if A1 and A4 fail simultaneously, the network cannot work well and

perform its tasks correctly because for tracking one target, at least three agents should cooperate for activating of corresponding sectors.

By attempting the problems discussed above, as we can see in figure3, we propose the framework that differ from others and solve these problems (it is like the previous frame work in some cases). This framework is based on DCSP, too; but prevents the occurrence of previous DCSP based problems.

Our proposed method performs as follow:

Agents (sensors) are variables; domain of each variable contains all of sensed targets by each sensor and distance of them with corresponding sensors because as mentioned in section II, each sensor can detect the speed and distance of targets. Constraints are divided in two groups: internal constraints and external constraints. Just like previous approach since internal constraints are dynamic, specify them with tuple (P, C) in which C is a familiar constraint in DCSP and explain the target that should be selected and has the smallest distance from corresponding sensor. P is arbitrary predicate and checked at each time to ensure that selected target be in the domain of agent. Since targets are mobile therefore they can exit from sensors domain. In this case, P should set to False, thereby, constraint C automatically is not satisfied and then selected target delete from sensors domain. External constraints specify that selected target by one of sensors can be selected by others again. The solution is that all of targets should be tracked successfully.

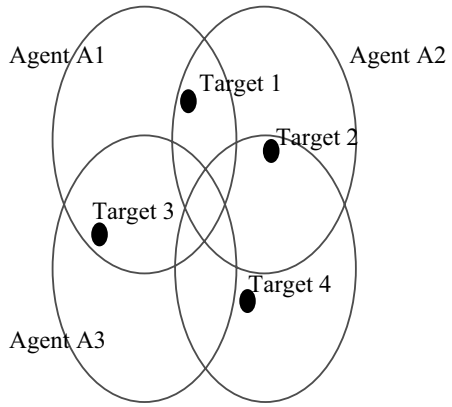


Fig.3. each sensor (agent) contributed from only one sector

For solving resource allocation problem we use backtracking and constraint propagation together as follows: For each variable which satisfies its internal constraint (C), choose targets that have smallest distance from corresponding sensor and check at any that P be true. So constraint propagation is used for satisfying external constraint, it means that each agent should send the message, which contains the name and distance of selected target, for other neighboring agents. On the other hand, neighboring agents receive this message and compare the distance of selected target with its domains, if this parameter is low, it omit from there domains and try next. Denote that each sensor only can track one target at each time. If all of targets are tracked successfully solution is found and task is performed successfully. Otherwise, backtracking to the previous state occurs and other target from domain will be selected and this routine will be repeated again.

VI. Experimental Results

In this point of the paper we compare two DCSP based methods, previous method and our proposed method. Both of methods are evaluated in a sensor network with 10 and 500 nodes respectively. We have considered two parameters for this comparison, number of tracked targets and fault tolerance rate. As we can see in figure 4, the number of tracked targets in our proposed methods is more than previous method; the reason is that in our method the number of operations of each agent is reduced. Therefore, average number of tracked targets in a sensor network with 10 nodes is about 5.5 targets for proposed method and 2.9 targets to other.

Also, we generalized our simulation to a sensor network with 500 nodes. As we can see in figure.6, the average number of tracked targets for proposed method is about 250 targets. However, this rate for it is about 130 targets, it means that proposed method is twice better than previous method.

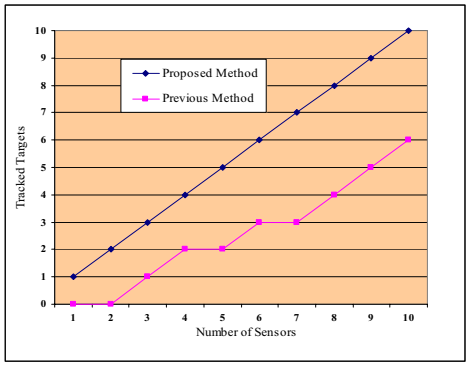


Fig.4. Number of tracked targets in sensor network with 10 nodes

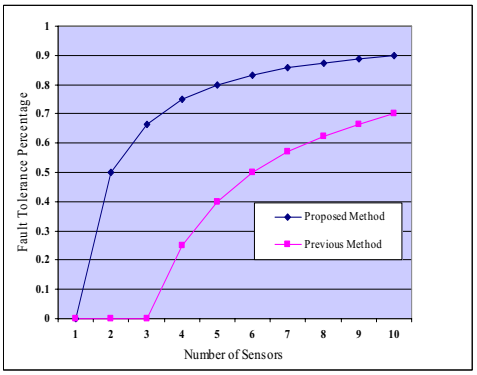


Fig.5. Fault Tolerance in sensor network with 10 nodes

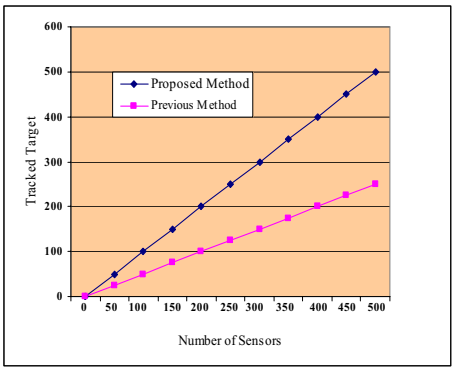


Fig.6. Number of tracked targets in sensor network with 500 nodes

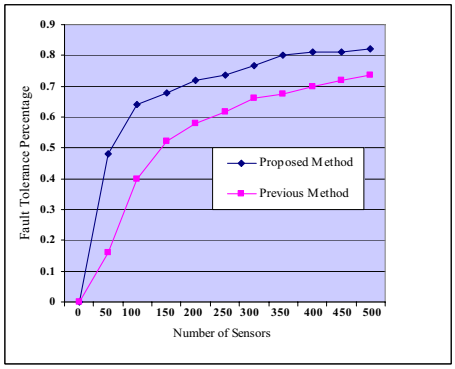


Fig.7. Fault Tolerance in sensor network with 500 nodes

This is the fact that in many applications fault tolerance of the systems is important parameter and even is emergence. It means that the system should be able to perform its tasks despite occurring hardware or software faults. We compare both of methods from this perspective. As figure5 shows, fault tolerance rate of proposed method is more than previous method because any of agents for performing tracking task has less dependency to other agents. Experimental results show that in a sensor network with 10 nodes, fault tolerance percentage for proposed method is about 70% and 39% for previous method. In a sensor network with 500 nodes, as we can see in figure7, this rate is about 66% for proposed method and about 52% for previous method.

VII. Conclusion

In this paper we use a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem and compare two different methods. The second method is our proposed method for improving target tracking task. As mentioned above, since proposed method does not require to is cooperated of many agents, response time of the system really is reduced and tracking task performs faster than others. Also we have not any problem for tracking mobile targets because this task can perform easily with checking the condition P of internal constraint at each time. Failing one or more agent causes weak performance of the network since other sound agents can not work properly; this is the next problem of previous method. However, we improve this weakness of previous method in proposed method. Since proposed method is only controlled by its agent, failing one sensor does not affect working of others. So our proposed method is fault tolerant.

References

[1]. K.Decker and J. Li. Coordinated hospital patient scheduling. In *ICMAS*, 1998.
[2]. Hiroaki Kitano. Robocup rescue: A grand challenge for multi-agent systems. In *CMAS*, 2000.
[3]. Sanders. Ecm challenge problem, http://www.sanders.com/ants/ecm.htm. 2001.
[4]. M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *ICMAS*, July 1998.
[5]. C. Frei and B. Faltings. Resource allocation in networks using abstraction and constraint satisfaction techniques. In *Proc of Constraint Programming*, 1999.
[6]. Pragnesh Jay Modi et all. Dynamic Distributed Resource Allocation: A Distributed Constraint Satisfaction Approach. *University of Southern California, 2003.*