

Distributed Resource Allocation: Formalization, Complexity Results and Mappings to Distributed CSPs *

Pragnesh Jay Modi Wei-Min Shen Milind Tambe

University of Southern California
Information Sciences Institute
Department of Computer Science
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{modi,tambe,shen}@isi.edu

November 27, 2002

Abstract

In distributed resource allocation a set of agents must assign their resources to a set of dynamic tasks. This problem arises in many real-world domains like the one 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. Such formalizations are necessary to understand the complexity of different types of problems and to develop solution strategies that translate across domains. This paper takes a step towards this goal by proposing a formalization of resource allocation that represents both dynamic and distributed aspects of the problem and allows tractable subclasses to be identified. In addition, this paper defines the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDisCSP) and proposes two generalized mappings to translate distributed resource allocation problems to DyDisCSP. Each mapping is proven to correctly perform resource allocation problems of specific difficulty. Our theoretical results are verified in practice by an implementation on a real-world distributed sensor network.

*This paper is an extension of a earlier conference paper [12]. Detailed exposition, experimental results and detailed proofs are included here.

1 Introduction

Distributed resource allocation is a general problem in which a set of agents must perform operations and intelligently assign their resources to a set of distributed tasks. This problem arises in many real-world domains such as distributed sensor networks [21], disaster rescue[8], hospital scheduling[4], and others.

Resource allocation problems of this type are difficult because they are both *distributed* and *dynamic*. First, 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. Second, another implication is that agents face *global ambiguity* — an agent may know based on the results of its local operations, that some task (out of a possible set of tasks) is present. However, it may not be able to locally determine exactly which task is present. The agents must collaborate to determine which one of the many possible tasks is actually present and needs to be done. Third, different tasks may require the same resources and thus, *resource contention* may occur. In these situations, agents must take care to allocate critical resources to appropriate tasks – allocating a critical resource incorrectly may lead to situations where some other tasks must go unperformed. 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. The agents must have a way to express and cope with such changes in the problem.

Despite the significant progress in distributed resource allocation, there are currently two key shortcomings in this research. First, abstract formalization of the problem and domain types, which allow us to understand the complexity of different problem categories, are missing. Such abstract formalizations are important because if constructed appropriately, they allow tractable subclasses to be identified, allow theoretical results to transfer across domains and allow solution strategies and modeling effort to be reused. Second, general mapping strategies that allow distributed resource allocation problem types to be mapped to key problem solving paradigms — in our case, we will focus on distributed constraint satisfaction – are missing.

In this paper, we attempt to develop a systematic formalization of the problem and a general solution strategy in order to obtain the above benefits. First, we propose a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. This formalization is significant because it allows us to understand the complexity of different types of resource allocation problems and may also enable future researchers to understand the difficulty of their own resource allocation problem. We present detailed complexity results for various subclasses of resource allocation in our formal model and identify tractable subclasses. Second, in order to solve these types of resource allocation problems, we define the notion of a Dynamic Distributed Constraint Satisfaction Problem (DyDisCSP). Distributed constraint reasoning in general has been shown to be an effective way to model and solve complex distributed problems in a distributed manner [12] [7] [2] [20] [14] and is a very active area of current research [27] [30][11][18]. Yokoo, Durfee, Ishida, and Kuwabara have done seminal work in distributed constraint satisfaction (DisCSP) [28]. While this paper does not focus on new algorithms for DisCSP, it does focus on applying DisCSP in service of distributed resource allocation problems. In dynamic domains where features of the environment may not be known in advance, it is difficult to completely specify a DisCSP problem in advance. To address this difficulty, DyDisCSP generalizes DisCSP by allowing agents to add or remove local constraints from the problem as external

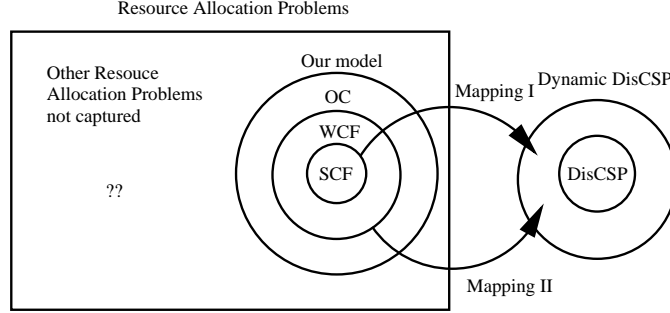


Figure 1: Graphical depiction of the methodology described in this paper.

environmental conditions change.

Given the distributed resource allocation formalism and the DyDisCSP problem, we present two reusable, generalized mappings, Mapping I and II, that automatically convert a given distributed resource allocation problem into a DyDisCSP. Each mapping is proven to correctly perform resource allocation problems of specific difficulty. These generalized mappings enable existing distributed constraint reasoning technologies to be brought to bear directly onto the distributed resource allocation problem. In addition, they allow future advances in distributed constraint reasoning to also be directly applied to the distributed resource allocation problem without significant re-modeling effort. Thus, our formalism and generalized mappings provide researchers with an automatic method for representing their resource allocation problem using constraints, while existing and future distributed constraint reasoning techniques provide automatic solution algorithms.

Figure 1 depicts our methodology. The presentation of these ideas is as follows. Section 2 describes a concrete distributed resource allocation domain, distributed sensor networks, which is used throughout the paper to illustrate our mathematics. Section 3 describes related work in resource allocation and sensor networks. Our abstract model of distributed resource allocation, also intended to apply to domains beyond sensor networks, is presented in Section 4. Section 4 also presents a series of definitions (Strongly Conflict Free (SCF), Weakly Conflict Free (WCF), Overconstrained (OC), etc.) that are used to classify resource allocation domain types of varying degrees of difficulty. Section 5 leaves the resource allocation problem for a moment to define the Dynamic Distributed Constraint Satisfaction Problem (DyDisCSP). Armed with the definition of DyDisCSP, Section 6 goes on to define the first mapping, Mapping I, of SCF problems into DyDisCSP. Proofs of correctness of this mapping are also presented in this section. Analogous to Section 6, Section 7 presents a new mapping, Mapping II, for WCF problems and similar proofs of correctness are presented. Section 8 presents a complexity result for OC problems. Finally, Section 9 experimentally verifies Mapping II within the distributed sensor network domain described in Section 2. Section 10 concludes by summarizing and describing future work.

2 Application Domain

We describe a real-world dynamic distributed resource allocation problem which we will use to both illustrate and validate our contributions. This domain consists of multiple stationary sensors, each controlled by an independent agent, and targets moving through their sensing range (Figure 2.a illustrates the real hardware and simulator screen, respectively). Each sensor is equipped with

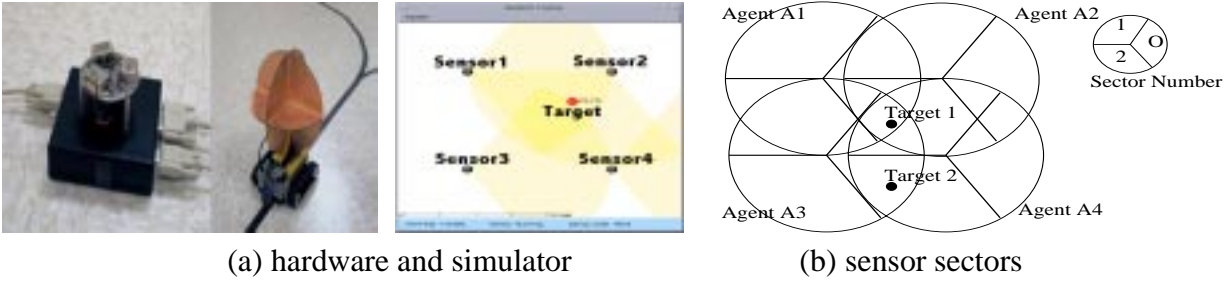


Figure 2: A distributed sensor network for tracking moving targets.

a Doppler radar with three sector heads. An agent may activate at most one sector of the sensor it controls at a given time, or may switch the entire 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.

The first difficulty is that the domain is inherently distributed. In order for a target to be tracked accurately, at least three agents must collaborate — they must concurrently activate their sensors so the target is painted by at least three overlapping sector heads. For example, in Figure 2.b which corresponds to the simulator in Figure 2.a, if an agent A1 detects target 1 in its sector 0, it must inform two of its neighboring agents, A2 and A4 for example, so that they activate their respective sectors that overlap with A1’s sector 0. Activating a sector is an agent’s operation. Since each sensor has three sectors of 120 degrees and only one sector can be active at a time, each agent has three mutually exclusive operations it can perform.

The second difficulty is that when an agent is informed about a target, it may face ambiguity in which sector to activate. Each sensor can detect only the distance and speed of a target, so an agent that detects a target can only inform another agent about the general area of where a target may be, but cannot tell other agents specifically which sector they must activate. For example, suppose there is only target 1 in Figure 2.b and agent A1 detects that a target is present in its sector 0. A1 can tell A2 that a target is somewhere in the region of its sector 0, but it cannot tell A2 which sector to activate because A2 has two sectors (sector 1 and 2) that overlap with A1’s sector 0. In order to resolve this ambiguity, A2 may be forced to first activate its sector 1, detect no target, then try its sector 2. Thus, activating the correct sector requires a collaboration between A1 and A2 – A1 informs A2 that a target exists in some ambiguous region and A2 then resolves the remaining ambiguity. The significance here is that no single agent can determine the correct allocation of all sectors to targets.

The third difficulty is that resource contention may occur when multiple targets must be tracked simultaneously. For instance, in Figure 2.b, A4 needs to decide whether to track target 1 or target 2 – it cannot do both since it may activate only one sector at a time. If A4 chooses target 1, target 2 may go untracked because A4 may be critical for tracking target 2. Determining whether an agent is critical requires non-local information, so agents must again collaborate. In this example, target 1 and 2 are tasks that conflict with one another. Targets that are spatially distant do not conflict with each other and thus can easily be tracked without resource contention. Thus, as we will see, the relationship among tasks will affect the difficulty of the overall resource allocation problem.

Finally, the situation is dynamic as targets move through the sensing range. Even after agents find a configuration that is accurately tracking all targets, they will have to reconfigure, or reallocate themselves, as targets move over time.

While we will focus mainly on the distributed sensor network problem, a second domain which motivates our work is large-scale urban disaster recovery, e.g., agents performing search and rescue operations after an earthquake. Specifically, Robocup Rescue [8] is a detailed simulation environment of the 1995 Kobe, Japan earthquake in which over 5000 people were killed. In this simulator, multiple fire engines, ambulances and police cars must collaborate to save trapped civilians from burning buildings. No centralized control is available to allocate all of the emergency response resources since communication infrastructure may be damaged or overloaded. Individual agents must communicate locally and collaborate with one another in order to allocate their resources correctly. For instance, an ambulance agent which must rescue a civilian trapped in a burning building must collaborate with a fire engine agent who is able to extinguish the fire. 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 to the distributed sensor network problem.

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

3 Related Work

A variety of researchers have focused on formalizing resource allocation as a centralized CSP[5]. In addition, the Dynamic Constraint Satisfaction Problem has been studied in the centralized case by [17]. In centralized CSP, there is no distribution or ambiguity during the problem solving process. However, the fact that the resource allocation problem is inherently distributed in many domains means that ambiguity must be dealt with. We also categorize different resource allocation problems and provide detailed complexity results.

Distributed sensor networks have gained significant attention in recent years. Sensor hardware nodes are becoming more sophisticated and able to support increasing levels of both computation and communication. This trend, in turn, has led to the consideration of the Distributed AI (DAI) and the Multiagent perspective for addressing technical problems in distributed sensor networks [16] [6] [19] [22]. Soh and Tsatsoulis [19] describe a case-based reasoning approach where agents choose different negotiation strategies depending on environmental circumstances in order to collaboratively track moving targets. Vincent et al. [22] and Horling et al. [6] bring existing multi-agent technologies, such as the TAEMS modelling language [3] and the Design-to-Criteria plan scheduler [23], to bear on the distributed sensor network problem. These approaches report positive results in customizing and applying existing DAI technologies to the specific problem of coordination in distributed sensor networks. However, this existing research does not draw any theoretical conclusions about distributed resource allocation in general.

There is significant research in the area of distributed resource allocation. For instance, Liu and Sycara [10] address resource allocation in the distributed job-shop scheduling problem. The solution technique presented is an extension of local dispatch scheduling – the extension allows agents to use non-local information to make their local decisions. Schedules are shown to improve using this technique. Chia et al’s [1] work on job-shop scheduling for airport ground service schedules is another example of distributed resource allocation. The authors are able to reduce schedule inefficiencies by allowing agents to communicate heuristic domain specific information about their local jobs. Finally, the Distributed Vehicle Monitoring Testbed (DVMT) of Lesser et al. [9] is well known in distributed AI as a domain for distributed problem solving. A set of problem-solving nodes must cooperate and integrate distributed sensing data to accurately track a moving vehicle. This domain is inherently distributed and exhibits both dynamics and ambiguity. To summarize, while previous work in distributed resource allocation has been effective for particular problem domains, a formalization of the general problem which allows tractable subclasses to be identified, is yet to be developed.

A recent approach to distributed resource allocation that has received significant attention is that of market-based systems, or multi-agent auctions[26]. These price-based search techniques coordinate a set of agents by allowing them to buy and sell goods in order to maximize local utility. The price of a good implicitly conveys to an agent non-local information about the global utility of obtaining that good. This approach has been used effectively to structure distributed search in many multi-agent domains including multi-commodity flows [26], multi-agent task allocation [24] and even distributed propositional satisfiability [25]. While market-based approaches show great promise and applicability in open systems, they may be unnecessarily restrictive in collaborative domains explicitly engineered by a system designer. In collaborative domains, agents may be able to reach global optimal solutions faster by exchanging more information beyond prices. Sandholm and Suri [15] also discuss the need for non-price attributes and explicit constraints in conjunction with market protocols.

4 Formalization of Resource Allocation

While Section 2 provided a concrete example domain, in this section, we abstract to a general formalization of resource allocation problems. We divide this section into two subsections. The first subsection presents our formal model, along with two definitions and a Notification Assumption that formalizes the model’s interface with the environment. The second subsection goes on to introduce a group of definitions that allow a particular domain to be categorized into a subclass. Later, we will show that these subclasses correspond to varying degrees of complexity. Finally, note that we will use the distributed sensor domain as a concrete example for illustration throughout this section, however, the formalization provided is intended to be a general one, applicable to other domains (such as disaster rescue described in Section 2).

4.1 Formal Definitions

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 their 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.

• **Definition 1:** A Distributed Resource Allocation Problem is a structure $\langle \mathcal{A}g, \Omega, \Theta \rangle$ where

- $\mathcal{A}g = \{A_1, A_2, \dots, A_n\}$ is a set of agents.
- $\Omega = \{O_1^1, O_2^1, \dots, O_p^i, \dots, O_q^n\}$ is a set of operations, where operation O_p^i denotes the pth 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.
- $\Theta = \{T_1, T_2, \dots, T_n\}$ is a set of tasks, where each task $T \in \Theta$ is a set of non-empty sets $\{t_1, t_2, \dots, t_n\}$ and each $t_i \in T$ is a set of operations. Each t_i is called a *minimal set* because they must satisfy the following property: $\forall t_r, t_s \in T, t_r \not\subseteq t_s$ and $t_s \not\subseteq t_r$. This requires that each set of operations in a task should be minimal in the sense that no other set is a subset of it. Two minimal sets *conflict* if they contain an operation belonging to the same agent.

Intuitively, the minimal sets of a task specify the alternative ways (sets of operations) to complete the task. A *solution* to a resource allocation problem then, involves choosing a minimal set for each present task such that the minimal sets do not conflict. In this way, when the agents perform the operations in those minimal sets, all present tasks are successfully completed. One key difficulty lies in assigning the correct resources to each task so that all tasks get their required resources. An incorrect allocation of resources to one task may result in a shortage of resources for some other task.

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 2.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 $\Omega = \{O_0^1, O_1^1, O_2^1, O_0^2, O_1^2, O_2^2, O_0^3, O_1^3, O_2^3, O_0^4, O_1^4, O_2^4\}$. To specify the subset of operations belonging to a particular agent, say A_1 , we can use $Op(A_1) = \{O_0^1, O_1^1, O_2^1\}$.

We now define our task set Θ . We will define a separate task for each region of overlap of sectors. In this way, the existence of a target in a particular location corresponds to a task that needs to be performed. Regions of overlap that do not currently contain a target are tasks that do not currently need to be performed. In the situation illustrated in Figure 2.b, we have only two targets shown and for simplicity, let us assume these are the only possible target locations. So, we define our task set $\Theta = \{T_1, T_2\}$. Remember that each target requires three agents to track it so that its position can be triangulated. Thus, task T_1 requires any three of the four possible agents to activate their correct sector, so we define a minimal set corresponding to all the (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. In the example, task T_2 can only be tracked by two agents. For simplicity, let us assume this is sufficient and $T_2 = \{\{O_0^3, O_2^4\}\}$.

For each task, we use $\Upsilon(T_r)$ to denote the union over all the minimal sets of T_r , and for each operation, we use $T(O_p^i)$ to denote the set of tasks T_r that may require O_p^i , i.e., include O_p^i in $\Upsilon(T_r)$. For instance, $\Upsilon(T_1) = \{O_0^1, O_2^2, O_0^3, O_1^4\}$ and $T(O_0^3) = \{T_1, T_2\}$.

As mentioned above, 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 and require resources. 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. We do not model resource allocation problems where the resources may be dynamic (i.e., where agents may come and go). Agents must determine which tasks are currently present by executing their operations. The success of an operation is determined by the set of tasks that are currently present. The following definition formalizes this interface with the environment.

- **Definition 2:** $\forall O_p^i \in \Omega$, 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*. If an operation is executed, but it has no corresponding task in $\Theta_{current}$, the operation is said to *fail*.

So in our example, if agent A_1 executes operation O_0^1 (activates sector 1) and if $T_1 \in \Theta_{current}$ (target 1 is present), then O_0^1 will succeed (A_1 will detect a target), otherwise it will fail. In other domains such as disaster rescue, ambulance agents may use sensors that detect the presence of life in a building in order to detect civilians that need medical attention (tasks). Note that our notion of operation failure corresponds to a sensor signal indicating the absence of a task, not actual hardware failure. Hardware failure or sensor noise is an issue not modeled in our formalism here. However, an actual system built using this formalism has been able to incorporate techniques for dealing with noise and failure by using a two-layered architecture, whereby a lower layer of the implementation deals with these issues[16].

A dynamic problem requires agents to detect new tasks as they appear in the environment. This can be done in the distributed sensor domain by agents “scanning” for targets by rotating sectors when they are currently not tracking any target. In disaster rescue, ambulances can drive around searching for injured civilians. This notification procedure is outside of our formalism and we rely on the following assumption.

- **Notification assumption:**

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

(i) states that if a task is present, at least one agent is notified of the task by the success of one of its operations, i.e., no present task goes unnoticed by everyone. (ii) states that the notifying operation O_p^i (from (i)) must not be required for any other present task. This implies that the success of operation O_p^i will uniquely identify the task T_r among all present tasks. This assumption prevents the possibility whereby the success of a single operation serves to notify an agent of two present tasks. For example, in distributed sensor networks, hardware restrictions preclude this possibility – two targets simultaneously present in a single sector results in a garbled signal received by the sensor. Note that the ambiguity problem (figuring out which tasks are present and not present) is not eliminated because we do *not* require the task to be uniquely identified among all

(e.g., not present) tasks. Similarly, other difficulties such as resource contention, are not eliminated by this assumption.

Finally, a task is *performed* (the target is tracked) when all the operations in some minimal set succeed (enough radar sectors are painting the target). More formally,

- **Definition 3:** $\forall T_r \in \Theta$, T_r is *performed* iff there exists a minimal set $t_r \in T_r$ such that all the operations in t_r succeed. A task that is not present cannot be performed, or equivalently, a task that is performed must be included in $\Theta_{current}$.

For example, task T_2 is performed if and only if A_3 executes operation O_0^3 and A_4 executes operation O_2^4 and both operations succeed.

4.2 Properties of Resource Allocation

We now state some definitions that will allow us to categorize a given resource allocation problem and analyze its difficulty. In particular, we notice some properties of task and inter-task relationships. Definitions 4 through 7 are used to describe the complexity of a given task in a given problem, i.e., the definitions relate to properties of an individual task. Next, definitions 8 through 10 are used to describe the complexity of inter-task relationships, i.e., the definitions relate to the interactions between a set of tasks.

4.2.1 Task Complexity

Many resource allocation problems have the property that each task requires any k agents from a pool of n ($n \geq k$) 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 4:** $\forall T_r \in \Theta$, T_r is **task-** $\binom{n}{k}$ **-exact** iff T_r has exactly $\binom{n}{k_r}$ minimal sets of size k_r , where $n = |\Upsilon(T_r)|$ and $k_r (\leq n)$ depends on T_r .

For example, the task T_1 (corresponding to target 1 in Figure 2.b) is task- $\binom{4}{3}$ -exact because it has exactly $\binom{4}{3}$ minimal sets of size $k = 3$, where $n = 4 = |\Upsilon(T_1)|$. The following definition just defines the class of resource allocation problems where every task is task- $\binom{n}{k}$ -exact.

- **Definition 5:** $\binom{n}{k}$ -**exact** denotes the class of resource allocation problems $\langle \mathcal{A}g, \Omega, \Theta \rangle$ such that $\forall T_r \in \Theta$, T_r is task- $\binom{n}{k_r}$ -exact.

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, each task contains only a single minimal set.

- **Definition 6:** $\binom{n}{n}$ -**exact** denotes the class of resource allocation problems $\langle \mathcal{A}g, \Omega, \Theta \rangle$ 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 2.b) is task- $\binom{2}{2}$ -exact.

- **Definition 7: Unrestricted** denotes the class of resource allocation problems $\langle \mathcal{A}g, \Omega, \Theta \rangle$ with no restrictions on tasks.

Note that $\binom{n}{n}$ -exact $\subset \binom{n}{k}$ -exact \subset Unrestricted.

4.2.2 Task Relationship Complexity

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 8:** A resource allocation problem is called **strongly conflict free (SCF)** if for all T_r and $T_s \in \Theta_{current}$ and all agents $A_i \in \mathcal{A}g$, $|Op(A_i) \cap \Upsilon(T_r)| + |Op(A_i) \cap \Upsilon(T_s)| \leq 1$, i.e., no two tasks have in common an operation from the same agent.

The SCF condition implies that we can choose any minimal set out of the given alternatives for a task and be guaranteed that it will lead to a solution where all tasks are performed, i.e., no backtracking is ever required to find a solution.

- **Definition 9:** A resource allocation problem is called **weakly conflict free (WCF)** if there exists some choice of minimal set for every present task such that all the chosen minimal sets are non-conflicting.

The WCF condition is much weaker than the SCF condition since it only requires that there exists some solution. However, a significant amount of search may be required to find it. Finally, we define problems that may not have a solution.

- **Definition 10:** A resource allocation problem is called **over-constrained (OC)** if it is not necessarily WCF, i.e., all tasks may not necessarily be able to be performed concurrently because resources are insufficient.

Note that $SCF \subset WCF \subset OC$.

4.3 Subclasses of Resource Allocation

Given the above properties, we can define 9 subclasses of problems according to their task complexity and inter-task relationship complexity: SCF and $\binom{n}{n}$ -exact, SCF and $\binom{n}{k}$ -exact, SCF and unrestricted, WCF and $\binom{n}{n}$ -exact, WCF and $\binom{n}{k}$ -exact, WCF and unrestricted, OC and $\binom{n}{n}$ -exact, OC and $\binom{n}{k}$ -exact, OC and unrestricted.

Table 1 summarizes our complexity results for the subclasses of resource allocation problems just defined. The columns of the table, from top to bottom, represent increasingly complex tasks. The rows of the table, from left to right, represent increasingly complex inter-task relationships. Sections 6, 7 and 8 will present proofs of these results. In addition, methods for solving SCF and WCF problems by mapping them into DyDisCSP (defined next in Section 5) are also presented in Section 6 and 7.

Table 1: Complexity Classes of Resource Allocation, $n =$ size of task set Θ , $m =$ size of operation set Ω . Columns represent task complexity and rows represent inter-task relationship complexity.

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

5 Dynamic Distributed CSP

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 (DyDisCSP).

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 variables belonging to a single agent is called a *local constraint*. In contrast, an *external constraint* involves variables of different agents. Solving a DisCSP requires that agents not only solve their local constraints, but also communicate with other agents to satisfy external constraints.

DisCSP assumes that the set of constraints are fixed in advance. This assumption is problematic when we attempt to apply DisCSP 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 DisCSP 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 DisCSP one step further by defining Dynamic DCSP (DyDisCSP). A DyDisCSP is a DisCSP where constraints are allowed to dynamic, i.e., agents are able to add or remove constraints from the problem according to changes in the environment. More formally,

- **Definition 11:** A *dynamic* constraint is given by a tuple (P, C) , where P is an arbitrary predicate that is evaluated by an agent sensing its environment and C is a familiar constraint in DisCSP.

When P is true, C must be satisfied in any DyDisCSP solution. When P is false, C may be violated. An important consequence of dynamic DisCSP 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 DyDisCSP i.e. it is only necessary that *local constraints* be dynamic.

AWC [29] is a sound and complete algorithm for solving DisCSPs. 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 $\{(A_j, v_j), (A_k, v_k), \dots\}$. 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 DisCSP 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.

6 Strongly Conflict Free Problems

In this section, we state the complexity of SCF resource allocation problems as a theorem and map our formal model of the resource allocation problem onto DyDisCSP. 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.

Although our formalism and mappings will address dynamic problems, our complexity analysis here deals with a static problem. 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, since distributed problem solving is no easier than a centralized approach (due to communication delays, limited communication range/bandwidth, etc.), all our complexity results are based on a centralized problem solver.

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. \square

6.1 Mapping SCF Problems into DyDisCSP

We now describe a solution to this subclass of resource allocation problems by mapping onto DyDisCSP. Mapping I is motivated by the following idea. The goal in DisCSP 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 for each variable A_i : $\{O_1^i \text{yes}, O_2^i \text{yes}, O_3^i \text{yes}, O_1^i \text{no}, O_2^i \text{no}, O_3^i \text{no}\}$.

It turns out that even this mapping is inadequate due to ambiguity. Ambiguity arises when an operation can be required for multiple tasks but only one task is actually present. To resolve ambiguity, we desire agents to be able to not only communicate about which operation to perform, but also to communicate for which task they intend the operation. For example in Figure 2.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 when it communicates with A2, so that A2 will be able to activate its correct respective sector. 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 DyDisCSP is defined over a set of n variables,

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

The domain of each variable is given by:

- $\forall A_i \in \mathcal{A}g, \text{Dom}(A_i) = \bigcup_{O_p^i \in \Omega} O_p^i \times T(O_p^i) \times \{\text{yes}, \text{no}\}$.

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.b, Agent A3 has one operation (sector 0) with two possible tasks (target 1 and 2). Although the figure does not show targets in sector 1 and sector 2 of agent A3, let us assume that targets may appear there for this example. Thus, let task T_3 be defined as a target in A3's sector 1 and let task T_4 be defined as a target in A3's sector 2. This means A3 would have 8 values in its domain: $\{O_0^3 T_1 \text{yes}, O_0^3 T_1 \text{no}, O_0^3 T_2 \text{yes}, O_0^3 T_2 \text{no}, O_1^3 T_3 \text{yes}, O_1^3 T_3 \text{no}, O_2^3 T_4 \text{yes}, O_2^3 T_4 \text{no}\}$

A word about notation: $\forall O_p^i \in \Omega$, the set of values in $O_p^i \times T(O_p^i) \times \{\text{yes}\}$ will be abbreviated by the term $O_p^i * \text{yes}$ and the assignment $A_i = O_p^i * \text{yes}$ denotes that $\exists v \in O_p^i * \text{yes}$ 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 is 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, the corresponding agent executes 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 (as defined in Section 5).

- **Dynamic Local Constraint 1 (LC1):** $\forall T_r \in \Theta, \forall O_p^i \in \Upsilon(T_r)$,
LC1(A_i) = (P, C), where Predicate P: O_p^i succeeds.
Constraint C: $A_i = O_p^i * \text{yes}$
- **Dynamic Local Constraint 2 (LC2):** $\forall T_r \in \Theta, \forall O_p^i \in \Upsilon(T_r)$,
LC2(A_i) = (P, C), where Predicate P: O_p^i does not succeed.
Constraint C: $A_i \neq O_p^i * \text{yes}$

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 also change over time, and hence the corresponding DyDisCSP will need to be continually monitored and resolved as necessary.

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

- **External Constraint:** $\forall T_r \in \Theta, \forall O_p^i \in \Upsilon(T_r), \forall A_j \in A$,
EC(A_i, A_j): (1) $A_i = O_p^i T_r \text{yes}$, and
(2) $\forall t_r \in T_r, O_p^i \in t_r, \exists q O_q^j \in t_r$.
 $\Rightarrow A_j = O_q^j T_r \text{yes}$

The EC constraint requires some explanation. It basically says that if A_i detects a task, then other agents in minimal set t_r must also help with the task. In particular, Condition (1) states

that an agent A_i is executing a successful operation O_p^i 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, i.e., O_q^j is an operation that can help perform T_r , the consequent requires A_j to choose operation O_q^j . Note that every pair of variables A_i and A_j have an EC constraint between them. If A_j is not required for T_r , condition (2) is false and EC is trivially satisfied.

6.2 Correctness of Mapping I

We now show that mapping I can be used to model any given SCF Resource Allocation Problem as a DyDisCSP. Theorem II states that our DyDisCSP always has a solution. This means the constraints as defined above are not inconsistent and thus, it is always possible to solve the resulting DyDisCSP. Theorem III then states that if agents reach a solution, all tasks are performed. Note that the converse of the Theorem III does not hold, i.e. it is possible for agents to be performing all tasks *before* a solution to the DyDisCSP 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 some 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 DyDisCSP obtained from Mapping I.

proof: We proceed by presenting a solution to any given DyDisCSP problem obtained from Mapping I.

Let $B = \{A_i \in A \mid \exists T_r \in \Theta_{current}, \exists O_p^i \in \Upsilon(T_r)\}$. B contains precisely those agents whose have an operation that can contribute to some current task. We will first assign values to variables in B , then assign values to variables that are not in B . If $A_i \in B$, we assign $A_i = O_p^i T_r yes$, where $T_r \in \Theta_{current}$ and $O_p^i \in \Upsilon(T_r)$. We know such T_r and O_p^i exist by the definition of B . If $A_i \notin B$, we may choose any $O_p^i T_r no \in \text{Domain}(A_i)$ 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 $\text{EC}(A_i, A_j)$ 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 $\text{EC}(A_i, A_j)$ 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 predicate P of LC2 is not true 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 predicate 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 DyDisCSP. \square

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

proof: Let a solution to the DyDisCSP 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 $T_r \in \Theta_{current}$ be given. 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 choose a task, $T_s \in T(O_p^i)$, and LC1 requires it to assign the value $O_p^i T_s yes$. We will show that A_i could not have chosen incorrectly since we are in solution state. 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. \square

7 Weakly Conflict Free Problems

In this section, we first show the complexity of $\binom{n}{k}$ -exact WCF resource allocation problems and that of unrestricted WCF resource allocation problems. The complexity results are based on a static centralized problem solver but as mentioned, the dynamic distributed case can be no easier. Next, Section 7.2 begins with a discussion of the difficulty in using Mapping I for solving WCF problems. This leads to the introduction of a second mapping, Mapping II, which is able to map WCF problems into DyDisCSP so that it can be efficiently solved using existing distributed constraint reasoning methods.

7.1 Complexity of WCF Problems

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

proof: Since every task in an $\binom{n}{n}$ -exact problem has a single minimal set, there is no backtracking required. Greedily choose the single minimal set for each task. We are guaranteed to find a solution by the WCF condition. \square

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 through 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. \square

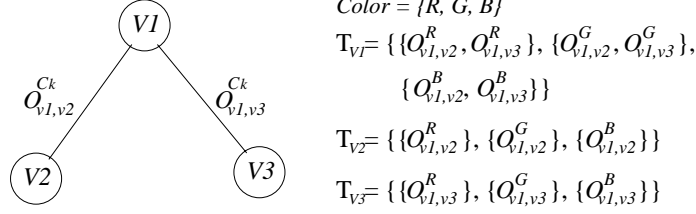


Figure 3: Reduction of graph 3-coloring to Resource Allocation Problems

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

proof: 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 Ag .

Figure 3 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.

7.2 Mapping WCF Problems into DyDisCSP

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 DyDisCSP becomes overconstrained. This is due to the fact that Mapping I requires all agents who can possibly help perform a task to do so. If only three out of four agents are required for a task, Mapping I will still require all four agents to perform the task. In some sense, this results in an overallocation of resources to some tasks. This is not a problem when all tasks are independent as in the SCF case. However, in the WCF case, this overallocation may leave 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 n-ary ($n > 2$) external constraints to the mapping. This is problematic in a distributed situation because there are no efficient algorithms for non-binary distributed CSPs – they require extraordinary amounts of inter-agent communication. Instead, we create a new mapping by extending mapping I to n-ary constraints, then taking its dual representation. In the dual representation, variables correspond to tasks and values correspond to operations. This allows all n-ary constraints to be *local* within an agent and all external constraints are reduced to equality constraints. Restricting n-ary constraints to be local rather than external is more efficient because

it reduces the amount of communication needed between agents. This new mapping, Mapping II, allocates only minimal resources to each task, allowing WCF problems to be solved. Mapping II 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 DyDisCSP is defined as follows:

- **Variables:** $\forall T_r \in \Theta, \forall O_p^i \in \Upsilon(T_r)$, create a DyDisCSP 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.

In this way, we have a variable for each task and a copy of each such variable is assigned to each agent that has operation for that task. For example in Figure 2.b, Agent A1 has one variable, $T_{1,1}$, Agent A2 has one variable $T_{1,2}$, Agent A3 has two variables, $T_{1,3}$ and $T_{2,3}$, one for each task it can perform, and Agent A4 has two variables, $T_{1,4}$ and $T_{2,4}$. The domain of each $T_{1,i}$ variable has five values, one for each of the four minimal sets as described in Section 4, plus the NP value.

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:

Predicate P: O_p^i succeeds

Constraint C: $\exists T_{r,i} \in B T_{r,i} \neq \text{NP}$.

- **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:

Predicate P: O_p^i does not succeed

Constraint C: $T_{r,i} = \text{NP}$

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 $T_{r,i} = t_{r,i}$ and $T_{s,i} = t_{s,i}$, then $t_{r,i}$ and $t_{s,i}$ cannot conflict. NP does not conflict with any value.
- **External Constraint (EC):** $\forall i, j, r T_{r,i} = T_{r,j}$

For example, if Agent A4 assigns $T_{1,4} = \{O_0^1, O_2^2, O_2^4\}$, then LC3 says it cannot assign a minimal set to its other variable $T_{2,4}$, that contains any operation of either Agent A1, A2 or A4. Since $T_{2,4}$ has only one minimal set, $\{O_0^3, O_2^4\}$ which contains Agent A4, the only compatible value is NP. Note that if Target 1 and 2 are both present simultaneously as shown in Figure 2.b, the situation is overconstrained since the NP value will be prohibited by LC1.

7.3 Correctness of Mapping II

We will now prove that Mapping II can be used to represent any given WCF Resource Allocation Problem as a DyDisCSP. As in Mapping I, the Theorem VII shows that our DyDisCSP always has a solution, and the Theorem VIII shows that if agents reach a solution, all current tasks are performed.

Theorem VII: Given a WCF Resource Allocation Problem $\langle Ag, \Omega, \Theta \rangle$, $\Theta_{current} \subseteq \Theta$, there exists a solution to DyDisCSP 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 DyDisCSP. \square

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

proof: Let a solution to the DyDisCSP be given. We want to show that all tasks in $\Theta_{current}$ are performed. We proceed by contradiction. Let $T_r \in \Theta_{current}$ be a task that is not performed in the given solution state. Condition (i) of the **Notification Assumption** says some operation O_p^i , required by T_r will be executed and (by definition) succeed. LC1 requires the corresponding agent A_i , to assign a minimal set to some task which requires O_p^i . There may be many choices of tasks that require O_p^i . Suppose A_i chooses a task T_s . So A_i assigns a minimal set, say t_s , to the variable $T_{s,i}$. The EC constraint will then require that all other agents A_j , who have a local copy of T_s called $T_{s,j}$, to assign $T_{s,j} = t_s$. In addition, if A_j has an operation O_q^j in the minimal set t_s , it will execute that operation. Also, we know that A_j is not already doing some other operation since t_s cannot conflict with any other chosen minimal set (by LC3).

We now have two cases. In case 1, suppose $T_s \neq T_r$. Condition (ii) of the **Notification Assumption** states that T_r is the only task that both requires O_p^i and is actually present. Thus, T_s cannot be present. By definition, if T_s is not present, it cannot be performed. If it cannot be performed, there cannot exist a minimal set of T_s where all operations succeed (def of “performed”). Therefore, some operation in t_s must fail. Let O_q^j be an operation of agent A_j that fails. Since A_j has assigned value $T_{s,j} = t_s$, LC2 is violated by A_j . This contradicts the fact we are in a solution state. So case 1 is not possible. This leaves case 2 where $T_s = T_r$. Then, all operations in t_s succeed and T_r is performed. We assumed T_r was not performed, so by contradiction, all tasks in $\Theta_{current}$ must be performed. \square

8 Overconstrained Problems

In this section, we state the complexity of overconstrained resource allocation problems where sufficient resources may not be available to complete all present tasks. In overconstrained problems, the problem is to find $\Theta_{sat} \subseteq \Theta_{current}$, such that $\langle \mathcal{A}g, \Omega, \Theta_{sat} \rangle$ has a solution and $|\Theta_{sat}|$ is maximized. In other words, we wish to choose a subset of the tasks so that the maximum number of tasks are completed. We show complexity of this problem to be NP-Complete.

Theorem IX: Overconstrained resource allocation is NP-Complete.

proof: We show that a special case is NP-Complete, namely, assume the problem is $\binom{n}{n}$ -exact. If we are given a subset of Θ , we can determine if the RAP is solvable in linear time. So the problem is in NP. To show this problem is NP-hard, we will reduce from the INDEPENDENT-SET problem. INDEPENDENT-SET is defined as follows: Let $G = (V, E)$ be an undirected graph, and let $I \subseteq V$. The set I is *independent* if whenever $i, j \in I$ then there is no edge between i and j . The goal is to find the largest independent set in graph G .

The reduction from INDEPENDENT-SET is as follows. For each node i , we create a task T_i with exactly one minimal set. For each edge, $i, j \in E$, we create an operation $O_{i,j}$ and add it to the minimal set of T_i and T_j . Finally, create one agent for each operation. We can see that two tasks conflict if and only if their nodes in G have an edge between them. Let $I \subseteq V$ be a solution to the INDEPENDENT-SET. Let Θ_{sat} be the set of tasks corresponding to the nodes in I . There is no edge between any $i, j \in I$, so there cannot be any operation in common between T_i and T_j in Θ_{sat} . Thus, Θ_{sat} is the solution to the resource allocation problem. The reverse direction is similar. \square

In overconstrained domains, agents must reason about the quality of alternative allocations and find the allocation that satisfies the most tasks. Methods to solve such problems using CSP techniques are not currently available. Part of the difficulty is that sufficiently advanced distributed constraint reasoning techniques which are able to handle optimization problems have not yet been developed. Indeed, this is an active area of on-going research [13].

9 Experiments

The purpose of the experimental results in this section is to demonstrate our formalism, mapping and solution methodology in an actual implementation using the distributed sensor network problem. This is important to ground the theory and understand its computational properties.

9.1 Distributed Resource Allocation

We empirically investigate some of the characteristics of a resource allocation problem that make it difficult or easier to solve using the distributed constraint methodology outlined in this paper. In particular, we evaluate the effect of increasing the number of resources (agents) while keeping the number of tasks constant and the effect of increasing the number of tasks while keeping the number of resources constant. Finally, we examine a heuristic whereby the most “useful” resources (i.e., agents that can contribute to one of many possible tasks) are allocated first. The intuition behind this heuristic is that when highly constrained resources are allocated early, if they happen to be allocated incorrectly, it will be discovered quickly and the error can be corrected. This heuristic is realized in the constraint framework by ordering variables so that the variables with the

most constraints are prioritized higher in the variable instantiation order. Figure 4 shows a sensor configuration and associated priority ordering where the most constrained agents have highest priority.

We experiment with distributed constraint problems obtained from applying Mapping II to various target and sensor configurations. Experimenting with Mapping II is appropriate because it addresses WCF resource allocation problems, which are more complex than the SCF problems addressed by Mapping I as seen in Table 1. Note that these are not toy examples but rather real problems from hardware sensor configurations similar to that shown in Figure 4. The problems are solved using state-of-the-art distributed constraint reasoning algorithms [29] [13]. To measure progress over time in a system that is completely asynchronous and distributed, previous methods in distributed constraint reasoning of counting "synchronous cycles" [29] were not realistic and acceptable. Instead, we measure the number of search cycles by counting the maximum number of backtrack messages sent by any agent in the system. In the following, we present our hypotheses for the experiments and then present the experimental results.

Hypothesis 1: Mapping II scales well as number of resources (agents) increases but number of tasks remains constant. In terms of the constraint graph, increasing the number of agents corresponds to increasing the number of variables and constraints in the problem. However, keeping the number of tasks constant means that choosing assignments for the new constraints is easy. Few cycles should be required between agents with these types of variables. Figure 5(left) plots the experimental results with two configurations, chain and grid, where we keep the number of interacting tasks constant at one, two, three or four, while increasing the number of agents. The number of agents is shown on the x-axis and number of cycles to solution on the y-axis. Figure 5(right) plots the total number of messages exchanged. The scaling is almost constant so we see that our hypothesis is verified. This is a desirable property of our mapping because it means that the number of agents, by itself, does not adversely affect time to solution.

Hypothesis 2: The Distributed Resource Allocation Problem becomes harder as number of tasks increases but number of resources (agents) remains constant. However, this effect can be mitigated by allowing the most constrained resources to be allocated first. If the number of tasks increases while the number of resources remains constant, the allocation problem should become more difficult to solve and a larger number of cycles will be necessary to reach the solution. Figure 6 (left) shows results for two sensor configurations. The chain configuration has 28 variables with domain size 5 and 94 constraints and the grid configuration has 36 variables with domain size 5 and 154 constraints. The graph shows that the number of cycles (y-axis) increases exponentially as more interacting tasks (x-axis) are added. Figure 6 (right) shows similar results for total number of messages exchanged. Variables are ordered arbitrarily. However, Figure 7 shows the same experiment but the variable ordering is modified so that most constrained agents are higher in the ordering. The results show that this priority ordering is superior to the arbitrary ordering from Figure 6 in terms of both cycles and number of messages. This supports our hypothesis.

9.2 Target Tracking using DyDisCSP

We have also applied our solution methodology to the distributed sensor network problem described in Section 2 using both real hardware and a detailed simulator. Although it was infeasible to do extensive experiments directly on the hardware due to unavailability of the equipment in our

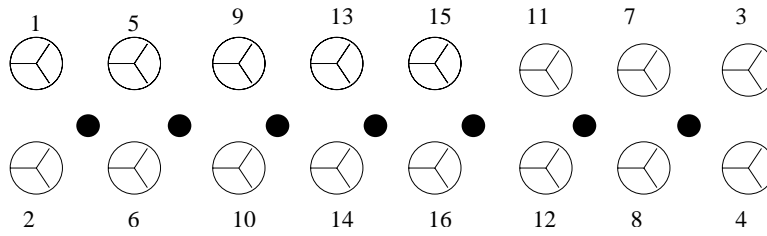


Figure 4: A chain configuration of sensors and targets. Sensors are numbered according to their priority order where a higher number indicates higher priority. If the sensors of higher priority in the middle of the chain choose their sectors first, a global solution can be found more quickly.

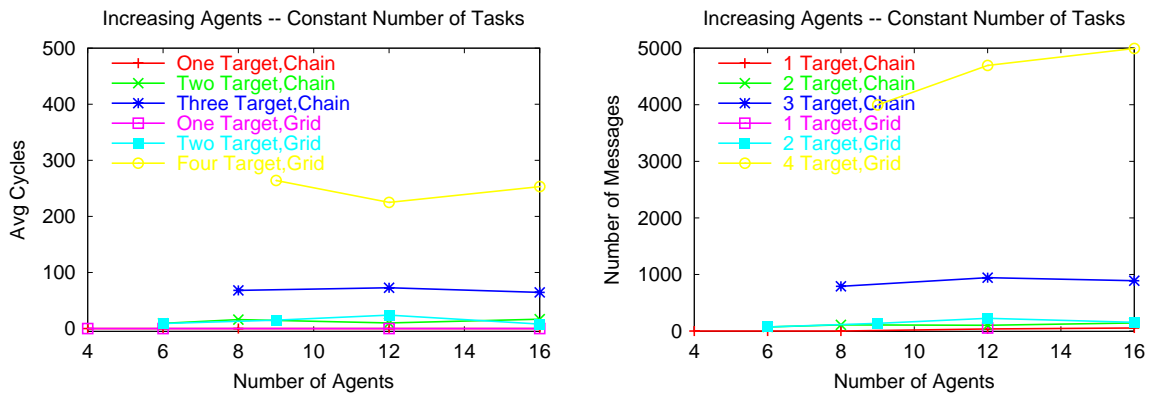


Figure 5: Number of cycles (left) and number of messages (right) with increasing number of agents and constant number of tasks

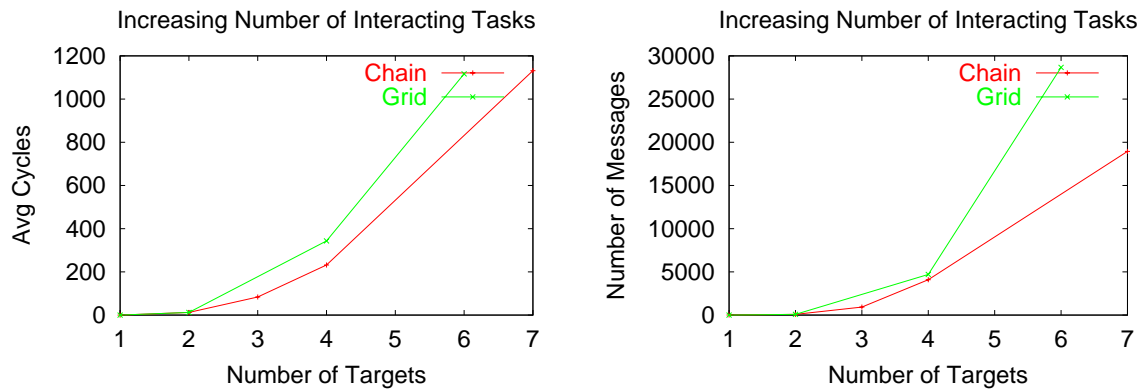


Figure 6: Number of cycles (left) and number of messages (right) with increasing number of interacting tasks

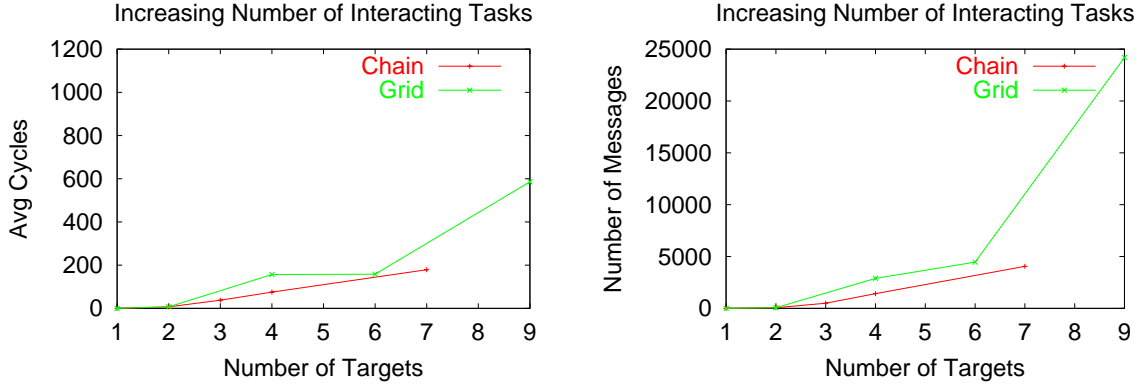


Figure 7: Heuristic priority ordering. Number of cycles (left) and number of messages (right) with increasing number of interacting tasks. Number of cycles and number of messages decreases as compared to the arbitrary priority ordering from Figure 6.

lab, we were able to successfully complete evaluation trials conducted in government labs where agents collaboratively tracked a moving target. In particular, our DyDisCSP implementation and the mapping introduced in Section 7 was successfully run on eight communicating hardware sensor nodes (see Figure 2.a). This target tracking requires addressing noise, communication failures, and other real-world problems; this was done outside the DyDisCSP framework and hence not reported here. However, the hardware trials are significant since they provide evidence that the assumptions underlying our solution methodology do not violate crucial real-world constraints.

In addition to hardware experiments, a 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 DyDisCSP formalization. Table 2 presents some experimental results from the implementation in the simulator. One evaluation criteria in distributed sensor networks is how accurately targets are tracked. 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 up to 3 units as acceptable. 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.

10 Summary and Future Work

In this paper, we have made three contributions. First, we proposed a formalization of distributed resource allocation that is expressive enough to represent both dynamic and distributed aspects of the problem. Second, we defined different categories of difficulties of the problem and presented detailed complexity results for each category. Third, in order to address these formalized prob-

Problem 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 2: Results from sensor network domain for dynamic resource allocation problems.

lems, we define the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDisCSP) and present a general strategy for mapping distributed resource allocation into DyDisCSP. Through both theoretical analysis and experimental verifications, we have shown that our theoretical framework for 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 DyDisCSP, with automatic guarantees for correctness of the solution.

A variety of future issues remain in order to fully address the distributed resource allocation problem. First, agents may have insufficient resources available (e.g. the *over-constrained* (OC) resource allocation problems defined in Section 4) or resources may have varying costs. In such domains, agents must reason about the quality of alternative allocations and find the allocation that satisfies the most tasks for the least cost, for example. Methods to solve such problems using CSP techniques are not currently available but is an active area of on-going research [13]. Second, agents may encounter sensor noise or agent failure. We desire the system to be able to continue to perform tasks by transferring tasks to more capable agents or recruiting new agents if needed. Resource allocation methods that are robust these problems are missing. Finally, agents may need to solve difficult allocation problems under severe real-time constraints. For example, the environment may impose hard deadlines after which an obtained solution is useless. In order to be effective in such domains, agents must be able to explicitly reason about the amount of time they have available to solve a given problem and obtain solutions accordingly.

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. We thank Hyuckchul Jung for his contributions to an earlier conference version of this paper. We also thank Shriniwas Kulkarni and Paul Scerri for their contributions to the experimental results and implementation in the distributed sensor network domain. We also thank Makoto Yokoo for useful discussions related to distributed constraint reasoning during his visit to the Information Sciences Institute, and Bhaskar Krishnamuchri and Bart Selman for useful discussions related to select complexity results.

References

- [1] M. Chia, D. Neiman, and V. Lesser. Poaching and distraction in asynchronous agent activities. In *International Conference on Multiagent Systems*, 1998.
- [2] S.E Conry, K. Kuwabara, V.A. Lesser, and R.A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Trans. Systems, Man and Cybernetics*, 1991.
- [3] K. Decker and V. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 2(4), 1993.
- [4] K. Decker and J. Li. Coordinated hospital patient scheduling. In *Proceedings of International Conference on Multi-Agent Systems*, 1998.
- [5] C. Frei and B. Faltings. Resource allocation in networks using abstraction and constraint satisfaction techniques. In *Proc of Constraint Programming*, 1999.
- [6] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.
- [7] H. Jung, M. Tambe, and S. Kulkarni. Argumentation as distributed constraint satisfaction: Applications and results. In *Proceedings of the International Conference on Autonomous Agents*, 2001.
- [8] H. Kitano, S. Todokoro, I. Noda, H. Matsubara, and T Takahashi. Robocup rescue: Search and rescue in large-scale disaster as a domain for autonomous agents research. In *Proceedings of the IEEE International Conference on System, Man, and Cybernetics*, 1999.
- [9] V. Lesser and D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3), 1983.
- [10] J. Liu and K. Sycara. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of International Conference on Multi-Agent Systems*, 1996.
- [11] P. Mesequer and M. A. Jiménez. Distributed forward checking. In *Proceedings of CP-00 Workshop on Distributed Constraint Satisfaction*, 2000.
- [12] P. J. Modi, H. Jung, W. Shen, M. Tambe, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Principles and Practice of Constraint Programming*, 2001.
- [13] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for general distributed constraint optimization. In *Proc of Autonomous Agents and Multi-Agent Systems Workshop on Distributed Constraint Reasoning*, 2002.
- [14] V. Parunak, A. Ward, M. Fleischer, J. Sauter, and T. Chang. Distributed component-centered design as agent-based distributed constraint optimization. In *Proc. of the AAAI Workshop on Constraints and Agents*, 1997.

- [15] T. Sandholm and Subhash Suri. Side constraints and non-price attributes in markets. In *International Joint Conference on Artificial Intelligence Workshop on Distributed Constraint Reasoning*, 2001.
- [16] P. Scerri, P.J. Modi, W. Shen, and M. Tambe. Are multiagent algorithms relevant for robotics applications? a case study of distributed constraint algorithms. In *ACM Symposium on Applied Computing*, 2003.
- [17] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 1994.
- [18] M.C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*.
- [19] L-K. Soh and C. Tsatsoulis. Reflective negotiating agents for real-time multisensor target tracking. In *International Joint Conference On Artificial Intelligence*, 2001.
- [20] Katia Sycara, Steven F Roth, Norman Sadeh-Konieczpol, and Mark S. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:1446–1461, 1991.
- [21] BAE Systems. Ecm challenge problem, <http://www.sanders.com/ants/ecm.htm>. 2001.
- [22] R. Vincent, B. Horling, V. Lesser, and T. Wagner. Implementing soft real-time agent control. In *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.
- [23] T. Wagner and V. Lesser. Criteria-directed heuristic task scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 1998.
- [24] W. Walsh and M. Wellman. A market protocol for decentralized task allocation. In *International Conference on Multi-Agent Systems*, 1998.
- [25] W. Walsh, M. Yokoo, K. Hirayama, and M. Wellman. On market-inspired approaches to propositional satisfiability. In *International Joint Conference on Artificial Intelligence*, 2001.
- [26] Michael Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, pages 1–23, 1993.
- [27] M. Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer, 2001.
- [28] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 1998.
- [29] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of International Conference on Multiagent Systems*, 1998.

- [30] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.