# Dynamic Distributed Resource Allocation: A Distributed Constraint Satisfaction Approach

**Pragnesh Jay Modi, Hyuckchul Jung, Milind Tambe, Wei-Min Shen, Shriniwas Kulkarni**
University of Southern California/Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{modi,jungh,tambe,shen,kulkarni}@isi.edu

## 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 proposing a formalization of distributed resource allocation that represents both dynamic and distributed aspects of the problem. We define four categories of difficulties of the problem. To address this formalized problem, the paper defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DDCSP). 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 an implementation on a real-world distributed sensor network.

## 1 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 [7], disaster rescue[4], hospital scheduling[2], 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 propose a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. 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 define four classes of difficulties of the problem. In order to address the resource allocation problem, the paper also defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DDCSP).

The central contribution of the paper is a reusable, 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 an implementation on a real-world distributed sensor network. Ideally, our formalization may enable researchers to understand the difficulty of their resource allocation problem, choose a suitable mapping using DDCSP, with automatic guarantees for correctness of the solution.

## 2 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 illustrates the real hardware and simulator screen, respectively). Each sensor is equipped with a Doppler radar with 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. Since each sensor agent can detect only the distance and speed of a target, an agent that detects a target cannot

(a) sensor(left) and target(right) (b) simulator (top-down view)

**Figure 1:** A distributed sensor domain

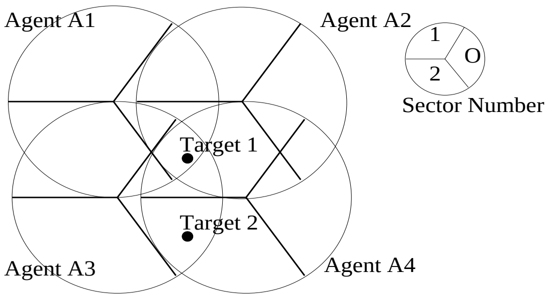

**Figure 2:** Each sensor (agent) has three sectors.

tell other agents which sectors to activate. Assume that there is only target 1 in 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 sectors 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 [4] 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 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 with the distributed sensor network problem.

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

## 3 Formalization

A Dynamic Distributed Resource Allocation Problem is a structure $<\mathcal{A}g$, $\Omega$, $\Theta>$ where

- $\mathcal{A}g$ is a set of agents, $\mathcal{A}g = \{A_1, A_2, ..., A_n\}$.
- $\Omega = \{O^1_1, O^1_2, ..., O^i_p, ..., O^n_p\}$ is a set of operations, where operation $O^i_p$ denotes the p'th operation of agent $A_i$. Let $Op(A_i)$ denote the set of operations of $A_i$. Operations in $Op(A_i)$ are *mutually exclusive*; an agent can only perform one operation at a time.
- $\Theta$ is a set of tasks, where a task is a collection of sets of operations that satisfy the following properties:
(i) $\forall T \in \Theta, T \subseteq P(\Omega)$
(ii) $T$ is nonempty and, $\forall t \in T$, $t$ is nonempty;
(iii) $\forall t_r, t_s \in T, t_r \not\subseteq t_s$ and $t_s \not\subseteq t_r$.

Property (iii) requires that each set of operations in a task should be minimal in the sense that no other set is a subset of it. For instance, in Figure 2, a set of three operations is necessary and sufficient to track target 1. That is, $\{A_1$'s activating sector 0, $A_2$'s activating sector 2, $A_4$'s activating sector $1\}$ is a minimal set of operations for the task of tracking target 1. For each task, we use Max($T_r$) to denote the union of all the minimal sets of $T_r$. More formally,

- $\forall\, T_r \in \Theta$, Max$(T_r) = \bigcup_{t_r \in T_r} t_r$

For instance, in Figure 2, target 1 can be tracked by four separate sets of operations as described in Section 2. Here, if we use $O^i_p$ to denote the operation of $A_i$'s activating sector $p$, target 1 can be also tracked by the union of the four sets, Max$(T_1) = \{O^1_0, O^2_2, O^3_0, O^4_1\}$.

We use $T(O^i_p)$ to denote the tasks for which $O^i_p$ is required:

- $\forall\, O^i_p \in \Omega$, $T(O^i_p) = \{T_r \mid O^i_p \in \text{Max}(T_r)\}$

We require that $\forall\, O^i_p \in \Omega$, $\mid T(O^i_p) \mid \neq 0$. That is, every operation should serve some tasks.

All the tasks in $\Theta$ are not always present. We use $\Theta_{current}$ to denote the set of tasks that are currently present. This set is determined by the environment. Agents can execute their operations at any time, but the success of an operation is determined by the set of tasks that are currently present. More formally,

- **Definition 1**: $\forall\, O^i_p \in \Omega$, if $O^i_p$ is executed and $\exists\, T_r \in \Theta_{current}$ such that $O^i_p \in$ Max$(T_r)$, then $O^i_p$ is said to *succeed*.

A task is performed when all the operations in some minimal set succeed. More formally,

- **Definition 2**: $\forall T_r \in \Theta$, $T_r$ is *performed* iff $\exists t_r \in T_r$ such that all the operations in $t_r$ succeed.

In Robocup Rescue domain, if a rescue task requires a set of operations from two fire engines, an ambulance, and a police car, all the required operations should succeed for the success of the rescue task.

We assume that no mutually exclusive operations are required for the same task just as, in sensor network problem, each sensor agent cannot activate more than one sector to perform one task. Thus, we have the following requirement:

- $\forall T_r \in \Theta$ and $\forall A_i \in \mathcal{A}g$, $\mid$ Max$(T_r) \cap$ Op$(A_i) \mid \leq 1$.

To eliminate the possibility of performing phantom tasks, we assume that a performed task should be a task in the current task set:

- **No phantom tasks assumption**: $\forall T_r \in \Theta$, if $T_r$ is performed, then $T_r \in \Theta_{current}$.

We also assume that at least one agent is notified of the existence of a current task by having its corresponding operation succeed. This is reasonable because the task will fail if no agent is notified.

- **Notification assumption**: $\forall T_r \in \Theta$, if $T_r \in \Theta_{current}$, then $\exists$ $O_p^i \in$ Max$(T_r)$ such that $O_p^i$ succeeds.

We now define some properties of a given resource allocation problem. Multiple tasks in $\Theta_{current}$ can be performed concurrently with some conditions. For instance, two different tasks $T_1$ and $T_2$ can be performed concurrently if all the operations for performing $T_1$ can be executed concurrently with those for performing $T_2$. We define two types of *conflict-free* to denote tasks that can be performed concurrently from $\Theta_{current}$.

- **Definition 3**: $\forall$ $T_r, T_s \in \Theta_{current}$, $T_r$ and $T_s$ are *strongly conflict free* iff the following statement is true:
  if $T_r \neq T_s$, then $\forall$ $A_i \in \mathcal{A}g$, $\mid$ Max$(T_r) \cap Op(A_i) \mid$ +
  $\mid$ Max$(T_s) \cap Op(A_i) \mid \leq 1$ holds.

The condition implies that an agent can serve at most one current task.

- **Definition 4**: $\forall$ $T_r, T_s \in \Theta_{current}$, $T_r$ and $T_s$ are *weakly conflict free* iff the following statement is true:
  if $T_r \neq T_s$, then there exist $t_r \in T_r$ and $t_s \in T_s$ s.t. $\forall$ $A_i \in \mathcal{A}g$, $\mid t_r \cap Op(A_i) \mid$ + $\mid t_s \cap Op(A_i) \mid \leq 1$ holds.

Here, the condition implies that, for two current tasks, there exist two sets of operations that do not require mutually exclusive operations from an agent.

When there is no centralized control, no agent knows what the current task is. When an operation $O_p^i$ of an agent $A_i$ succeeds, the agent only knows that there is an unidentified task to be performed from a task set $\Theta$. If $\mid T(O_p^i) \mid$ = 1, there is no ambiguity in identifying a task to be performed. $A_i$ whose operation succeeds can look up the task and inform other related agents of what the current task is. However, there can be multiple tasks for which $O_p^i$ is required. If $\mid T(O_p^i) \mid > 1$, $A_i$ has to decide what the task is in a cooperative way with other agents. A task $T_r \in T(O_p^i)$ can be identified when all the operations in an operation set $t_r \in T_r$ succeed. Since operations from different agents are involved, the decision problem cannot be solved by an individual agent.

The Dynamic Distributed Resource Allocation Problem is to identify current tasks that can change over time and assign operations that are required by the current tasks. The difficulty of the problem depends on both the ambiguity of the tasks from a dynamic distributed environment and the relation among tasks which may require conflicting resources. Here, we outline four problem classes of increasing difficulty based on ambiguity and the relation among tasks.

- **Class 1 (static, strongly conflict free, no ambiguity)**: In this class of problems, the tasks in $\Theta_{current}$ fixed, but unknown, and are *strongly conflict free*. Furthermore, $\forall O_h^i \in \Omega$, $\mid T(O_h^i) \mid$ = 1, so there is no ambiguity in identifying a task for when an operation succeeds.
- **Class 2 (static, strongly conflict free, ambiguity)**: In this class of problems, the tasks in $\Theta_{current}$ fixed, but unknown, and are *strongly conflict free*. Furthermore, $\forall O_h^i \in \Omega$, $\mid T(O_h^i) \mid > 1$, so there is ambiguity in identifying a task for when an operation succeeds.
- **Class 3 (dynamic, strongly conflict free, ambiguity)**: Dynamism means that $\Theta_{current}$ changes over time. In other words, the set of tasks that must be performed by the agents is dynamic.
- **Class 4 (dynamic, weakly conflict free, ambiguity)**: By relaxing *strongly conflict free* tasks assumption, we make agents' decision even harder. With *weakly conflict free* tasks, agents may need to select an operation among mutually exclusive operations. Agents should negotiate to find a solution without involving mutually exclusive operations for each task in $\Theta_{current}$.

As the number of class increases, the problem hardness also increases. A solution method can solve any problem in lower classes.

## 3.1 Application of formalism

To illustrate this formalism in the distributed sensor network domain, we cast each sensor as an agent and activating a sector is an operation. A task then, is a set of minimal sets of operations that can track a target. For example, in Figure 2, target 1 can be tracked by four different sets of operations:$\{O_0^1, O_2^2, O_0^3\}$, $\{O_2^2, O_0^3, O_1^4\}$, $\{O_0^1, O_0^3, O_1^4\}$, and $\{O_0^1, O_2^2, O_1^4\}$. We define a task for each target in an area of overlap of sectors. For the situation illustrated in Figure 2, $\Theta_{current}$ = $\{T_1, T_2\}$, $T_1$ = $\{\{O_0^1, O_2^2, O_0^3\}, \{O_2^2, O_0^3, O_1^4\}, \{O_0^1, O_0^3, O_1^4\}, \{O_0^1, O_2^2, O_1^4\}\}$, $T_2$ = $\{\{O_0^3, O_2^4\}\}$.

When an operation that is required for multiple task succeeds, an agent may not know which of those tasks are actually present. Suppose that only target 1 is in Figure 2 and $A_3$ is the first agent who detects the target 1. After $A_3$'s operation $O_0^3$ is successfully executed, $A_3$ cannot identify the current task ($T_1$ above) among the tasks whose element includes $O_0^3$: for instance, $O_0^3$ is included in both $T_1$ and $T_2$. $A_3$ and other related agents should cooperatively find the current task and solve the resource allocation problem.

# 4 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. 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 [8] is a sound and complete algorithm for solving DCSPs. An agent with local variable $A_i$, chooses a value $v_i$ for $A_i$ and sends this value to agents with whom it has external constraints. It then waits for and respond to messages. When the agent receives a variable value ($A_j = v_j$) from another agent, this value is stored in an AgentView. Therefore, an AgentView is a set of pairs $\{(A_j, v_j)$, $(A_k, v_k)$, ...$\}$. 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. Whenever a NoGood is found, it is stored so that those assignments are not considered in the future. For example, an agent with variable $A_i$ may find that the set $\{(A_j, v_j)$, $(A_k, v_k)\}$ is a NoGood because, given these values for $A_j$ and $A_k$, it cannot find a value for $A_i$ that satisfies all constraints, In this case, AWC will store the set $\{(A_j, v_j)$, $(A_k, v_k)\}$. This means that these value assignments cannot be part of any solution.

The most straightforward way to attempt to deal with dynamism in DCSP is to consider AWC as a subroutine that is invoked anew everytime 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 as the constraints change asynchronously. A 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 allowing agents to store their own variable values as part of NoGoods. For example, if an agent with variable $A_i$ finds that a value $v_i$ does not satisfy all constraints given the AgentView $\{(A_j, v_j)$, $(A_k, v_k)\}$, it will store the set $\{(A_i, v_i)$, $(A_j, v_j)$, $(A_k, v_k)\}$ as a NoGood. With this modification to AWC, NoGoods remain "no good" even as local constraints change.

## 5 Generalized Mapping

In this section, we map the Class 3 Resource Allocation Problem, which subsumes Class 1 and 2, onto DDCSP. 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 $A_i$ has three operations it can perform, $\{O^i_1, O^i_2, O^i_3\}$, 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.

More formally, given a Class 3 Resource Allocation Problem $\langle \mathcal{A}g, \Omega, \Theta \rangle$, the corresponding DCSP is defined over a set of $n$ variables,

- $A = \{A_1, A_2,..., A_n\}$, one variable for each $A_i \in$ Ag. We will use the notation $A_i$ to interchangeably refer to an agent or its variable.

The domain of each variable is given by:

- $\forall A_i \in \mathcal{A}g$, Dom($A_i$) = $\bigcup_{O^i_p \in \Omega} O^i_p$x$T(O^i_p)$x{yes,no}.

.

In this way, we have a value for every combination of operations an agent can perform, a task for which this operation is required, and whether the operation succeeds or fails. For example in Figure 2, Agent A3 has two operations (sector 1 and 2) with only one possible task (target) and one operation (sector 0) with two possible tasks (target 1 and 2). This means it would have 8 values in its domain.

A word about notation: $\forall\ O^i_p \in \Omega$, the set of values in $O^i_p$x$T(O^i_p)$x{yes} will be abbreviated by the term $O^i_p$*yes and the assignment $A_i = O^i_p$*yes denotes that $\exists v \in O^i_p$*yes s.t. $A_i = v$. Intuitively, the notation is used when an agent detects that an operation is succeeding, but it is not known which task is being performed. This analagous to the situation in the distributed sensor network domain where an agent may detect a target in a sector, but not know its exact location. Finally, when a variable $A_i$ is assigned a value, we assume the corresponding agent is required to execute the corresponding operation.

Next, we must constrain agents to assign "yes" values to variables only when an operation has succeeded. However, in Class 3 problems, an operation may succeed at some time and fail at another time since tasks are dynamically added and removed from the current set of tasks to be performed. Thus, every variable is constrained by the following dynamic local constraints.

- **Dynamic Local Constraint 1 (LC1)**: $\forall T_r \in \Theta$, $\forall O_p^i \in$ Max($T_r$), we have LC1($A_i$) = (P, C), where

  P: $O_p^i$ succeeds.

  C: $A_i = O_p^i$*yes
- **Dynamic Local Constraint 2 (LC2)**: $\forall T_r \in \Theta$, $\forall O_p^i \in$ Max($T_r$), we have LC2($A_i$) = (P, C), where

  P: $O_p^i$ does not succeed.

  C: $A_i \neq O_p^i$*yes

The truth value of P is not known in advance. Agents must execute their operations, and based on the result, locally determine if C needs to be satisfied. In the Class 1 and 2 problems, the set of current tasks does not change and thus, the truth value of P, although initially unknown, once known will not change over time. On the other hand, in the Class 3 and 4 problems, where the set of current tasks is changing over time, the truth value of P will change, and hence the corresponding DCSP will be truly dynamic.

We now define the external constraint (EC) between variables of two different agents. EC is a normal static constraint and is always present.

- **External Constraint**: $\forall T_r \in \Theta$, $\forall O_p^i \in$ Max($T_r$), $\forall A_j \in A$,

  EC($A_i$, $A_j$):
  (1) $A_i$ = $O_p^i T_r$yes, and
  (2) $\forall t_r \in T_r$ s.t. $O_p^i \in t_r$, $\exists q$ s.t. $O_q^j \in t_r$.
  $\Rightarrow A_j$ = $O_q^j T_r$yes

The EC constraint requires some explanation. Condition (1) states that an agent $A_i$ has found an operation that succeeds for task $T_r$. Condition (2) quantifies the other agents whose operations are also required for $T_r$. If $A_j$ is one of those agents, the consequent requires it to choose its respective operation for the $T_r$. If $A_j$ is not required for $T_r$, condition (2) is false and EC is trivially satisfied. Finally, note that every pair of variables $A_i$ and $A_j$, have two EC constraints between them: one from $A_i$ to $A_j$ and another from $A_j$ to $A_i$. The conjunction of the two unidirectional constraints can be considered one bidirectional constraint.

We will now prove that our mapping can be used to solve any given Class 3 Resource Allocation Problem. The first theorem shows that our DDCSP always has a solution, and the second theorem shows that if agents reach a solution, all current tasks are performed. It is interesting to note that the converse of the second theorem does not hold, i.e. it is possible for agents to be performing all tasks *before* a solution state is reached. This is due to the fact that when all current tasks are being performed, agents whose operations are not necessary for the current tasks could still be violating constraints.

**Theorem I** : Given a Class 3 Resource Allocation Problem $\langle \mathcal{A}g,\Omega,\Theta\rangle$, $\Theta_{current} \subseteq \Theta$, there exists a solution to the corresponding DDCSP.

*proof:* We proceed by presenting a variable assignment and showing that it is a solution.

Let $B = \{A_i \in A \mid \exists T_r \in \Theta_{current}, \exists O_p^i \in \text{Max } (T_r)\}$. We will first assign values to variables in $B$, then assign values to variables that are not in $B$. $\forall A_i \in A$: if $A_i \in B$, by the def of B, $\exists T_r \in \Theta_{current}, \exists O_p^i \in$ Max($T_r$). In our solution, we assign $A_i = O_p^i T_r yes$. If $A_i \notin B$, we may choose any $O_p^i T_r no \in$ Dom($A_i$) and assign $A_i = O_p^i T_r no$.

To show that this assignment is a solution, we first show that it satisfies the EC constraint. We arbitrarily choose two variables, $A_i$ and $A_j$, and show that EC($A_i$, $A_j$) is satisfied. We proceed by cases.

Let $A_i, A_j \in A$ be given.

*case 1:* $A_i \notin B$
Since $A_i = O_p^i T_r no$, condition (1) of EC constraint is false and thus EC is trivially satisfied.

*case 2:* $A_i \in B, A_j \notin B$
$A_i = O_p^i T_r yes$ in our solution. Let $t_r \in T_r$ s.t. $O_p^i \in t_r$. We know that $T_r \in \Theta_{current}$ and since $A_j \notin B$, we conclude that $\nexists O_q^j \in t_r$. So condition (2) of the EC constraint is false and thus EC is trivially satisfied.

*case 3:* $A_i \in B, A_j \in B$
$A_i = O_p^i T_r yes$ and $A_j = O_q^j T_s yes$ in our solution. Let $t_r \in T_r$ s.t. $O_p^i \in t_r$. Since $T_s$ and $T_r$ are both in $\Theta_{current}$ and therefore strongly conflict free, $\nexists\, O_n^j \in \Omega$ s.t. $O_n^j \in t_r$. So condition (2) of EC($A_i$,$A_j$) is false and thus EC is trivially satisfied.

Next, we show that our assignment satisfies the LC constraints. If $A_i \in B$ then $A_i = O_p^i T_r yes$, and LC1, regardless of the truth value of P, is clearly not violated. Furthermore, it is the case that $O_p^i$ succeeds, since $T_r$ is present. Then the precondition P of LC2 is not satisfied and thus LC2 is not present. If $A_i \notin B$ and $A_i = O_p^i T_r no$, it is the case that $O_p^i$ is executed and, by definition, does not succeed. Then the precondition P of LC1 is not satisfied and thus LC1 is not present. LC2, regardless of the truth value of P, is clearly not violated. Thus, the LC constraints are satisfied by all variables.

We can conclude that all constraints are satisfied and our value assignment is a solution to the DDCSP. □

**Theorem II** : Given a Class 3 Resource Allocation Problem $\langle \mathcal{A}g,\Omega,\Theta\rangle$, $\Theta_{current} \subseteq \Theta$ and the corresponding DDCSP, if an assignment of values to variables in the DDCSP is a solution, then all tasks in $\Theta_{current}$ are performed.

*proof sketch:* Let a solution to the DDCSP be given. We want to show that all tasks in $\Theta_{current}$ are performed. We proceed by choosing a task $T_r \in \Theta_{current}$. If we can show that it is indeed performed and since our choice is arbitrary, we can conclude that all members of $\Theta_{current}$ are performed.

Let $T_r \in \Theta_{current}$. By the **Notification Assumption**, some operation $O_p^i$, required by $T_r$ will be executed. However, the corresponding agent $A_i$, will be unsure as to which task it is performing when $O_p^i$ succeeds. This is due to the fact that $O_p^i$ may be required for many different tasks. It may randomly choose a task, $T_s \in T(O_p^i)$, and LC1 requires it to assign the value $O_p^i T_s$yes. The EC constraint will then require that all other agents $A_j$, whose operations are required for $T_s$ also execute those operations and assign

| Num nodes | 4 | 6 | 8 |
|---|---|---|---|
| one target | 57,6 (9.50) | 190,23 (8.26) | 247,35 (7.05) |
| two targets | – | 91,13 (7.00) | 244,29 (8.41) |

**Table 1:** Results from Sensor Network Domain.

$A_j = O_q^j T_s yes$. We are in solution, so LC2 cannot be present for $A_j$. Thus, $O_q^j$ succeeds. Since all operations required for $T_s$ succeed, $T_s$ is performed. By the **No Phantom Tasks Assumption**, $T_s \in \Theta_{current}$. But since we already know that $T_s$ and $T_r$ have an operation in common, the Strongly Conflict Free condition requires that $T_s = T_r$. Therefore, $T_r$ is indeed performed. $\square$

Given our formalization of dynamic environments, when tasks change, this implies changes only in local constraints of agents, and hence the theorems are able to address dynamic changes in tasks. Class 4 tasks require dynamism in external constraints as well, and this is not handled and an issue for future work.

## 6 Experiments in a Real-World Domain

We have successfully applied the DDCSP approach to the distributed sensor network problem, using the mapping introduced in Section 3. Indeed, in recent evaluation trials conducted in government labs in August and September 2000, this DDCSP implementation was successfully tested on four actual hardware sensor nodes (see Figure 1.a), where agents collaboratively tracked a moving target. This target tracking requires addressing noise, communication failures, and other real-world problems; although this was done outside the DDCSP framework and hence not reported here.

The unavailability of the hardware in our lab precludes extensive hardware tests; but instead, a detailed simulator that very faithfully mirrors the hardware has been made available to us. We have done extensive tests using this simulator to further validate the DDCSP formalization: indeed a single implementation runs on both the hardware and the simulator. One key evaluation criteria for this implementation is how accurately it is able to track targets, e.g., if agents do not switch on overlapping sectors at the right time, the target tracking has poor accuracy. Here, the accuracy of a track is measured in terms of the *RMS* (root mean square) error in the distance between the real position of a target and the target's position as estimated by a team of sensor agents. Our results here — assuming a square sensor node configuration of Figure 1.b — are as follows. Domain experts expect an RMS of approx 3 units, and thus they believe these results are satisfactory. Our RMS for Class 1 problems is 0.9 units, and for Class 3 problems, the RMS is 3.5 units.

Table 1 presents further results from the implementation. Experiments were conducted for up to 8 nodes solving Class 3 problems. Each cell in the table presents the number of messages exchanged, the number of sector changes, and in parenthesis, the number of messages exchanged per sector change. For instance, in configuration involving one dynamic target and 6 nodes, agents exchanged 190 messages, for 23 sector changes, i.e., 8.23 message per sector change. More nodes involve more sector changes, since a dynamic target passes through regions covered by different nodes, and the nodes must search for this target to pin down its location.

The key results here are that: (i) The dynamic DCSP algorithm presented in Section 4 is able to function correctly; (ii) increasing number of nodes does not result in increasing numbers of messages per sector change, providing some evidence for scalability of our mapping.

## 7 Summary and Related Work

In this paper, we proposed a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. We define four categories of difficulties of the problem and address these formalized problems by defining the notion of Dynamic Distributed Constraint Satisfaction Problem (DDCSP). The central contribution of the paper is a generalized mapping from distributed resource allocation to DDCSP. Through both theoretical analysis and experimental verifications, we have shown that this approach to dynamic and distributed resource allocation is powerful and unique, and can be applied to real-problems such as the Distributed Sensor Network Domain.

In terms of related work, there is significant research in the area of distributed resource allocation; for instance, Liu and Sycara's work[5] extends dispatch scheduling to improve resource allocation. Chia et al's work on distributed vehicle monitoring and general scheduling (e.g. airport ground service scheduling) is well known but space limits preclude us from a detailed discussion [1]. However, a formalization of the general problem in distributed settings is yet to be forthcoming. Our work takes a step in this direction and provides a novel and general DDCSP mapping, with proven guarantees of performance. Some researchers have focused on formalizing resource allocation as a centralized CSP, where the issue of ambiguity does not arise[3]. The fact that resource allocation is distributed and thus ambiguity must be dealt with, is a main component of our work. Furthermore, we provide a mapping of the resource allocation problem to DDCSP and prove its correctness, an issue not addressed in previous work. Dynamic Constraint Satisfaction Problem has been studied in the centralized case by [6]. However, there is no distribution or ambiguity during the problem solving process. The work presented here differs in that we focus on distributed resource allocation, its formalization and its mapping to DDCSP. Indeed, in the future, our formalization may enable researchers to understand the difficulty of their resource allocation problem, choose a suitable mapping using DDCSP, with automatic guarantees for correctness of the solution.

## Acknowledgements

This research is sponsored in part by DARPA/ITO under contract number F30602-99-2-0507, and in part by AFOSR under grant number F49620-01-1-0020.

## REFERENCES

[1] M. Chia, D. Neiman, and V. Lesser. Poaching and distraction in asynchronous agent activities. In *ICMAS*, 1998.

[2] K. Decker and J. Li. Coordinated hospital patient scheduling. In *ICMAS*, 1998.

[3] C. Frei and B. Faltings. Resource allocation in networks using abstraction and constraint satisfaction techniques. In *Proc of Constraint Programming*, 1999.

[4] Hiroaki Kitano. Robocup rescue: A grand challenge for multi-agent systems. In *ICMAS*, 2000.

[5] J. Liu and K. Sycara. Multiagent coordination in tightly coupled task scheduling. In *ICMAS*, 1996.

[6] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *AAAI*, 1990.

[7] Sanders. Ecm challenge problem, http://www.sanders.com/ants/ecm.htm. 2001.

[8] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *ICMAS*, July 1998.