A Dynamic Distributed Constraint Satisfaction Approach to Resource Allocation

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 disaster rescue, hospital scheduling and the domain described in this paper: distributed sensor networks. Despite the variety of approaches proposed for distributed resource allocation, a systematic formalization of the problem and a general solution strategy are missing. This paper takes a step towards this goal by proposing a formalization of distributed resource allocation that represents both dynamic and distributed aspects of the problem and a general solution strategy that uses distributed constraint satisfaction techniques. This paper defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDCSP) and proposes two generalized mappings from distributed resource allocation to DyDCSP, each proven to correctly perform resource allocation problems of specific difficulty and 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 perform operations and assign their resources to a set of tasks such that all tasks are performed. This problem arises in many real-world domains such as distributed sensor networks [7], disaster rescue[4], hospital scheduling[2], and others. Resource allocation problems of this type are difficult because they are both *distributed* and *dynamic*. A key implication of the distributed nature of this problem is that the control is distributed in multiple agents; yet these multiple agents must collaborate to accomplish the tasks at hand. Another implication is that the multiple agents each obtain only local information, and face global ambiguity — an agent may know the results of its local operations but it may not know which other collaborators must be involved to fulfill the global task and which operations these collaborators must perform for success. Finally, the situation is dynamic so a solution to the resource allocation problem at one time may become obsolete when the underlying tasks have changed. This means that once a solution is obtained, the agents must continuously monitor it for changes and must have a way to express such changes in the problem.

In this paper, we first propose a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. This formalization allows us to understand the complexity of different types of resource allocation problems. Second, in order to address this type of resource allocation problem, we define the notion of a Dynamic Distributed Constraint Satisfaction Problem

(DyDCSP). DyDCSP is a generalization of DCSP (Distributed Constraint Satisfaction Problem) [8] that allows constraints to be added or removed from the problem as external environmental conditions change. Third, we present two reusable, generalized mappings from distributed resource allocation to DyDCSP, each proven to correctly perform resource allocation problems of specific difficulty and experimentally verified through implementation in a real-world application. In summary, our central contribution is 1) a formalization that may enable researchers to understand the difficulty of their resource allocation problem and 2) generalized mappings to DyDCSP which provide automatic guarantees for correctness of the solution.

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. 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 means that ambiguity must be dealt with. 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.

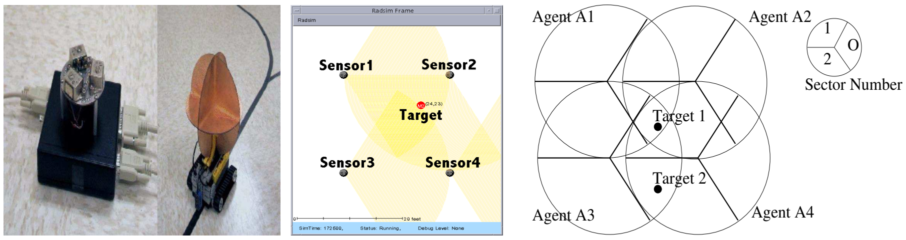
The paper is structured as follows: Section 2 describes the application domain of our resource allocation problem and Section 3 presents a formal model and defines subclasses of the resource allocation problem. Section 4 introduces Dynamic Distributed Constraint Satisfaction Problems. Then, Sections 5 and 6 describe solutions to subclasses of resource allocation problems of increasing difficulty, by mapping them to DyDCSP. Section 7 describes empirical results and Section 8 concludes.

2 Application Domain

The domain in which this work has been applied 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) 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 collaborate — concurrently activating overlapping sectors. For example, in Figure 1.b which corresponds to the simulator in Figure 1.a, 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 tell other agents which sectors to activate. When there is only target 1 in Figure 1.b and agent A1 detects the target first, A1 can tell A4 to activate sector 1. However,



(a) hardware and simulator (b) sensor sectors

Fig. 1. A distributed sensor domain

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.

Third, if there are multiple targets, that introduces resource contention — an agent may be required to activate more than one sector, which it can not! For instance, in Figure 1.b, 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 will affect the difficulty of the resource allocation problem.

Fourth, 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 above application 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. On the other hand, adopting a formal model allows our problem and its solution to be stated in a more general way, possibly increasing our solution's usefulness. More importantly, a formal treatment of the problem also allows us to study its complexity and provide other researchers with some insights into the difficulty of their own resource allocation problems. Finally, a formal model allows us to provide guarantees of soundness and completeness of our results. The next section presents our general, formal model of resource allocation.

3 Formalization of Resource Allocation

A Distributed Resource Allocation Problem consists of 1) a set of agents that can each perform some set of operations, and 2) a set of tasks to be completed. In order to be completed, a task requires some subset of agents to perform the necessary operations. Thus, we can define a task by the operations that agents must perform in order to complete it. The problem to be solved is an allocation of agents to tasks such that all tasks are performed. This problem is formalized next.

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

- $\mathcal{A}g$ is a set of agents, $\mathcal{A}g = \{A_1, A_2, ..., A_n\}$.
- $\Omega = \{O_1^1, O_2^1, ..., O_p^i, ..., O_q^n\}$ is a set of operations, where operation O_p^i denotes the p'th operation of agent A_i. An operation can either succeed or fail. 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.

- Θ is a set of tasks, where a task is a collection of sets of operations that satisfy the following properties: $\forall T \in \Theta$,
  (i) $T \subseteq P(\Omega)$ (Power set of Ω)
  (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 $t_s \not\subseteq t_r$. t_r and t_s are called *minimal sets*. Two minimal sets *conflict* if they contain operations belonging to the same agent.

Notice that there may be alternative sets of operations that can complete a given task. Each such set is a minimal set. (Property (iii) above requires that each set of operations in a task should be minimal in the sense that no other set is a subset of it.) A *solution* to a resource allocation problem then, involves choosing a minimal set for each task such that the minimal sets do not conflict. In this way, when the agents perform the operations in those minimal sets, all tasks are successfully completed.

To illustrate this formalism in the distributed sensor network domain, we cast each sensor as an agent and activating one of its (three) sectors as an operation. We will use O_p^i to denote the operation of agent A_i activating sector p. For example, in Figure 1.b, we have four agents, so $\mathcal{A}g = \{A_1, A_2, A_3, A_4\}$. Each agent can perform one of three operations, so $\Omega = \bigcup_{A_i \in \mathcal{A}g} Op(A_i)$, where $Op(A_i) = \{\, O_0^i, O_1^i, O_2^i \,\}$.

Now we only have left to define our task set Θ. We will define a separate task for each target in a particular location, where a location corresponds to an area of overlap of sectors. In the situation illustrated in Figure 1.b, we have two targets shown, so we define two tasks: $\Theta = \{T_1, T_2\}$. Since a target requires three agents to track it so that its position can be triangulated, Task T_1 requires any three of the four possible agents to activate their correct sector, so we define a minimal set corresponding to the all (4 choose 3) combinations. Thus, $T_1 = \{\{O_0^1, O_2^2, O_0^3\}, \{O_2^2, O_0^3, O_1^4\}, \{O_0^1, O_0^3, O_1^4\}, \{O_0^1, O_2^2, O_1^4\}\}$. Note that the subscript of the operation denotes the number of the sector the agent must activate. Task T_2 can only be tracked by two agents, both of which are needed, so $T_2 = \{\{O_0^3, O_2^4\}\}$.

For each task, we use $\Upsilon(T_r)$ to denote the union of all the minimal sets of T_r, and for each operation, we use $T(O_p^i)$ to denote the set of tasks that include O_p^i. For instance, $\Upsilon(T_1) = \{O_0^1, O_2^2, O_0^3, O_1^4\}$ and $T(O_0^3) = \{\, T_1, T_2 \,\}$. We will also require that every operation should serve some task, i.e. $\forall\, O_p^i \in \Omega$, $|\, T(O_p^i) \,| \neq 0$. Formal definitions for Υ and T are as follows:

- $\forall\, T_r \in \Theta, \Upsilon(T_r) = \bigcup_{t_r \in T_r} t_r$
- $\forall\, O_p^i \in \Omega, T(O_p^i) = \{T_r \mid O_p^i \in \Upsilon(T_r)\}$

All the tasks in Θ may not always be present. We use $\Theta_{current}$ ($\subseteq \Theta$) to denote the set of tasks that are currently present. This set is determined by the environment. We call a resource allocation problem *static* if $\Theta_{current}$ is constant over time and *dynamic* otherwise. So in our distributed sensor network example, a moving target represents a dynamic problem. 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. The following two definitions formalize this interface with the environment.

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

So in our example, if agent A_1 executes operation O_0^1 and if $T_1 \in \Theta_{current}$, then O_0^1 will succeed, otherwise it will fail. Next, 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. All tasks that satisfy this definition are contained in $\Theta_{current}$.

Agents must somehow be informed of the set of current tasks. The notification procedure is outside of this formalism. Thus, the following assumption states that at least one agent is notified that a task is present by the success of one of its operations. (This assumption can be satisfied in the distributed sensor domain by agents "scanning" for targets by rotating sectors when they are currently not tracking a target.)

Notification assumption: $\forall T_r \in \Theta$, if $T_r \in \Theta_{current}$, then $\exists\ O_p^i \in \Upsilon(T_r)$ such that $\forall T_s(\neq T_r) \in \Theta_{current}$, $O_p^i \notin T_s$ and O_p^i succeeds.

We now state some definitions that will allow us to categorize a given resource allocation problem and analyze its complexity. In many resource allocation problems, tasks have the property that they require at least k agents from a pool of n ($n > k$) available agents. That is, the task contains a minimal set for each of the $\binom{n}{k}$ combinations. The following definition formalizes this notion.

Definition 3: $\forall\ T_r \in \Theta$, T_r is **task-$\binom{n}{k}$-exact** iff T_r has exactly $\binom{n}{k}$ minimal sets of size k, where $n = |\ \Upsilon(T_r)\ |$.

For example, the task T_1 (corresponding to target 1 in Figure 1.b) is task-$\binom{4}{3}$-exact. The following just defines the class of resource allocation problems where all tasks satisfy the above definition.

Definition 4 : **$\binom{n}{k}$-exact** denotes the class of resource allocation problems $<\mathcal{A}g$, Ω, $\Theta>$ such that $\forall\ T_r \in \Theta$, T_r is task-$\binom{n}{k_r}$-exact for some k_r.

We find it useful to define a special case of $\binom{n}{k}$-exact resource allocation problems, namely those when $k = n$. In other words, the task contains only one minimal set.

Definition 5: **$\binom{n}{n}$-exact** denotes the class of resource allocation problems $<\mathcal{A}g$, Ω, $\Theta>$ such that $\forall\ T_r \in \Theta$, T_r is task-$\binom{n_r}{k_r}$-exact, where $n_r = k_r =|\ \Upsilon(T_r)\ |$.

For example,the task T_2 (corresponding to target 2 in Figure 1.b) is task-$\binom{2}{2}$-exact.

Definition 6: **Unrestricted** denotes the class of resource allocation problems $<\mathcal{A}g$, Ω, $\Theta>$ with no restrictions on tasks.

The following definitions refer to relations between tasks. We define two types of *conflict-free* to denote resource allocation problems that have solutions, or equivalently, problems where all tasks can be performed concurrently.

Definition 7: A resource allocation problem is called **strongly conflict free (SCF)** if $\forall\ T_r, T_s \in \Theta$, the following statement is true:

– if $T_r \neq T_s$, then $\forall\ t_r \in T_r, \forall\ t_s \in T_s, \forall\ A_i \in \mathcal{A}g$, $|\ t_r \cap Op(A_i)\ | + |\ t_s \cap Op(A_i)\ | \leq 1$.

The SCF condition states that an operation of an agent can be required for no more than one task. This implies that we can choose any minimal set for each task and be guaranteed a non-conflicting solution.

Definition 8: A resource allocation problem is called **weakly conflict free (WCF)** if $\forall\ T_r, T_s \in \Theta$, the following statement is true:

– if $T_r \neq T_s$, then $\exists\, t_r \in T_r$, $\exists\, t_s \in T_s$ $\forall\, A_i \in \mathcal{A}g$, $|\, t_r \cap Op(A_i)\,| + |\, t_s \cap Op(A_i)\,| \leq 1$.

The WCF condition is much weaker in that it implies that there exists a choice of minimal sets from the tasks that are non-conflicting or in other words, there exists at least one solution.

4 Dynamic DCSP

In order to solve general resource allocation problems that conform to our formalized model, we will use distributed constraint satisfaction techniques. Existing approaches to distributed constraint satisfaction fall short for our purposes however because they cannot capture the dynamic aspects of the problem. In dynamic problems, a solution to the resource allocation problem at one time may become obsolete when the underlying tasks have changed. This means that once a solution is obtained, the agents must continuously monitor it for changes and must have a way to express such changes in the problem. In order to address this shortcoming, the following section defines the notion of a Dynamic Distributed Constraint Satisfaction Problem (DyDCSP).

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 (DyDCSP). DyDCSP 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 DyDCSP 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 responds 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. 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 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 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 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 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. 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.

5 Solving SCF Problems via DyDCSP

In this section, we state the complexity of SCF resource allocation problems and map our formal model of the resource allocation problem onto DyDCSP. Our goal is to provide a general mapping so that any unrestricted SCF resource allocation problem can be solved in a distributed manner by a set of agents by applying this mapping.

Our complexity analysis (not the DyDCSP mapping, but just the complexity analysis) here assumes a static problem. This is because a dynamic resource allocation problem can be cast as solving a sequence of static problems, so a dynamic problem is at least as hard as a static one. Furthermore, our results are based on a centralized problem solver. We conjecture that distributed problem solving is no easier due to ambiguity, which requires more search.

Theorem I: Unrestricted Strongly Conflict Free resource allocation problems can be solved in time linear in the number of tasks.

proof: Greedily choose any minimal set for each task. They are guaranteed not to conflict by the Strongly Conflict Free condition. □

We now describe a solution to this subclass of resource allocation problems by mapping onto DyDCSP. Mapping I is motivated by the following idea. The goal in DCSP is for agents to choose values for their variables 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, $\{O_1^i, O_2^i, O_3^i\}$, then the variable corresponding to this agent will have three values in its domain. However, this simple mapping attempt fails due to the dynamic nature of the problem; operations of an agent may not always succeed. Therefore, 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 due to ambiguity. Ambiguity arises when 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 1.b, 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.

Mapping I: Given a Resource Allocation Problem $\langle \mathcal{A}g, \Omega, \Theta \rangle$, the corresponding DyDCSP 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$(A_i)$ = $\bigcup\limits_{O_p^i \in \Omega} O_p^ixT(O_p^i)x\{yes,no\}$.

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 1.b, 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: $\forall$ $O_p^i \in \Omega$, the set of values in O_p^ix$T(O_p^i)$x$\{$yes$\}$ will be abbreviated by the term O_p^i*yes and the assignment A_i = O_p^i*yes denotes that $\exists v \in O_p^i$*yes such that $A_i = v$. 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 analogous 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 dynamic 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)**: $\forall T_r \in \Theta$, $\forall O^i_p \in \Upsilon T_r)$, we have
  LC1(A_i) = (P, C), where P: O^i_p succeeds.
  C: $A_i = O^i_p$*yes
- **Dynamic Local Constraint 2 (LC2)**: $\forall T_r \in \Theta$, $\forall O^i_p \in \Upsilon(T_r)$, we have
  LC2(A_i) = (P, C), where P: O^i_p does not succeed.
  C: $A_i \neq O^i_p$*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 dynamic problems, where the set of current tasks is changing over time, the truth value of P will change, and hence the corresponding DyDCSP will also be 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**: $\forall T_r \in \Theta$, $\forall O^i_p \in \Upsilon(T_r)$, $\forall A_j \in A$,
  EC(A_i, A_j): (1) A_i = $O^i_p T_r$yes, and
  (2) $\forall t_r \in T_r$, $O^i_p \in t_r$, $\exists q$ $O^j_q \in t_r$.
  $\Rightarrow A_j$ = $O^j_q T_r$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.

The following theorems state that our mapping can be used to solve any given SCF Resource Allocation Problem. The first theorem states that our DyDCSP always has a solution, and the second theorem states 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 II: Given an unrestricted SCF Resource Allocation Problem $\langle \mathcal{A}g,\Omega,\Theta\rangle$, $\Theta_{current} \subseteq \Theta$, a solution always exists for the DyDCSP obtained from Mapping I.

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

Let $B = \{A_i \in A \mid \exists T_r \in \Theta_{current}, \exists O_p^i \in \Upsilon(T_r)\}$. We will first assign values to variables in B, then assign values to variables that are not in B. If $A_i \in B$, then $\exists T_r \in \Theta_{current}, \exists O_p^i \in \Upsilon(T_r)$. 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 $t_r \in T_r$, $O_p^i \in t_r$. 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 $t_r \in T_r$, $O_p^i \in t_r$. T_s and T_r must be strongly conflict free since both are in $\Theta_{current}$. If $T_s \neq T_r$, then $\nexists$ $O_n^j \in \Omega$, $O_n^j \in t_r$. So condition (2) of EC(A_i,A_j) is false and thus EC is trivially satisfied. If $T_s = T_r$, then EC is satisfied since A_j is helping A_i perform T_r.

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 DyDCSP. □

Theorem III: Given an unrestricted SCF Resource Allocation Problem $\langle \mathcal{A}g$,Ω,$\Theta \rangle$, $\Theta_{current} \subseteq \Theta$ and the DyDCSP obtained from Mapping I, if an assignment of values to variables in the DyDCSP is a solution, then all tasks in $\Theta_{current}$ are performed.

proof: Let a solution to the DyDCSP be given. We want to show that all tasks in $\Theta_{current}$ are performed. We proceed by choosing a task $T_r \in \Theta_{current}$. Since our choice is arbitrary and tasks are strongly conflict free, if we can show that it is indeed performed, 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 $A_j = O_q^j T_s yes$. 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 definition, $T_s \in \Theta_{current}$. 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. □

6 Solving WCF problems via DyDCSP

In this section, we state the complexity of $\binom{n}{k}$-exact WCF resource allocation problems and that of unrestricted WCF resource allocation problems. The following complexity results are based on a centralized problem solver, but as mentioned we conjecture that distributed problem solving is no easier. We also present a second mapping for WCF problems onto DyDCSP (Section 6.1).

Theorem IV: $\binom{n}{n}$-exact WCF resource allocation problems can be solved in time linear in the number of tasks.

proof: Greedily choose the single minimal set for each task.

Theorem V: $\binom{n}{k}$-exact WCF resource allocation problems can be solved in time polynomial in the number of tasks and operations.

proof: To prove this theorem, we convert a given $\binom{n}{k}$-exact resource allocation problem to a network-flow problem which is known to be polynomial. See Appendix.

Theorem VI: Determining whether an unrestricted resource allocation problem is Weakly Conflict Free is NP-Complete.

proof-sketch: We reduce from 3 coloring problem. For reduction, let an arbitrary instance of 3-color with colors c_1, c_2, c_3, vertices V and edges E, be given. We construct the resource allocation problem as follows:

- For each vertex $v \in V$, add a task T_v to Θ.
- For each task $T_v \in \Theta$, for each color c_k, add a minimal set $t_v^{c_k}$ to T_v.
- For each edge $v_i, v_j \in E$, for each color c_k, add an operator $O_{v_i,v_j}^{c_k}$ to Ω and add this operator to minimal sets $t_{v_i}^{c_k}$ and $t_{v_j}^{c_k}$.
- Assign each operator to a unique agent $A_{O_{v_i,v_j}^{c_k}}$ in $\mathcal{A}g$.

Figure 2 illustrates the mapping from a 3 node graph to a resource allocation problem. With the mapping above, it is somewhat easy to show that the 3-color problem has a solution if and only if the constructed resource allocation problem is weakly conflict free. (We preclude a detailed proof due to space limits)

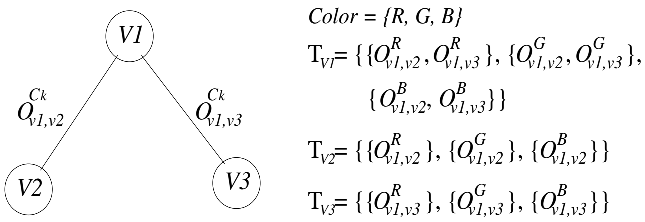


Fig. 2. Reduction of graph 3-coloring to Resource Allocation Problems

6.1 Mapping II

Our first mapping has allowed us to solve any SCF resource allocation problem. However, when we attempt to solve WCF resource allocation problems with this mapping, it fails because the DyDCSP becomes overconstrained. This is due to the fact that the mapping requires all agents who can possibly help perform a task to do so. In some sense, this results in an overallocation of resources to some tasks. This in turn leaves other tasks without sufficient resources to be performed. One way to solve this problem is to modify the constraints in the mapping to allow agents to reason about relationships among tasks. However, this requires adding non-binary external constraints to the mapping. This is problematic in a distributed situation because there are no efficient algorithms for non-binary distributed CSPs. Instead, create a new mapping that has only binary external constraints. This mapping is similar to the dual of a version of mapping I with non-binary external constraints. This new mapping allocates only minimal resources to each task, allowing WCF problems to be solved. This mapping is described next and proven correct. Here, each agent has a variable for each task in which its operations are included.

Mapping II: Given a Resource Allocation Problem $\langle \mathcal{A}g, \Omega, \Theta \rangle$, the corresponding DyDCSP is defined as follows:

- **Variables:** $\forall T_r \in \Theta, \forall O_p^i \in \Upsilon(T_r)$, create a DyDCSP variable $T_{r,i}$ and assign it to agent A_i.
- **Domain:** For each variable $T_{r,i}$, create a value $t_{r,i}$ for each minimal set in T_r, plus a "NP" value (not present). The NP value allows agents to avoid assigning resources to tasks that are not present and thus do not need to be performed.

Next, we must constrain agents to assign non-NP values to variables only when an operation has succeeded, which indicates the presence of the corresponding task. However, in dynamic 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 (Non-Binary) Constraint (LC1)**:
  $\forall A_i \in \mathcal{A}g, \forall O_p^i \in Op(A_i)$, let B = $\{ T_{r,i} \mid O_p^i \in T_r \}$. Then let the constraint be defined as a non-binary constraint over the variables in B as follows:
  P: O_p^i succeeds
  C: $\exists T_{r,i} \in$ B $T_{r,i} \neq$ NP.
- **Dynamic Local Constraint (LC2)**: $\forall T_r \in \Theta, \forall O_p^i \in \Upsilon(T_r)$, let the constraint be defined on $T_{r,i}$ as follows:
  P: O_p^i does not succeed
  C: $T_{r,i}$ = NP

We now define the constraint that defines a valid allocation of resources and the external constraints that require agents to agree on a particular allocation.

- **Static Local Constraint (LC3)**: $\forall T_{r,i}, T_{s,i}$, if $T_{r,i} = t_{r,i}$, then the value of $T_{s,i}$ cannot conflict with the minimal set $t_{r,i}$. NP does not conflict with anything.

– **External Constraint (EC)**: $\forall i, j, r\ T_{r,i} = T_{r,j}$

We will now prove that Mapping II can also be used to solve any given WCF Resource Allocation Problem. The first theorem shows that our DyDCSP always has a solution, and the second theorem shows that if agents reach a solution, all current tasks are performed.

Theorem VII: Given a WCF Resource Allocation Problem $\langle \mathcal{A}g, \Omega, \Theta \rangle$, $\Theta_{current} \subseteq \Theta$, there exists a solution to DyDCSP obtained from Mapping II.

proof: For all variables corresponding to tasks that are not present, we can assign the value "NP". This value satisfies all constraints except possibly LC1. But the P condition must be false since the task is not present, so LC1 cannot be violated. We are guaranteed that there is a choice of non-conflicting minimal sets for the remaining tasks (by the WCF condition). We can assign the values corresponding to these minimal sets to those tasks and be assured that LC3 is satisfied. Since all variable corresponding to a particular task get assigned the same value, the external constraint is satisfied. So we have a solution to the DyDCSP.□

Theorem VIII: Given a WCF Resource Allocation Problem $\langle \mathcal{A}g, \Omega, \Theta \rangle$, $\Theta_{current} \subseteq \Theta$ and the DyDCSP obtained from Mapping II, if an assignment of values to variables in the DyDCSP is a solution, then all tasks in $\Theta_{current}$ are performed.

proof Let a solution to the DyDCSP be given. We want to show that all tasks in $\Theta_{current}$ are performed. We proceed by randomly choosing a task from $\Theta_{current}$ and showing that it is performed. Since we are in a solution state, LC3 allows us to repeat this argument for every task in $\Theta_{current}$.

Let $T_r \in \Theta_{current}$. By the **Notification Assumption**, some operation O^i_p, required by T_r will be executed and (by definition) succeed. LC1 requires the corresponding agent A_i, to assign a minimal set, say t_r to the variable $T_{r,i}$. The EC constraint will then require that all other agents A_j, whose operation O^j_q is in the minimal set t_r, to assign $T_{r,j} = t_r$ and execute that operation. LC2 requires that it succeeds. Since all operations required for T_r succeed, T_r is performed. □

7 Experiments in a Real-World Domain

We have successfully applied the DyDCSP approach to the distributed sensor network problem, using the mapping introduced in Section 6. In the last evaluation trials conducted in government labs in August and September 2000, this DyDCSP 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 DyDCSP 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 DyDCSP 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. Domain experts termed the RMS error of upto 3 units as acceptable.

Table 1 presents our results from the implementation with the Mapping II in Section 6. Experiments were conducted in different dynamic situations varying the type of resource allocation problem, the number of nodes/targets, and the configuration. RMS error, message number, and sector change are averaged over the number of involved agents. In the "node number" column, the number in parentheses indicates the number of rows and columns of the grid configuration where sensor agents are located. For instance, the last row represents the result of the WCF resource allocation problem with 12 sensor nodes (in 3x4 grid) and 4 four targets: the RMS of 3.24 units with average 30 messages and 2 sector changes per node.

The results show that our mapping works, and agents are able to accurately track targets, with average RMS around 3 units as the experts require. This proves the usefulness of the DyDCSP approach to this resource allocation problem. Furthermore, scaling up the number of nodes and targets does not degrade the tracking accuracy. Some interesting differences between WCF and SCF arise: WCF resource allocation problems require more number of messages and sector changes than SCF problems. These are due to the fact that, given WCF problems, agents need to reason about possible minimal sets of the current tasks to be performed.

RAP type	node number	target number	avg RMS	avg message number	avg sector changes
WCF/SCF	4 (2x2)	1	2.58	14	0.5
SCF	8 (2x4)	2	3.21	17.08	0.5
SCF	9 (3x3)	2	3.21	21.89	0.2
SCF	16 (4x4)	4	2.58	23.13	0.5
WCF	6 (2x3)	2	2.50	45.17	1.6
WCF	12 (3x4)	4	3.24	30	2.0

Table 1. Results from sensor network domain for dynamic resource allocation problems.

8 Summary

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 different categories of difficulties of the problem and present complexity results for them. Table 2 summarizes these complexity results. To address these formalized problems, we define the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDCSP) and present a generalized mapping from distributed resource allocation to DyDCSP. 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. Indeed, in the future, our formalization may enable researchers to understand the difficulty of their resource allocation problem, choose a suitable mapping using DyDCSP, 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.

	SCF	WCF
$\binom{n}{n}$-exact	O(n)	O(n)
$\binom{n}{k}$-exact	O(n)	O($(n+m)^3$)
unrestricted	O(n)	NP-Complete

Table 2. Complexity Classes of Resource Allocation, n = size of task set Θ, m = size of operation set Ω

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.

Appendix

Theorem V: $\binom{n}{k}$-exact WCF resource allocation problems can be solved in time polynomial in the number of tasks and operations.

proof: We can convert a given $\binom{n}{k}$-exact resource allocation problem to a network-flow problem known to be polynomial. Let such a resource allocation problem be given. We first construct a tripartite graph and then convert it to a network-flow problem.

- Create three empty sets of vertices, U, V, and W and an empty edge set E.
- For each task $T_r \in \Theta$, add a vertex u_r to U.
- For each agent $A_i \in \mathcal{A}g$, add a vertex v_i to V.
- For each agent A_i, for each operation $O_p^i \in Op(A_i)$, add a vertex w_p^i to W.
- For each agent A_i, for each operation $O_p^i \in Op(A_i)$, add an edge between vertices v_i, w_p^i to E.
- For each task T_r, for each operation $O_p^i \in \Upsilon(T_r)$, add an edge between vertices u_r, w_p^i to E.

We convert this tripartite graph into a network-flow graph in the usual way. Add two new vertices, a supersource s, and supersink t. Connect s to all vertices in V and assign a max-flow of 1. For all edges among V, W, and U, assign a max-flow of 1. Now, connect t to all vertices in U and for each edge (u_r, t), assign a max-flow of k_r. We now have a network flow graph with an upper limit on flow of $\sum_{i=1}^{|\theta|} k_i$.

We show that the resource allocation problem has a solution if and only if the max-flow is equal to $\sum_{i=1}^{|\theta|} k_i$.

$\Rightarrow$ Let a solution to the resource allocation problem be given. We will now construct a flow equal to $\sum_{i=1}^{|\theta|} k_i$. This means, for each edge between vertex u_r in U and t, we must assign a flow of k_r. It is required that the in-flow to u_r equal k_r. Since each edge between W and U has capacity 1, we must choose k_r vertices from W that have an edge into u_r and fill them to capacity. Let T_r be the task corresponding to vertex u_r, and $t_r \in T_r$ be the minimal set chosen in the given solution. We will assign a flow of 1 to all edges (w_p^i, u_r) such that w_p^i corresponds to the operation O_p^i that is required in t_r. There are exactly k_r of these. Furthermore, since no operation is required for two different tasks, when we assign flows through vertices in U, we will never choose w_p^i again. For vertex w_p^i such that the edge (w_p^i, u_r) is filled to its capacity, assign a flow of 1 to the edge (v_i, w_p^i). Here, when a flow is assigned through a vertex w_p^i, no other flow is assigned through $w_q^i \in Op(A_i)$ ($p \neq q$) because all operations in $Op(A_i)$ are mutually exclusive. Therefore, v_i's outflow cannot be greater than 1. Finally, the assignment of flows from s to V is straightforward. Thus, we will always have a valid flow (inflow=outflow). Since all edges from U to t are filled to capacity, the max-flow is equal to $\sum_{i=1}^{|\theta|} k_i$.

$\Leftarrow$ Assume we have a max-flow equal to $\sum_{i=1}^{|\theta|} k_i$. Then for each vertex u_r in U, there are k_r incoming edges filled to capacity 1. By construction, the set of vertices in W matched to u_r corresponds to a minimal set in T_r. We choose this minimal set for the solution to the resource allocation problem. For each such edge (w_p^i, u_r), w_p^i has an in-capacity of 1, so every other edge out of w_p^i must be empty. That is, no operation is required by multiple tasks. Furthermore, since outgoing flow thorough v_i is 1, no more than one operation in $Op(A_i)$ is required. Therefore, we will not have any conflicts between minimal sets in our solution. □