12-17-2007

A Unified Framework for Solving Multiagent Task Assignment Problems

Kevin Cousin

A Unified Framework For Solving
Multiagent Task Assignment Problems

DISSERTATION

Kevin Couśin, Major, USAF

AFIT/DCS/ENG/08-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/DCS/ENG/08-01

A Unified Framework For Solving
Multiagent Task Assignment Problems

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Kevin Couśin, BS, MA

Major, USAF

December 2007

A Unified Framework For Solving Multiagent Task Assignment Problems

Kevin Cousin, BS, MA

Major, USAF

Approved:

_____   __28 Nov 07__

Dr. Gilbert L. Peterson   Date

Chairman

_____   __28 Nov 07__

Dr. John O. Miller   Date

Dean's Representative

_____   __28 Nov 07__

Dr. Gary B. Lamont   Date

Member

_____   __28 Nov 07__

Maj Robert J. Weber   Date

Member

Accepted:

_____   __17 Dec 07__

M. U. THOMAS   Date

Dean, Graduate School of Engineering
  and Management
Air Force Institute of Technology

AFIT/DCS/ENG/08-01

Abstract

Multiagent task assignment problem descriptors do not fully represent the complex interactions in a multiagent domain, and algorithmic solutions vary widely depending on how the domain is represented. This issue is compounded as related research fields contain descriptors that similarly describe multiagent task assignment problems, including complex domain interactions, but generally do not provide the mechanisms needed to solve the multiagent aspect of task assignment.

This research presents a unified approach to representing and solving the multiagent task assignment problem for complex problem domains. Ideas central to multiagent task allocation, project scheduling, constraint satisfaction, and coalition formation are combined to form the basis of the constrained multiagent task scheduling (CMTS) problem. Basic analysis reveals the exponential size of the solution space for a CMTS problem, approximated by $O(2^{n(m+n)})$ based on the number of agents and tasks involved in a problem. The shape of the solution space is shown to contain numerous discontinuous regions due to the complexities involved in relational constraints defined between agents and tasks. The CMTS descriptor represents a wide range of classical and modern problems, such as job shop scheduling, the traveling salesman problem, vehicle routing, and cooperative multi-object tracking.

Problems using the CMTS representation are solvable by a suite of algorithms, with varying degrees of suitability. Solution generating methods range from simple random scheduling to state-of-the-art biologically inspired approaches. Techniques from classical task assignment solvers are extended to handle multiagent task problems where agents can also multitask. Additional ideas are incorporated from constraint satisfaction, project scheduling, evolutionary algorithms, dynamic coalition formation, auctioning, and behavior-based robotics to highlight how different solution generation strategies apply to the complex problem space.

iv

Furthermore, a new approach to solving the CMTS problem using a distributed system shows how to scale the adapted algorithms to solve increasingly larger domain problems. The distributed approach introduces several methods for decomposing a problem and recomposing subsolutions into a complete solution without significantly compromising solution quality. On the other hand, decomposition techniques show methods to reduce the search space by several orders of magnitude allowing for improved search efficiency.

Finally, experimentation in the search and recovery problem domain instance for the CMTS problem class demonstrates the ability to use the CMTS descriptor as a unified descriptor for a variety of algorithms. The results of various algorithms in turn empirically demonstrate key characteristics of the solving the search and rescue problem with and without enhanced heuristics, such as those from dynamic coalition formation and behavior-based robotics. Ultimately, the efficiency and effectiveness of solving the problem in a distributed process empirically demonstrates methods to dramatically reduce search space complexity while maintaining acceptable quality solutions in this domain. Experiments in alarm handling and search and rescue problem domains show search space reductions of 10^{39} and 10^{106} solution points respectively while producing competitive solutions.

AFIT/DCS/ENG/08-01

For the family, all of the Couśins, everywhere.

Acknowledgements

I would like to foremost thank my advisor Dr. Bert Peterson for the guidance, enlightenment and opportunities he has given to me. I learned more about artificial intelligence and the business and art of doing research under his watch than I thought possible.

I also thank my committee members Major Bob Weber for helping me get into the program, Dr. John Miller, and especially Dr. Gary Lamont whom I hold in the highest regard for his expertice and experience. I also acknowledge the camaraderie and tremendous support I have received from my classmates, ENG faculty and staff, AFIT staff members, and commanders and mentors from past assignments who made this possible in so many ways.

I deeply appreciate the research support and inspiration I received from Dr. Elizabeth Downie of the Dayton Area Graduate Studies Institute, Dr. Mikel Miller and Dr. Jason Campbell from the Air Force Research Laboratory, and Dr. Roger Quinn and Alex Boxerbaum of Case Western Reserve University. Their professionalism and continual approval motivated this work in many ways.

Most importantly, I thank my beautiful wife for encouraging me from day one, my children who are as proud of me as I am of them, and my parents' and brother's support. There is nothing more important to me than having such a supportive family.

Kevin Couśin

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

List of Abbreviations

A Unified Framework For Solving
Multiagent Task Assignment Problems

I. Introduction to Multiagent Task Assignment

The task assignment problem belongs to the general class of satisfiability problems [89]. Essentially, one pairs every task in a job to a worker capable of performing that task. These types of problems have been the focus of active research for several decades, dating back as early as the 1950s [73]. Researchers have since studied dozens of aspects of the task assignment problem [22, 31, 42] and provided numerous ways to solve it [28, 74, 81, 105, 107]. With the growing size of task assignment problems and the new complexities introduced over the years, multiagent task assignment problems have become an important focus of task assignment research [45, 120]. This research consolidates describing and solving the task assignment problem into a unified framework approach.

However, the problem descriptors of the majority of published work do not adequately represent realistic modern problems. Taxonomies of the task assignment problem [31, 44] which define key aspects of the general problem domain show that current descriptors generally work within key subsets of task assignment taxonomies, but few encompass every aspect of the problem domain. Particularly, modern problems support notions of multi-capable agents concurrently performing groups of tasks, each task possibly requiring multiple agents (perhaps in great numbers). Many problem descriptors only provide for simpler problem domains with one-to-many, many-to-one or one-to-one relationships between agents and tasks.

Even fewer problem descriptors involve relational constraints or stochastic environments. Relational constraints restrict how agents are assigned to tasks or the order that tasks are performed. For example, it seems crucial that the frame of a vehicle is assembled before the interior is installed (task prerequisites), and that the

1

frame assembler is not working on the same vehicle at the same time as the interior installer (agent inhibition). Of course, in a stochastic environment, the existence or some quality of an agent or task could be unknown during the assignment process.

But several solutions are available to solve specific subsets of the multiagent task assignment problem as defined by the many known problem descriptors. Most algorithms employ standard approaches [89] to solving satisfiability problems, such as solutions for the set covering or set partitioning problem [57, 107]. Some algorithms use novel approaches to tree-based searches [103, 105], iterative solution construction [30, 86, 112] or evolutionary approaches [6, 33, 61]. Yet others use economic [13, 46, 121] or social-behavioral based rules [36, 96] to develop solutions, such as auctions or impatience.

These algorithmic approaches are generally sensitive to the structure of the specific problem it is designed to solve and, if the algorithm is unmodified, fail when the problem structure changes. This unfortunately leads to the development of a variety of algorithms that solve similar problems but require substantive changes since they look at different aspects of the problem. In general, there is a need [45, 92] for a comprehensive algorithmic approach to solve the overarching multiagent task assignment problem.

1.1   A Unifying Approach

The central idea behind defining a framework is a supporting structure around which something can be built [19, 88, 100]. A unified framework combines the perspective ideas from several related frameworks in a way such that systems developed from one perspective remain consistent in the unified framework. The popularity of unified frameworks is prevalent across several disciplines but is especially present in multiagent research [3, 48, 82]. In many cases, a unified framework combined definitions that identify by multiplicity [44], represent a single foundational structure for developing various offspring algorithms [98, 108, 126], or define a set of core algorithms that can be combined to treat complex problems [25, 65].

This research proposes a unified framework to comprehensively define and solve the broad number of multiagent task assignment problems. This includes a unified problem descriptor capable of describing every taxonomic description found in [31, 44] based on the number of agents and tasks in a problem with different types of relational constraints and dynamic environment factors. Additionally, algorithmic solutions for problems using the unified descriptor are provided to generate optimal and approximate solutions while handling wide variations in problem domains without modification.

First, a problem descriptor is developed based on ideas central to multiagent task allocation [44], scheduling [83], distributed constraint satisfaction [125] and coalition formation [105]. Research in task allocation provides a solid basis for representing problems with everything from one-to-one to many-to-many agent-to-task relationships. However, ideas from project scheduling and constraint satisfaction are needed in order to properly represent relational complexities in the problem domain, such as task ordering, agent inhibition, and restricting solutions to acceptable measurable values. Combined, these ideas form the basis of a new problem class, the constrained multiagent task scheduling (CMTS) problem. Ideas from methods used to solve task allocation, project scheduling, and constraint satisfaction are incorporated into the problem descriptor with concepts from coalition formation and behavior-based robots.

The expressive power of the unifying CMTS problem descriptor is demonstrated by using identical notation to describe several classical and modern problems commonly found in single- and multi-agent task assignment problems. The re-expressed problems include the optimal assignment problem [73], flow and job shop [7, 32] scheduling, the multiple traveling salesman problem [90, 116], the vehicle routing problem [116], and cooperative multi-robot observation of multiple moving targets (CMOMMT) [97]. Since each of these describes a rather different type of problem, having a problem descriptor capable of single-handedly representing all of them provides grounds to claim significant multiagent problem domain unification.

Additionally, this research presents a select suite of standard algorithms that are modified to solve CMTS-described problems. The selected group shows the varying effectiveness of different algorithmic approaches to solving the CMTS problem. First are the direct approaches to generating a solution without regard for solution quality, the random search and task sequencer. Then follow the constructive algorithms, state-space searches, that rely on solution quality to build solutions, often by employing heuristics. These include two types of upgraded auctioning algorithms, breadth-first and depth-first search, A*, two types of beam search, ant colony optimization, and reinforcement learning. Finally, two evolutionary algorithms, the genetic algorithm and particle swarm optimization, show how to apply solution-space searches to the problem.

The framework also contains a distributed approach to solving CMTS problems to show how to improve efficiency. Improved efficiency is vital to scaling the standard algorithms to larger problem domain instances. The distributed process first involves decomposing problems into smaller "chunks" which are independently solved. The methods used by the framework offer four ways to decompose a problem: by each task or agent individually, by general classes of domain activities (where agents are mapped to tasks via a set of domain operations), or by dividing the agents and tasks into similarly sized subproblems. After each decomposed problem is solved, a recomposition combines the subsolutions into a complete solution for the decomposed problem.

1.2   Contribution

Ultimately, this research contributes a unified framework which 1) models a wide range of problems through the CMTS problem descriptor and 2) presents a core suite of distributable algorithmic techniques. The framework is forward-looking, allowing current and future task assignment approaches to utilize, by agglomeration, various novel, independently-developed techniques that contribute to improving some aspect of the defining or solving of the multiagent task assignment problem. The framework

presented here incorporates several fundamental concepts: task allocation, project scheduling, relational constraints, and coalition formation. It provides a means to include metrics from dynamic coalition formation, behavior-based characteristics and partially-observable environment constraints directly into the problem domain and algorithmic approaches. Ideally, researchers examining any specific portion of the general problem domain can plug their ideas into the framework and test it with a variety of CMTS-ready problem definitions and algorithms.

Also contributed is the first CMTS class of problems, autonomous search and recovery (ASAR), for solving the problem of streamlining cooperative robotic search and rescue/recovery. This problem is applicable to a wide range of operations, such as military or civilian search and rescue, hostage crisis action, and mobile target tracking. The solutions generated by the framework provide a systematic and flexible approach for allowing robots to autonomously find, track, and deliver items of interest, which can remove people from harm's way and potentially save lives.

1.3   Outline

This dissertation is organized as follows. Chapter II presents a literature review of concepts central to the unified approach to representing and solving multiagent task assignment problems, specifically detailing the different definitions that are combined to form the CMTS problem descriptor. A brief summary of some of the different methodologies used to solve various aspects of the definitions are also highlighted.

Chapters III, IV and V present the unified model for representing and solving multiagent task assignment problems. Chapter III fully develops the CMTS problem descriptor by combining the definitions from Section 2.1 into a single representation. Section 3.2 focuses on solution representations and methods for validating a solution. Section 3.3 provides an analysis of the solution space involved in CMTS problems, including various measurements. Section 3.4 completes the problem domain modeling by representing a number of classical and modern multiagent task assignment problems in CMTS form, showing the descriptive range expected of a unified modeler.

5

The chapter concludes with the definition of a problem class designed for autonomous search and rescue.

Methods presented in Chapter IV solve the CMTS-represented problems, including algorithmic solutions by direct search, informed solution construction, and solution-space searching. Section 4.8 briefly discusses additional, similar methods that can be modified to solve the CMTS problem. Section 4.9 presents a method to incorporate uncertain information in a solution process.

Chapter V presents methods for solving a multiagent task assignment problem using a distributed system. Based on two types of decomposition strategies described in Section 5.1, Section 5.2 introduces four ways to decompose a problem based on problem domain elements and characteristics of the distributed system. This is followed by a discussion on how to recompose a complete solution from a distributed system. These methods are employed in Section 5.4 which presents the standard distributed approach and a new meta-heuristic for online learning while developing a solution among a distributed system. The chapter concludes with the development of a distributed algorithm designed to generate optimal solutions.

Chapter VI details three experiments applying many of the revised algorithms from Chapter IV. Sections 6.1 and 6.2 demonstrate the effectiveness and efficiency of various task assignment methods for simple domain problems. Section 6.3 applies CMTS-ready algorithms to the ASAR problem detailed in Section 3.4.5, showing the general performance of algorithms in serial experiments in Section 6.3.3 and in distributed experiments in Section 6.3.4. The last section details some additional behavioral observations discovered during experimentation.

Finally, Chapter VII provides conclusive results about the unified multiagent task assignment model. Section 7.2 outlines additional directions for continuing this research in future projects.

II. Current Research in Task Assignment

This research primarily contributes to the intersection of multiagent research and task assignment by incorporating ideas from outside these two tracks of interest with modern approaches to solving multiagent task assignment problems. Specifically, the most relevant and fundamental ideas of describing and solving task allocation, task scheduling, and constraint satisfaction are summarized in order to better understand the components of the unified framework. Then, to gain the multiagent perspective, ideas behind the foremost multi-robot approaches and dynamic coalition formation are presented. The incorporation of each of these ideas into a single unified approach to describe and solve multiagent task assignment problems are respectively addressed in Chapters III and IV.

2.1 Defining Features of the Task Assignment Problem

The problem descriptor component of the unified framework involves combining elements from three distinct areas of research: task allocation, scheduling, and constraint satisfaction. However, the framework extends to support additional multiagent concepts such as coalition formation and robot behaviors. This section provides a brief summary of the different techniques in current literature on multiagent task allocation, resource constrained project scheduling, distributed constraint satisfaction, and dynamic coalition formation.

2.1.1 Multiagent Task Allocation. The task allocation problem fits within the class of NP-hard satisfiability problems [89], which includes problems such as job shop scheduling [32]. The deterministic multiagent task allocation problem is formally defined as a set of machines M, a set of tasks Y, each having a corresponding weight w_y, and a real-valued utility value $u_{xy} \geq 0$. The solution takes the general form $S = \{(x \in M, y \in Y)\}$ where (x, y) implies that agent x is assigned to task y. The

(a) ST-SR Class   (b) ST-MR Class

(c) MT-SR Class   (d) MT-MR Class
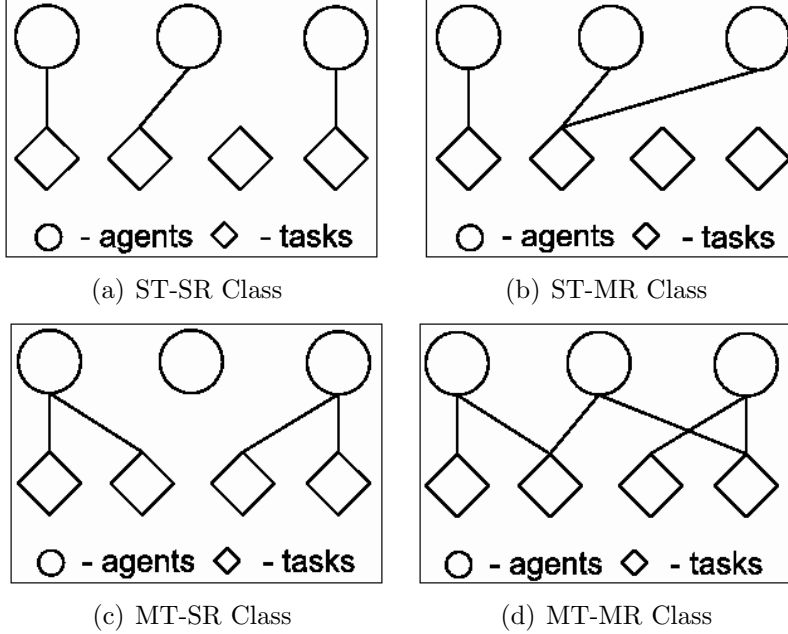
Figure 2.1: MRTA Class Examples.

objective is to construct a solution S that optimizes the utility value U, where

$$U(S) = \sum_{(i,j) \in S} u_{ij} w_j. \tag{2.1}$$

Optimizing the utility value depends on the problem domain. For example, in problems that involve cost functions, the utility is often maximized in terms of $profit - cost$ or minimized in terms of $cost$.

Gerkey develops a multi-robot task allocation (MRTA) taxonomy [42, 45] to analyze how multi-robot frameworks apply to the multiagent task allocation problem. The taxonomy accurately describes a fairly comprehensive set of multi-robot task allocation approaches based on three attributes commonly seen in multiagent task allocation problems. These attributes form axes in the taxonomy to produce eight different classes of problems.

The first attribute is the number of tasks a robot can concurrently perform. The taxonomy divides this into ST for single-task robots and MT for multi-task robots.

Table 2.1: MRTA Taxonomy [42, 45].

Class	Equivalent Problem
ST-SR-IA	Optimal Assignment Problem
ST-SR-TE	Scheduling Problem
ST-MR-IA	Set Partitioning Problem or Coalition Forming
ST-MR-TE	Dynamic Coalition Forming
MT-SR-IA	ST-MR-IA
MT-SR-TE	ST-MR-TEA
MT-MR-IA	Set Covering Problem
MT-MR-TE	Scheduling multi-processor tasks on multi-purpose machines

The distinction is that single task robots perform only one task at a time whereas multi-task robots can concurrently perform more than one. If any robot in a group is a multi-task robot, then the group as a whole falls into the MT class of problems.

The second attribute similarly groups by the number of robots required to perform a task. This class is divided into SR for single-robot tasks and MR for multi-robot tasks. Again, if any task in a set is a multi-robot task, then the set belongs in the MR class of problems.

The third attribute classifies how much of the allocation can be performed at any given moment. This class is divided into IA for instantaneous assignment and TE[1] for time-extended assignment. IA implies that no planning for future allocations occurs given the information (robots, tasks and environment) available at some time. For example, this happens when the total number of tasks may be unknown. TE, on the other hand, permits future modeling of allocations since there is enough information to do so.

Figure 2.1 illustrates agent-to-task mappings for the ST-SR, ST-MR, MT-SR and MT-MR classes.

Each of the taxonomy categories strongly correlates to classical assignment class problems, summarized in Table 2.1. The instantaneous assignment nature of the ST-SR problem represents the well-studied optimal assignment problem (OAP) for which

---

[1]The designation is denoted TA in [45].

there are numerous problem instances across a wide variety of research subjects, from the original stable marriage problem [38] to modern economic problems. The time-extended version of this class is regarded as a general scheduling problem since tasks are performed in some order. There is also a wealth of scheduling problems spanning several subjects.

Single-task robots in a multi-robot task problem, or conversely, multi-task robots in a single task problem, along the instantaneous taxonomy axis represent the set partitioning problem (SPP), the division of a set into a set of non-intersecting subsets. The ST-MR version also applies to coalition formation, further discussed in Section 2.1.4. Along the time-extended axis, the classifications represent dynamic coalition formation, which reforms coalitions from existing coalitions to meet a goal. Several problem domain sources exist for coalition-based problems, for example, cooperative multi-robot observation of multiple moving targets (CMOMMT) [97], the multiple traveling salesman problem (MTSP), or the vehicle routing problem (VRP) [116].

Multi-task robots in instantaneous assignment multi-robot task problems represent the set covering problem (SCP) where, similar to SPP, a set is divided into a set of subsets. However, the subsets are not necessarily unique. The TE axis forms the basis for most real world problems: multi-capable entities performing tasks requiring multiple skill sets using a schedule to order tasks. In terms of a problem, the best description is the scheduling of multiprocessor tasks on multi-purpose machines. Sources of domain problems for MT-MR class problems generally stem from appropriately extending ST-MR and MT-SR problems, such as cooperative box-pushing or CMOMMT.

Although the taxonomy collectively describes a significantly large group of multiagent task assignment problems, each class is equated to a specific type of problem class which is generally different from other classes. This carries a drawback of requiring multiple algorithms just to solve MRTA problems. Furthermore, the taxonomy does not address stochastic environments and estimations for the expected value of

10

the utility of an allocation solution. Unifying the problem descriptions of the classes in MRTA closes the gap on producing a framework that describes any instance of a multiagent task assignment problem.

2.1.2 Resource Constrained Project Scheduling. The project scheduling problem (PSP) involves assigning resources to a number of tasks or jobs in an ordered manner, often based on the allocation of jobs to resources at, or during, explicit times. One type of scheduling problem related to multiagent task assignment is resource-constrained project scheduling problem (RCPSP) [67, 77]. The RCPSP has a set of tasks Y with precedence relationships and a set of resources M with resource capacities. Tasks are assigned to resources based on meeting resource requirements and a starting time. The solution takes the form of a vector of start times for each of the tasks with the objective of minimizing the total time, or makespan, of completing the set of tasks.

The RCPSP presents definitions that constrain the allocation of tasks to resources (agents) which are not a feature available in the MRTA taxonomy. Since the single-task agent, single-agent task (ST-SR) taxonomy class generally describes scheduling when using time-extended features, merging the contraint features of RCPSP with ST-SR-TE definitions produces one definition for expressing value-based constraints in multi-robot task allocation problems.

A majority of the problem domains studied for PSPs are found in operations research. The operations research community has established several repositories of different kinds of problems, for instance, the PSPLIB [68, 69] which contains many PSPs. This library provides domain problem instances for many types of scheduling problems, including RCPSP, and typically includes known solutions useful for benchmarking [51, 66, 102] various approaches. Additionally, [114] contains several benchmark instances for basic scheduling and RCPSP problems.

Additionally, a number of research experiments examine the effects of uncertainty in scheduling projects. The use of partially-observable Markov decision prob-

lems (POMDPs) is applied to the RCPSP in [117, 118] with respect to general planning [76]. Essentially, the problems focus on either a probabilistic assignment of a task to a resource, or have a known assignment with only an estimated completion time, which potentially extends the total makespan of a solution.

The scheduling problem covers a significant portion of the multiagent task assignment problem domain, to include multiagent assignments with multitasking agents, scheduling uncertainty, and task ordering. However, constraints between resources are generally not addressed. To fill this shortfall, another type of research must be incorporated which addresses how resources interact, such as constraint satisfaction.

2.1.3 Distributed Constraint Satisfaction. The distributed constraint satisfaction problem (DCSP) [125] is a satisfiability problem where each entity of a group has boolean predicates indicating whether that member is present in a group solution based on a value associated to the entity. Essentially, the constraint satisfaction problem (CSP) involves careful selection of items in a group based on their value. The distributed form of the constraint satisfaction problem extends the CSP such that each group entity is associated to more than one value. The entity is then subject to meeting all of the constraints of each value associated to it.

The key feature taken from constraint satisfaction, in regard to this multiagent task assignment approach, is the representation of relational constraints. In the multiagent task assignment problem, there are a set of tasks to be performed and a set of agents that can perform such tasks. The CSP applies by introducing selection predicates for these sets that limit which tasks are concurrently performed and which agents collectively perform a task. DCSP addresses the selection of which components of an agent uses to perform a task when multiple tasks are concurrently assigned.

A variety of techniques have been applied in DCSP-based experiments involving multiagent problems. One features a synthesis of the DCSP and the POMDP [95], which introduces uncertainty into a constraint problem. Another approach supersedes

the DCSP by presenting the distributed constraint optimization problem (DCOP) [93] which adds iterative constraint optimization to the values of the DSCP problem while satisfying the relational constraint properties.

Constraint satisfaction problems abound in various areas of research. Classical problems include graph coloring and the eight-queens problem. Modern problems involve resource conflict mediation and efficiently manufacturing componentized products. Like scheduling problems, there are repositories of CSP problems, although some publications only offer summaries of research performed and not actual problem instance data. However, the generation of CSP data is defined in [123, 124] with an online repository at [78].

The chief drawback of the DCSP is also its greatest benefit, namely the association of relational constraints between entities of the same group. The central issue is that constructing a distributed solution involves decomposing the problem, but it is impossible to guarantee valid solutions if the subproblems do not involve all aspects of the relations being constrained. This tends to force DCSP solvers to have a more centralized solution approach [93]. Furthermore, DCSP solutions assign agents to tasks without regard for task order. Multiagent tasks are presumed to be solved in one step. Thus, the solution methods do not generally account for the sequencing of tasks, but do generate relationally constrained coalitions for tasks.

2.1.4 Dynamic Coalition Formation. Coalition formation [105, 107] is the cooperative effort of a collective of agents to perform an activity together. It is a highly visible and very important part of modern approaches to solving multiagent task assignment problems with a substantial surge of research in recent years. Essentially, a coalition is a subset of agents assigned to concurrently perform a task. The process of selecting the proper set of agents lies at the heart of coalition formation. Dynamic coalition formation [63] extends the definition of coalition formation by allowing changes in coalition structure during the duration of a problem.

Coalition formation and its dynamic extension are both fundamental axes in the MRTA taxonomy. With the prevalence of multiagent approaches to classic and current problem domains, coalition formation, in one form or another, has a strong influence the development of algorithms and heuristics for solving the task assignment problem. Any problem that is expressible as a time-extended set partitioning problem works as a problem for dynamic coalition formation. Unfortunately, there are few, if any, well-established repositories that provide source problems for testing and/or benchmarking of coalition-based algorithms.

However, there is a wealth of literature that demonstrate the use of coalition formation in several areas of research. For instance, forming coalitions for n-player games is examined in [4,28], and for marketplace economics in [80]. Constraints are imposed on how coalitions are formed, in the spirit of satisfiability, in [58]. Additionally, the use of coalitions in uncertain environments with Bayesian models is presented in [20], with reduced information/limited sensing in [71,91] and using POMDPs in [99].

One interesting aspect of dynamic coalition formation research is that no recent literature examines the notion that an agent can belong to multiple coalitions simultaneously. This is a basic requirement for an agent to be able to perform multiple tasks concurrently, where each overlapping task is performed by a unique coalition.

2.2 Solution Techniques For Multiagent Task Assignment

There are many published solution techniques covering a wide range of multiagent assignment domain problems. The predominant algorithms found in current research implement tree-based searches, particularly DFS, reinforcement learning (RL), coalition formation and evolutionary algorithms. However, novel approaches that apply to multiagent problems come from a variety of fields outside of the mainstream multiagent research community. This section highlights some of the recent solution techniques from the research areas listed in Section 2.1 used to solve multiagent task assignment problems.

2.2.1 Direct Search. Several researchers generate solutions to the multiagent task assignment problem by directly generating a solution, usually through a tree-based search technique such as best-first search or by set constructing methods such as set partitioning. Classically, the OAP, which represents the most basic of task allocation problems, has been solvable by the Hungarian method [73] since 1955 with a remarkable number of approaches developed since then.

Approaches to the DCOP problem are shown in [93] where a best-first search is performed with intelligent backtracking mechanisms on the graph coloring problem. In [95], the DCOP solution method is integrated with a POMDP representation to solve the distributed sensor net problem. A branch-and-bound technique is used in [91] to solve multiagent cooperative transport in an unknown, partially observable static environment. A theoretical model for the DCSP is presented in [92] using the asynchronous weak-commitment (AWC) [129] search method for solving the distributed sensor network problem.

However, many algorithms choose a variation of set partitioning to develop solutions to the problem. Greedy set covering and set partitioning methods are applied to the multiagent problem in [107] using the RETSINA robot system [113] for the block pushing problem. Likewise, a coalition structure generation algorithm for characteristic function games (CFG) [27, 28] provides anytime CFG algorithms using selection methods similar to set partitioning approaches to establish error bounds from optimal solutions for single-task coalitions.

Other algorithms directly solve the problem using alternate selection methods. Complexity bounds for task allocation with greedy selection and n-dimensional knapsack algorithms are used in [75] for general game-theoretic payoff problems. A direct hill-climbing approach, Asynchronous Partial Overlay [85] solves the DCSP as applied to the multi-object tracking problem. A series of greedy set-packing algorithms are presented in [81]. Hill climbing and k-means clustering are also applied to cooperative target observation in [84].

Various scheduling algorithms provide direct methods for solving the multiagent problem. Seminal papers in this area include [83], which provides several methods to solve the basic scheduling problem. The schedule generation scheme (SGS) [66, 77] solves RCPSP (see Section 2.1.2), and can do so optimally. Likewise, several constraint satisfaction approaches are described in [74].

Finally, a series of theoretic approaches are outlined in [105] to generate coalition stability methods for CFGs as applied to the distributed vehicle routing problem. Many of the approaches in this section are derived from these theoretical outlines.

2.2.2 *Policy-based Learning.* Substantial research in the machine learning community has produced a variety of policy-based approaches to multiagent task assignment, many based on RL [112]. A summary of current RL methods is presented in [126] for multiagent game theory. The authors previously show their work using seven variations of Q-learning [119] with Markov decision processes (MDPs) in the robot soccer [111] problem domain. More foundational work is described in [22] where Q-learning models are applied to multiagent games and in [5] which applies the models to the multiagent, multi-task pursuit domain to maximize reward payout for capturing a group of prey. More recently, RL is applied to the multi-robot transportation problem using task allocation vacancy chains (TAVCs) [115], a biologically-based resource distribution model where vacated worker positions are hierarchically filled, to predict group performance in a greedy producer-consumer service model. In [17], RL is applied to group utility functions to solve foraging tasks seen in computer games.

Other types of policy-based learning algorithms incorporate Bayesian models and MDPs to guide the selection of actions or coalition formation for the multiagent problem. A point-based value iteration approach in [99] uses a policy-based approach with a POMDP for robots that play tag. A Bayesian model, the Bayesian core, is introduced in [20] to reduce uncertainty in the value of a coalition of agents in probabilistic assignments. Some of the direct solution generation approaches in [95] are

16

translated into policy-based tree graphs that are used to solve POMDP problems. Otherwise, policy-driven planning approaches using MDPs in an uncertain environment are outlined in [117] for agents solving mazes with limited sensing ability.

   2.2.3 Evolutionary.   A variety of evolutionary algorithms are applied to the multiagent task assignment problem, particularly the genetic algorithm (GA) [6] and more recently ant colony optimization (ACO) [30]. The GA is often seen solving scheduling problems with a general treatment of such problems and solutions in [7]. Additional approaches, namely tabu search and hill climbing, are found in [32]. Constrained job shop scheduling with tabu search is also discussed in [54]. Tangentially related is a processor scheduling model in [109] that uses an acyclic directed graph (DAG) with genetic operations to evaluate the involvement-contention model for processors.

   A variation of the ACO meta-heuristic is tailored to solve the quadratic assignment problem (ACO-QAP) [86] with variations supporting concepts such as convergence speedup, multi-objective solutions, biasing the pheromones, and dynamic networks. ACO is also used to solve the resource constrained scheduling problem RCPSP in [87] for several problem instances from the PSPLIB [68, 69] repository. The ant-inspired approach is also used in cooperative block (in this case ball) pushing in [12] with variable pheromone placement techniques for guiding agents to tasks.

   Another emerging approach for solving the problem is particle swarm optimization (PSO) [61]. PSO is shown to be a viable approach for solving task assignment problems in [104]. Development of a hybrid-PSO using concepts from gentic algorithms is described in [127, 128]. The PSO complement to ACO-QAP is defined in [50] showing the ability of a swarm to tackle quadratic assignment problems from the QAP-LIB [18]. The approach is applied to UAV task modeling in [59]. Limited robot communications are examined in [101] using a PSO with genetic operators.

17

2.2.4 Behavioral. The most recently emerged field for solving multiagent task assignment problems is behavior-based, interactive approaches, which is argued as more significant than traditional algorithmic approaches [120]. Several behavioral models based on emotions are demonstrated by robot groups in interactive environments or their own social groups. One of the original robotic implementations is the fault-tolerant ALLIANCE [96] implementation which uses impatience, acquiescence and motivation behaviors to select tasks in a hazardous waste clean-up scenario. The B-CMOMMT behavioral model [70] is applied to problems where agents request help to perform surveillance in the CMOMMT problem.

Affection models [36] are used to control robot actions in two different problem groups: using shame [37] for autonomous robots navigating a minefield, and a waiter-refiller service environment [94] where robots interactively serve finger food at a public gathering. Three additional types of behavior mechanisms are presented in [49] to develop controllers for multi-robot collection tasks.

Other types of behavior-based techniques come from different disciplines altogether. Coalition stability metrics are derived from leaderless distributed flocking in [52] where agents can fail in the environment. Brownian motion and viscous fluid flow equations form the basis for micro-robot swarms to aggregate at target locations to perform specific tasks in [40]. Finally, a phenomenological model based on a summary of local observations in [79] determines dynamic task selections for agents in a multi-robot, multi-foraging problem.

2.2.5 Economic. Another popular method of solving the multiagent task assignment problem is through economic approaches, particularly auctioning (or negotiation). These solutions are inspired by economic processes that drive profit maximization business rules. Multiple algorithms in [80] center on coalition formation with combinatorial auction methods, or combinatorial coalition formation (CCF), to measure the effectiveness of buyer subcoalitions for an item as compared to the full coalition in an electronic marketplace. The request for proposal (RFP) problem in-

troduced in [71] establishes coalition formation heuristics that maximize payout for agent cooperatively performing subtasks with limited information. In [130], decomposed tasks are auctioned to available agents to perform area reconnaissance.

A number of economic auction methods are implemented for robot groups. The M+ [13, 14] implementation has distributed robots bidding for tasks in a hospital environment based on cooperative and opportunistic behavior values. The Broadcast for Local Eligibility (BLE) implementation [121] uses auction mechanisms for variations of the CMOMMT [97] problem using an eligibility score to request task assignments. Another approach along the lines of an economic system is the publish-subscribe system [46]. It is also implemented by for a robotic system, MURDOCH [43], where robots cooperatively build task allocation solutions using behavior-based auction methods.

2.3 Summary

Various elements of the multiagent task assignment problem have a rich history and are still active areas of research. These resources provide an initial starting point for developing a unified framework needed to represent and solve the broad range of domain problems associated with task assignment. Concepts embed in task allocation, project scheduling, constraint satisfaction, and coalition formation are key to developing a unified representation of multiagent task assignment problems. The task allocation formulation provides the basis for task assignment representation. Scheduling allows for representation of task ordering and problem uncertainty. Constraint satisfaction provides relational constraints among the tasks and/or agents. Then, coalition formation provides ways to quantify the assignment of a group of agents to a task.

Several novel approaches apply to directly solving different forms of the task assignment problem, but use representations that do not necessarily single-handedly apply to all types of problem domains. Fortunately, several algorithms outside of the mainstream multiagent research area provide a template for solving portions of

19

the multiagent task assignment problem. The majority of the applicable algorithms take a direct approach to solving the problem or apply techniques from reinforcement learning or evolutionary algorithms. Robotic-inspired implementations often generate solutions using behavior and emotion based value functions and auctioning methods. There has also been a number of successful distributed algorithms suggesting that a unified solver should also be distributable. Furthermore, studies on these problem classes have introduced and developed refined methods for estimating error bounds, incorporating constraints, measuring group dynamics, and incorporating uncertainty.

However, in most cases, this valuable research was applied with specialized algorithms that are not suited to solve the entire space of problems without some modification. A vast majority of the robotic experiments conducted using these algorithms used relatively simple problems, and often very few robot agents.

III. Modeling Multiagent Task Assignment Problems

Despite the volume of contributions available for solving task assignment problems, what remains lacking is a unifying representation of multiagent task assignment problems. Many multiagent scheduling and task allocation publications present a definition, but the definitions generally cover only one or two of several categories identified by various taxonomies available [31,35,44,75]. Furthermore, the taxonomies examine the different components of the problem but list algorithms that do not generally solve the entire span of problem types discussed. The taxonomies consistently identify assignments that require one or more agents to accomplish a task. Another key assignment aspect includes the number of tasks one agent can perform, which when combined with the number of agents allows for representing a range of one-to-one to many-on-many task assignment strategies.

The taxonomies also address other assignment problem considerations such as task order, communication features, heterogeneity/composition, constraints and observability. The constrained multiagent task scheduling (CMTS) problem descriptor, developed in this chapter, engages each of these topics with the exception of communication [31, 64]. This topic is addressed by the algorithmic components of the unified framework. Thus, this framework begins by exploiting the depth of research in several areas with the CMTS problem descriptor as a unifying representation for task assignment problems. This effectively allows any task scheduling or allocation algorithm to advantageously apply existing and forthcoming technologies to various multiagent task assignment problems.

3.1   Constrained Multiagent Task Scheduling (CMTS) Problems

The CMTS problem, formally stated in Figure 3.1, is expressed as a set of multi-capable agents and a set of tasks, each mapped to a set of common domain operations, with quantification functions. Every agent performs some subset of the domain operations through independent abilities, and each task requires a subset of domain operations for completion. Figure 3.2 illustrates an example mapping of

21

Assume a group of multi-capable agents, M, are to perform multi-agent tasks Y. Frequently, agents and tasks are defined with a common set of domain operations to ensure competent pairing; so, let D define a set of domain operations. Then, define for every agent a set of abilities derived from the domain operations, i.e., $\forall m \in M, C_m \subseteq D$ to base agent m's ability set C_m. Similarly, define for every task a set of requirements derived from the domain, i.e., $\forall y \in Y, R_y \subseteq D$ for task y's requirements R_y. Formulate constraint sets B_x, H_x and P_y such that the set $B_x \subset \bigcup_{m_i n M} C_m$ has every ability $x' \in \bigcup C_m$ that is bound to an ability x, $H_x \subset \bigcup_{m \in M} C_m$ identifies every ability $x' \in \bigcup_{m_i n M} C_m$ that inhibits agent ability x from performing a task, and $P_y \subset Y$ identifies every task that must completely precede a task y. Note: $x \in B_x$, $x \notin H_x$, $B_x \bigcap H_x = \emptyset$ and $y \notin P_y$.

Provided the constraints are satisfied, an assignment, (C_y, y), matches a coalition C_y such that $C_y \subseteq \bigcup_{m \in M} C_m$ to a task y to be performed during at given state s. Individual agent abilities are quantified by $v_x(y, s)$; tasks by $v_y(C_y, s)$. The assignment of a coalition C_y to any task is quantified by a function f^C over the agent ability functions, i.e., $f^C_{x \in C_y}(v_x(y, s))$. Thus, an assignment is quantified by the assignment utility function $u((C_y, y), s)$ as a function, f^A, of coalition and task associated value functions, $f^C(v_x)$ and v_y, as shown in Equation 3.1.

$$u((C_y, y), s) = f^A(f^C_{x \in C_y}(v_x(y, s)), v_y(C_y, s)) \tag{3.1}$$

A solution, S, is an acyclic directed graph of nodes (C_y, y) where the sequence of assignments are determined by paths in the graph. $U(S)$, Equation 3.2, quantifies the utility value of a solution S as a function F^U of the assignment utility values $u((C_y, y))$. Furthermore, the expected value of a solution, $E[U(S)]$, is evaluated by Equation 3.3 as the combined value of $U(S)$ and the expected value of assigning the entire agent set to the remaining tasks $\{y | y \notin S\}$.

$$U(S) \equiv U(S, s) = F^U_{(C_y, y) \in S} u((C_y, y), s) \tag{3.2}$$

$$E[U(S)] = F^U\left(U(S), f^A_{(X, y) \ni y \notin S} E[v_y(X, s)]\right) \tag{3.3}$$

Figure 3.1: Formal Model for the CMTS Problem.
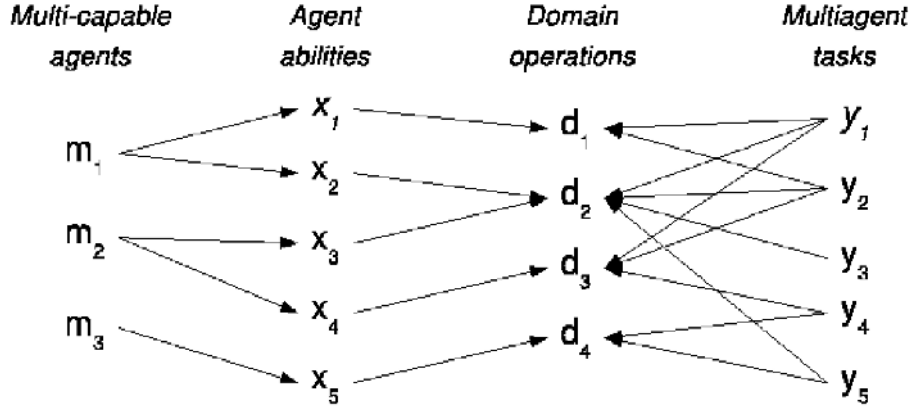
Figure 3.2: Example CMTS Problem.

agents to tasks through a set of domain operations. The solution to a CMTS problem involves coalitions of agent abilities assigned to perform every task in multi-sequential order.

The coalition of agent abilities forms by selecting a group from all of the abilities that meet any operation required by a task. For any task requirement, one or more abilities can perform the exact same operation, potentially from several different agents. Each ability has two types of relational constraints that determine membership in a coalition. The first constraint is inhibition. Inhibition prevents an ability from performing any operation of a task, and is caused by the presence of another ability in the coalition. For example, if it is not possible for an agent to push an item and lift it at the same time, the ability to push inhibits the ability to lift, and possibly vice-versa.

The other constraint is binding. Binding forces an ability to only perform the same task as the ability that bound it at any given moment. This constraint is most practical for physical problems where two abilities of a robot cannot perform spatially separated tasks such as pushing an item in one corner of a room while lifting an object in the center. In this case, the abilities bind each other so that they can only perform the same task. Note for the examples given that the combination of inhibition and binding forces a multiagent assignment to any task that requires pushing and lifting.

23

The task given to a coalition also has a relational constraint, prerequisites. Prerequisites are tasks that must precede a given task. This constraint enforces partial or full task ordering in the assignment sequences. For example, only solutions that enforce moving the furniture before cleaning the floor are acceptable, so moving is a prerequisite task to cleaning.

The value of a coalition ability or task in an assignment is quantifiable through ability and task value functions, synonymous to weighted utility values in task allocation or coalition formation. The task value function offers the value of performing a task given a coalition at some state in the problem domain. This value function is useful for, among many other things, defining the weighted priority of performing a task or the probability of successful completion in a stochastic environment. Complementary, the ability value function quantifies the value of an ability performing a given task for some problem state. Examples using this concept include robotic behaviors, coalition formation metrics, or cost functions. The value of a coalition of abilities is simply a function combining the ability value functions of each coalition ability, e.g., a summation or maximum value.

The value of an assignment is quantifiable based on the task value and coalition combination functions. An assignment combination function takes the coalition value and task value at a given state and produces a single result. One application comes from marketing where agents (abilities) bid their costs and the task yields a profit. The assignment combination function produces the difference to measure the value of assigning a coalition to some task.

The domain state used for deriving assignments and assignment utility values encapsulates domain specific information required to make the value calculations. For instance, one of the simplest domain states is time. An agent may be able to calculate its distance to an object if it knows how much time has elapsed, or a task might increase its prioritization if not promptly addressed. Alternately, a partially constructed solution containing the set of assignments already made and what needs

$$\{x_1, x_2, x_3, x_4\}; y_2$$

$$\{x_4, x_5\}; y_4 \qquad \{x_3\}; y_3$$

$$\{x_1, x_2, x_4\}; y_1 \qquad \{x_3, x_5\}; y_5$$
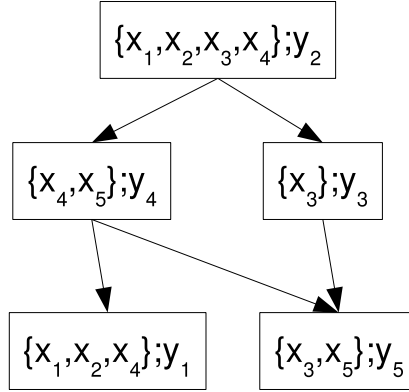
Figure 3.3: Solution Graph Representation.

to be accomplished can represent the domain state. For example, the effectiveness of a coalition can be measured by when and which abilities are available or by the types of tasks that have been assigned.

Ultimately, the utility value of a solution is realized by combining the assignment values in a meaningful way. This combining function represents the quantifiable portion of the objective for the problem. Example objective functions include maximizing profit or minimizing distance. The utility combination function for these examples might be a summation over the assignment values. Another complex approach is to sum the maximum profit of each agent ability. Algorithms seeking to find the best assignment solution for a problem attempt to optimize this utility value.

3.2 Solution Representation

The solution to a CMTS problem holds a multi-sequential set of assignment of coalitions to tasks. Tasks that are serially performed are often represented as a single sequence. However, to properly represent potentially overlapping assignments, the solution must use a branching structure. Therefore, the clear choice for representing a CMTS solution is a graph.

3.2.1 Directed Acyclic Graph Representation.   The solutions for CMTS problems in the framework are formed as a directed acyclic graph (DAG). The nodes in

the graph represent assignments of an agent ability coalition to a task. The edges in the graph, denoted $edge(a_i, a_j)$ for any two assignments, represent a state transition of agent abilities finishing one task to performing the next. A directed edge exists between two nodes, a parent and child, where the complete performance of the task in the parent node explicitly precedes the performance of the task in the child node. The ancestors (anc) of an assignment a include all assignments that precede a on every path containing a. Similarly, the descendants (des) of an assignment a are all the assignments that appear after a on every path containing a. A *path* is any connected, linear sequence of assignments in the graph.

The nodes in a solution graph have as many parent and child nodes as warranted, provided that there are no cycles. Semantically, an outbound branch represents the dissolution of a coalition where agents move on to different tasks. An inbound branch shows the sources of the coalition formed to solve a task. The edges indicate the scheduled order of performing the tasks. Two or more branches from a node means that the assignments in the child nodes are performed in any order, including concurrently. An agent ability assigned to tasks that are on separate branches is potentially multi-tasking by performing two unique tasks at the same time.

Acceptable DAG solutions can have multiple paths between two nodes, such as a node's parent (or any of its ancestors) directly referencing its children. Each task appears exactly once, and agent abilities appear as needed to solve the problem. For problems with recurring tasks, each instance of the task appears uniquely identified.

Figure 3.2 illustrates a graph representation of the components of a 3-agent, 5-task CMTS problem showing how agent abilities and tasks map to domain operations. A potential schedule graph solution appears in 3.3. The graph shows the pairing of a coalition of abilities to a task, forming the assignment nodes.

3.2.2 Validating Solutions. Five tests are performed to validate the correctness of a solution: coalition conflict, separated abilities, assignment cycles, prerequisite ordering, and unsatisfied requirements. In order to be a valid solution, each of

Table 3.1: CMTS Solution Constraint Checks.

Validation	Formal Constraint Definition
Ability inhibition	$H_x \bigcap C_y = \emptyset$
Ability binding	$\forall x' \in B_x, \exists path\left((C_y, y), (C_z, z)\right) \ni x \in C_y \wedge x' \in C_z \wedge y = z$
Assignment cycles	$\forall (C_y, y), (C_z, z) \in anc((C_y, y), \nexists edge\left((C_y, y), (C_z, z)\right)$
Prerequisite ordering	$\forall z \in P_y, \exists (C_z, z) \in anc\left((C_y, y)\right)$
Unsatisfied requirements	$\forall d \in R_y, \exists x \in C_y \ni x = d$

the five tests must be satisfied. The first test checks that no inhibiting agent ability is in the same coalition as a given ability. The second ensures that every bound ability is not working concurrently on a separate task which implies that it is separated from the binding ability. Any assignment that is not in the ancestry of descendency of the task an ability is assigned to has a potentially concurrently performed task. The third test is an assignment cycle check. The solution must preserve an acyclic structure so that no prerequisite assignment can become an ancestor and descendant to a constrained assignment. Fourth, for every task, each of its prerequisites must appear in ancestor assignments. This ensures that all prerequisite tasks are complete before a given task is started. The final check ensures that there exists at least one capable ability provided for every operational requirement of a task.

These checks ensure that solutions are valid and consistent. Any constructed solution that fails any one of the tests must be considered invalid. Table 3.1 lists the checks with descriptive and formal annotations. Note: an algorithmic approach to repairing invalid solutions is defined in Algorithm 19.

Table 3.2: Solution Matrix Representation.

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5
y_1	1	1	0	1	0	0	0	0	0	0
y_2	1	1	1	1	0	0	0	1	1	0
y_3	0	0	1	0	0	0	0	0	0	1
y_4	0	0	0	1	1	1	0	0	0	1
y_5	0	0	1	0	1	0	0	0	0	0

27

3.2.3   Matrix-Equivalent Representation.   Alternately, a CMTS solution DAG is compactly represented by a two-part binary matrix. The graph representation holds two sets of facts about a solution: the allocation of agent abilities to a task and the schedule of task execution. Then, the first part of a matrix representation, the allocation submatrix, shows coalition membership in an $|X| \times |Y|$ binary matrix where $|Y|$ is the number of tasks in Y, and $|X|$ is the total number of agent abilities in $\bigcup C_m$. The submatrix contains a value of one for every agent ability that is assigned to perform the row-indexed task. The second part, the sequencing submatrix, indicates task order in an $|Y| \times |Y|$ binary matrix, adjacent to the first part. This submatrix contains a value of one where a column-indexed task is an immediate child to the row-indexed task. All other values are zero. Note that a value of one in a column of the sequencing submatrix indicates which row-indexed tasks are parent tasks to the column-indexed task. The matrix form for the graph solution in Figure 3.3 appears in Table 3.2.

The matrix representation of the DAG has numerous advantages. First, the use of structured binary values has proven more useful for manipulating the solution structures in some of the algorithms used to solve the problem, especially the genetic algorithm. Also, the matrix can be stored, compressed, and transmitted using very few resources, particularly computer memory. Finally, the binary values allow for the application of mathematical operators, e.g., AND, bit shifting, Hamming distances, or non-overflow binary addition. The operations help visualize the solution space of the task assignment problem as a discrete, binary field. This reveals the structure of the solution space and supports basic solution space analysis.

3.3   Solution Space Analysis

Some mathematical analysis is proposed for the defined unified model and solution structures. Specifically, the size of the problem is defined for the homogeneous, unconstrained problem with additional modeling for sizing the problem with hetero-

geneous agents under constraint. The impact of constraints on the solution space is also analyzed.

3.3.1 *Size and Complexity Models.* A number of analytical expressions characterize the CMTS problem, solutions, and solution space. Two key metrics that are useful for comparing problems and qualifying the effectiveness of algorithmic approaches are problem domain size and an introduced problem complexity statistic.

The simplest task assignment problem only describes domains where each agent performs at most one task, and each task only requires any one agent. These problems are very simple one-to-one relationships, readily solved by any form of the optimal assignment algorithm which is an $O(n^3)$ algorithm. CMTS problems with one-to-one relationships between m agents and y tasks are solvable in $O((my)^3)$ time. By increasing the complexity of the problem to allow for agents performing more than one task at a time, and equivalently, tasks requiring more than one agent, the solution space increases exponentially due to the selection of multiple agent abilities per task or vice-versa for a one-to-many relationship. Variations on solving this problem have the complexity of solving the set partitioning problem, which is NP-Hard. However, when applying scheduling methods to solve a sequence of task allocations, the solution space size becomes exponential, making the problem NP-Complete [89].

The solution to a CMTS problem represents a series of coalition mappings to tasks and a sequencing of tasks. First, given $|X|$ independent abilities, there are a maximum of $2^{|X|} - 1$ coalitions that can perform a task. This assumes that every agent contains an ability for each domain operation, no ability is inhibited, and the task requires every operation. Clearly, if an agent does not perform every domain operation, then there are fewer coalitions. Each coalition is formed by selecting at least one ability for each task requirement. The coalition may also include abilities that do not perform the task but are still present. Specifically, let c_d be the set of all agent abilities that perform domain operation $d \in D$, i.e., $c_d = \{x \in \bigcup C_m | x = d, \forall m \in M\}$. Then the number of possible coalitions available to perform a task y is expressed in

29

Equation 3.4 as the independent selection of a subset of abilities that perform each required operation of a task and a selection of abilities that do not perform any requirement of the task.

$$|C| = \Pi_{y \in Y} \left[\Pi_{d \in R_y} \left(2^{|c_d|} - 1 \right) \cdot \Pi_{\overline{d} \ni R_y} 2^{|c_{\overline{d}}|} \right] \tag{3.4}$$

$$|C|_{x_j \in H_{x_i}} = \Pi_{d \in R_y \setminus \{x_i, x_j\}} \left(2^{|c_d|} - 1 \right) \cdot \begin{cases} 2^{|c_{x_i}|-2} & \text{if } x_i = x_j = d, \\ 2^{|c_{x_i}|-1} \cdot 2^{|c_{x_j}|-1} & \text{otherwise} \end{cases} \tag{3.5}$$

However, inhibition lowers the number of valid coalitions, which results in a very complex equation. For instance, Equation 3.5 provides the precise number of coalitions available to perform the task when an inhibition exists between just two abilities. The true total number of coalitions is then the universal size, $|C|$ minus all non-duplicate sets containing inhibitor conflicts. Over $|Y|$ tasks, the total number of coalition sets is approximated by $|C|^{|Y|}$.

Task sequencing, on the other hand, is the selection of tasks that are performed sequentially. For every task y, at most $|Y| - 1$ tasks can be subsequently performed. Thus, there are $2^{|Y|-1}$ possible selections, including the case where no task is performed afterwards. However, task prerequisites reduce the size of the selections to $\Pi_{y \in Y} 2^{|Y \setminus P_y|-1}$. Furthermore, additional selections are eliminated based on the formation of cycles in the solution. As with counting the total number of coalitions that are free from inhibitor conflicts, the expression for the number of cycle-free solutions with prerequisite constraints is complex, requiring knowledge of the task sequencing for all tasks. If S_y represents the number of valid sequences for a task, then there are $\Pi_{y \in Y} 2^{S_y}$ total complete task sequences for a problem solution.

Thus, the solution space has an maximum upper bound shown in Equation 3.6 for the completely unconstrained version. A tighter bound can be represented by using the constrained counts of coalitions and task sequences where available to calculate the lower upper bound (LUB) on the solution space size, shown in Equation

3.7.

$$UB = \left[2^{|X|} - 1\right]^{|Y|} \cdot \left[2^{|Y|-1}\right]^{|Y|} \sim O\left(2^{|Y|(|X|+|Y|)}\right) \tag{3.6}$$

$$LUB = \Pi_{y \in Y} \left[\Pi_{d \in R_y} \left(2^{|c_d|} - 1\right) \cdot \Pi_{\overline{d} \ni R_y} 2^{|c_{\overline{d}}|} \cdot \Pi_{y \in Y} 2^{|Y \setminus P_y| - 1}\right] \tag{3.7}$$

The UB shows that the size of the solution space for a CMTS problem is sufficiently large enough to solve problems with $O(n!)$ solutions where there is a fixed number of agent abilities. Without proof, the UB is much greater than the Sterling series approximation to $n!$—more specifically, $2^{|Y|(|X|+|Y|)} \gg |X| \cdot (|Y|!)$ (parenthesis added for clarity) for $|X|, |Y| \geq 1$ provided $|X|$ is not grossly larger than $|Y|$. If the number of agent abilities grows exponentially, then space built on the defined solution structure is quickly deficient since $(|X||Y|)!$ grows exceptionally faster than UB as $|X|$ and $|Y|$ increase. However, for unconstrained problems, the LUB is best approximated by the UB.

Additionally, the LUB is used in computing a statistical constraint metric. The constraint metric basically indicates how constrained a problem is by counting the number of constraints instead of the number of coalitions of sequences that have constraints. For coalition sets, there are exactly $\sum_{x \in X} |H_x|$ unique constraints out of $\binom{|X|}{2} = |X|(|X| - 1)$ possible constrained relationships. Similarly, there are $\sum_{y \in Y} |P_y|$ unique constraints out of $\binom{|Y|}{2} = |Y|(|Y| - 1)$ possible task sequencing assignments. Combined, these counts provide the base problem complexity shown in Equation 3.8, which is bound between 0 and 1.

$$\chi_C = \frac{\sum_{x \in X} |H_x| + \sum_{y \in Y} |P_y|}{|X|(|X| - 1) + |Y|(|Y| - 1)} \tag{3.8}$$

This metric captures the problem constraint, but does not take into account the number of unconstrained tasks and agents. To derive the complete constraint statistic, χ, the metric χ_C is treated as one variable in a two-variable distribution, where the other variable counts the number of unconstrained tasks and agents. That value, χ_E is represented in Equation 3.9 as the total number of entities with a constraint over

31
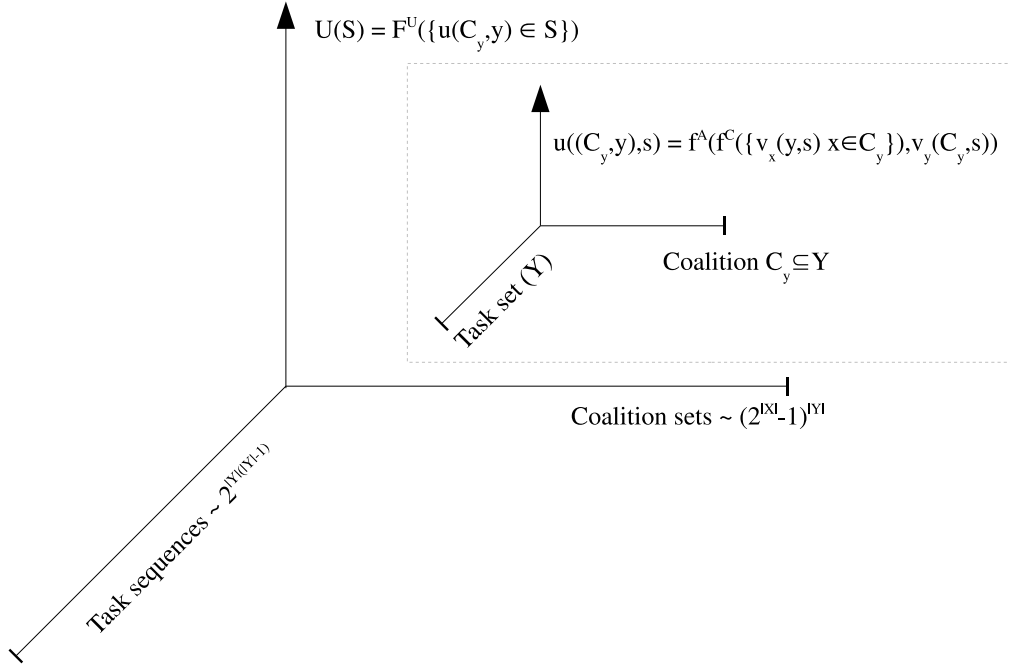
Figure 3.4: Solution Space Visualization on 3-Axis Coordinate System.

the total number of entities.

$$\chi_E = \frac{|\{x \in X \ni H_x \neq \emptyset\}| + |\{y \in Y \ni P_y \neq \emptyset\}|}{|X| + |Y|} \tag{3.9}$$

Then, the final constraint metric, χ, is $(\chi_C)^{\chi_E}$ as expressed in Equation 3.10.

$$\chi = (\chi_C)^{\chi_E} = \left(\frac{\sum_{x \in X} |H_x| + \sum_{y \in Y} |P_y|}{|X|(|X|-1) + |Y|(|Y|-1)} \right)^{\frac{|\{x \in X \ni H_x \neq \emptyset\}| + |\{y \in Y \ni P_y \neq \emptyset\}|}{|X| + |Y|}} \tag{3.10}$$

3.3.2 Landscape. Since each coalition set and task sequencing is enumerable (based on the binary matrix representation), one way to visualize the solution value space, the utility value function $U(S)$, is to form a discrete three-axis system. Two axes form the solution space with one axis enumerating the coalition set, and the other enumerating the sequences. The third axis holds the value of the solution formed by the other two axes. Figure 3.4 illustrates this visualization.

32
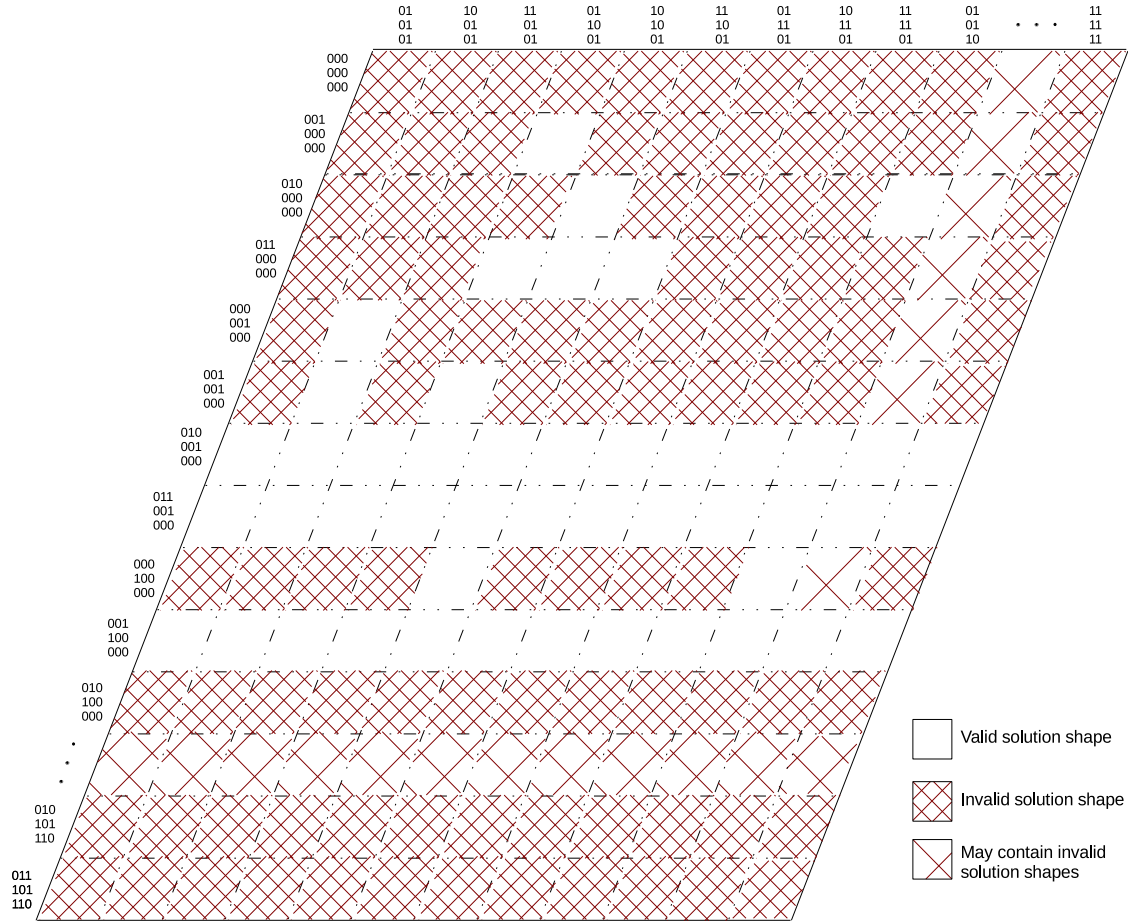
Figure 3.5: Sample Solution Space for a 2x3 Problem.

Constraints in the problem reduce the size of the solution space in various ways. An inhibition constraint reduces the number of coalitions that can form for a task. Task prerequisites eliminate certain sequences. These constraint-based reductions are represented in the three-axis visualization as point discontinuities in the value plane. For large enough or significant constraints, the solution space will contain sizeable holes which are seen in the value space as larger continuity jumps. Figure 3.5 illustrates a 2x3 problem with several types of constraints affecting the solution space. The horizontal axis represents each of the possible coalitions of two abilities for three tasks; the opposing axis shows all possible sequences for the tasks. From the
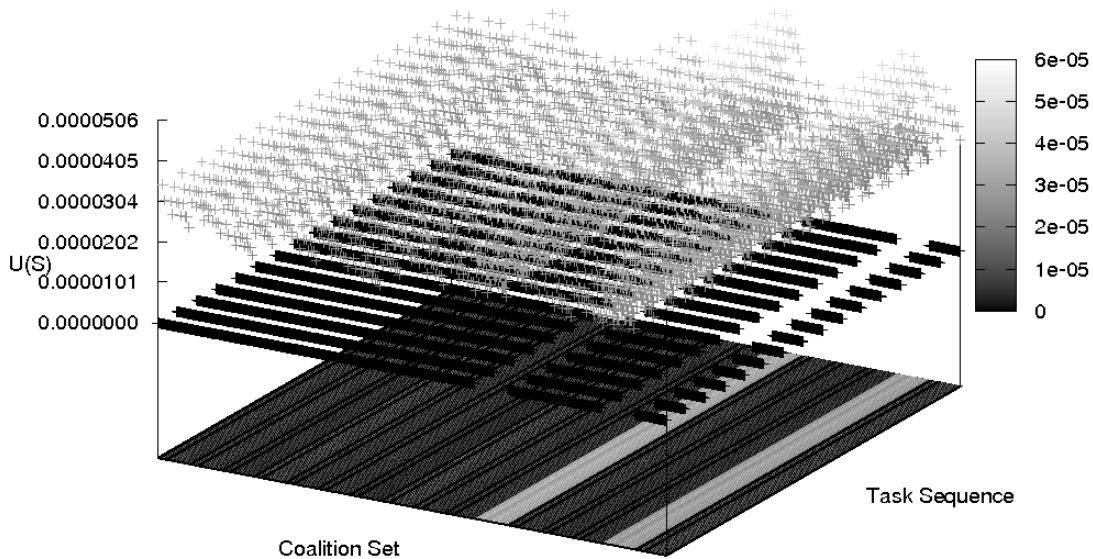
Figure 3.6: Utility Surface for 2x3 Problem on 3-Axis Coordinate System.

figure, every 2x3 problem, before relational constraints, already contains a significant number of invalid solution shapes.

The shape of the surface of the value plane depends on the functions and values used to compute the solutions utility value $U(S)$, namely the assignment utility values $u((C_y, y))$, and the enumeration of the coalition sets and sequences on the axes. When the changes in utility value are relatively small, the surface appears relatively smooth with some number of peaks and valleys regardless of the enumeration scheme of the solution space. However, with discontinuities, an enumeration causes various shaped "holes" in the surface. Value functions that change significantly can cause spiky surface features that challenge gradient-based search methods. Various enumeration schemes on dynamically ranged value functions can produce very different landscapes, most of which are likely to contain spiky surfaces. Value functions that are equal-valued for all but a few solutions instead produce landscapes with several

plateaus. Constraints in these problems can create discontinuities that cross plateaus. Ultimately, the shape of the landscape is defined by the structure and value functions of the CMTS problem itself.

An example of the surface plane for a relatively small 2 agent, 3 task problem is plotted in Figure 3.6 showing the full solution space sketched in Figure 3.5. The problem used to generate the figure has 5,184 possible coalitions and 16 task sequences. The bottom layer of the image shows a gradient contour map of utility values with lighter colors representing better utility values. The center layer shows where invalid solution shapes occur—each of these solutions is invalid by one of the constraints listed in Table 3.1. The upper layer plots the utility values $U(S)$ for valid solutions. This surface layer shows how spiky a typical solution space appears. The brighter spots on the bottom contour shows where are the better solutions.

3.4   Representing Classical and Modern Problems

Problem instances found in each of the contributing research areas used to form the CMTS problem descriptor are readily expressed using the CMTS formal notation. In this section, classical problem classes are rephrased using the new descriptor to show its wide range of expressiveness. Specifically, the fundamental problems of the job shop, the traveling salesman, vehicle routing, the flow shop, and multi-object tracking are expressed. However, to illustrate the full prowess of the descriptor, a new problem class based on search and recovery is constructed that intentionally involves every aspect of the CMTS problem since none of the classical problems are intended to do so.

3.4.1   Optimal Assignment Problem.   One of the most classical task assignment problems is the optimal assignment problem (OAP) [39]. In this problem, a group of jobs are assigned to a group of workers. In most instances, each worker has a quantitative performance rating associated with performing a job. The objective is to identify the set of assignments that maximizes overall performance.

This is the basic construct for task allocation. The problem as stated does not contain any of the constraints expressible in CMTS, however, the OAP can be presented with job prioritization. The problem does imply a one-to-one assignment process; i.e., each agent performs exactly one task, and each task requires at most one agent. Also, any agent can perform any task, and vice-versa, implying the system is homogeneous.

The CMTS presentation of the OAP is very straightforward. Exactly one domain operation is all that is required, and every agent performs the one operation. Likewise, each task requires that one operation. Every coalition assigned to a task consists of one agent ability. The agent value function, v_x, equals the worker's performance rating, and the task value function is identically zero. The assignment utility function, f^A, coalition combination function, f^C, and utility combination function, F^U, are each a summation. The objective is to maximize $U(S)$, which is simply $U(S) = \sum_{(C_y, y) \in S} v_x(x \in C_y, y)$. (Note: a state is not required to compute this utility.)

3.4.2 Flow Shop and Job Shop Scheduling. The flow shop scheduling problem (FSP) [7] is another classical approach that models task assignment. In a flow shop, a group of jobs is processed through a fixed series of machines. Each machine processes one operation of each job then passes the job to the next machine for continued processing. The goal is to arrange the jobs to minimize the time to process the entire group sequentially. In classical definitions, each machine only processes one part of a job at a time and machines can be idle while another machine is processing a job.

Representation by CMTS makes a few assumptions: each job is associated with a fixed processing time per machine which includes between-machine transfer times, jobs must be processed by every machine, and (for the classical version) there is no precedence among jobs. Then, the description uses a set of domain operations equivalent to the set of jobs where each job is represented by an independent job

task for every one of the operations. Thus, for n operations, every job has n tasks to be scheduled. Furthermore, since the machine operations are ordered and fixed, each job task has prerequisites such that job task $m_{i,j}$ is a prerequisite of $m_{i,k}$ iff $j < k$ with machine j performing the job before machine k. Finally, each machine performs exactly one operation. This construct ensures that no two job operations are performed on one machine. Furthermore, each machine inhibits every other machine. This constrains the problem such that only one machine can process a job at any given time.

A state is required to quantify the time required to process all jobs. For the assignment utility and value functions, the state is the partial solution of all assignments that have been made. Then, the job (task) value function provides the total processing time required to perform and transfer the job to the machine plus the total time spent on parent assignments, as derived from the state. The machine (agent) value function is identically zero. Since each machine has only one operation, the coalition combination function is the identity function ($f^C(v_x) = v_x$). And, since the agent value function v_x is non-contributing, the assignment utility value function provides only the value of the task value function, i.e., $f^A \equiv\equiv v_y(C_y, s)$. The solution utility function is the maximum of all of the assignment utility values, $F^U \equiv max$.

Thus, the solution utility value is $U(S) = max\{(v_y(C_y \in S, S)\}$. Various solutions are compared by $U(S)$ with the solution having the smallest utility value chosen as the schedule that minimizes the total amount of time to process the jobs. For a different metric, such as job latency, the value functions may require redefinition.

The typical graph solution structure for a valid solution resembles a diamond shape where only nodes containing the first assignment of a machine branch into two, one to continue processing the job on another machine and one to show a new job operation being assigned to the current machine. Other graph structures violate problem domain constraints or graph consistency rules (Section 3.2.1).

The job shop scheduling problem (JSP) [7] is similar to flow shop with one important distinction. Job shop scheduling involves processing a series of jobs by several machines without regard to a fixed chain of operations. In other words, a job can take a different route through a series of machines than another job. Classically, the job shop problem poses that each machine is unique, each machine performs only one operation of a job at a time, no two operations of a job are processed on the same machine, and machines can be idle while others are processing. The typical goal is to find the schedule that minimizes the total amount of time required to process the jobs.

Representation by CMTS keeps all of the assumptions of the FSP with the exception of the prerequisites on the tasks. Likewise, a state is required to evaluate a solutions utility value and the functions used in FSP hold for JSP provided the goal is to minimize the total time. Typical solution structures emerge between one long sequential processing (clearly non-optimal), or n distinct chains, where each assignment is uniquely one of n machines.

3.4.3 Multiple Traveling Salesman and Vehicle Routing Problem. The multiple traveling salesman problem (MTSP) [90, 116] and vehicle routing problem (VRP) [21, 116] are well defined problems that are readily converted into CMTS representation. Each problem consists of a number of salesmen or vehicles, n, that must visit each of m cities exactly once. Furthermore, the MTSP requires salesmen to return to their initial city. Any number of entities in a common city can travel together to another. In either problem, costs are associated with traveling between cities, and possibly differential costs in which salesman/vehicle visits a city. The objective, usually represented as a summation over the costs, is to generate a route that minimizes the overall cost of servicing the cities. Instances of both the MTSP and VRP are found in the OR-Library [9] with solutions to benchmark problems.

This problem exhibits characteristics where all agents, represented as salesman or vehicles, perform exactly one task (i.e., visit one city at a time) and each task,

represented as travel to a city, requires multiple agents. In CMTS, only one operation is performed, namely travel. Every agent has only one ability, and every task only has the one requirement. The assignment of several agents in a coalition to one task naturally fills the requirement to have multiple salesmen or vehicles travel to a destination city.

The agent value function, v_x, varies depending on the model of the problem, e.g. a cost associated to using an agent, but can suitably be set to zero since there is generally no associated cost on the agent's part. The task value function, v_y, equals the associated cost of performing the task. Common practice uses a distance function. The assignment utility function f^A and coalition combination function f^C are both summations to total the distance traveled by each agent in the problem. The solution utility combination function F^U is also summation.

3.4.4 Multi-object Tracking. Another benchmark problem domain is cooperative multi-robot observation of multiple moving targets (CMOMMT) [97]. Additional work on tracking and/or surveillance of targets appears in [70, 72, 84]. The CMOMMT problem involves a team of agents with range-limited, omni-directional sensing to track a set of targets within some defined region. The objective is to maximize the number of targets being observed in the region over a given time period with agents adjusting their positions to compensate for areas that are not being sensed.

The CMOMMT problem assumes that the targets and their paths are not known ahead of time which forces a dynamic solution to this problem. The CMTS descriptor is limited to solving static instances of a problem, so the CMOMMT problem assumptions are relaxed so that all targets are known ahead of time and that their paths are predictable once detected. Otherwise, the problem must be solved as a series of k-target instances, where up to k targets are detected or observed.

The problem is effectively a many-to-one mapping with several agents observing a target. However, only one agent is ever needed to observe the target, and that is the only ability of the agent. Thus, in CMTS, this problem only has one domain

operation, the observance of a target. Every agent has as many abilities needed to observe all of the targets such that $|C_m| = |Y|$. A task is created for each target with only one requirement, i.e., to be observed. States include a period of time.

The agent ability value function $v_x(y, s)$ reports one iff the ability is actively tracking the target associated with task y, otherwise zero. The coalition utility function f^C reports the OR of the ability functions, so that if any ability is tracking the target, then the coalition is tracking the target. The task value function v_y does not contribute, so it can be any appropriate identity function. The assignment utility function f^A then only reports a value of one or zero depending on whether or not the coalition is tracking its target. The solution utility function F^U is then the summation of the assignment utilities. This value indicates how may targets have a coalition actively tracking it. The objective is to maximize $U(S)$. Agents are free to move within the domain independent of the solution mechanism in order to maximize $U(S)$.

3.4.5 Autonomous Search And Recovery. Problem instances generally abound for specialized task assignment domains, particularly for OAP, MTSP, and VRP. However, some publications describe, but do not provide data for, many problems used in experiments, especially CMOMMT. Unfortunately, even if the data were available, these problems are not complex enough to fully utilize the expressive power of the CMTS descriptor in the unified framework.

To fully test the framework descriptor, a combined problem domain is introduced based on search and recovery (SAR) operations [62]. The autonomous search and recover (ASAR) problem is a combined application of vehicle routing (for route planning and transporting objects), cooperative multi-target tracking (for searching and tracking), and job scheduling (for prioritizing tasks over time) as developed from the translated MTSP, VRP, JSP and CMOMMT problems in previous sections.

The general concept behind the ASAR problem is that a group of multi-capable agents are to recover some number of items from an uncharted area. Agents must thoroughly search the area observing various items that must be moved to a recovery

area or otherwise dealt with. Discovery of new locations or items in a location warrant a task to either secure a location or to recover items from a location.

Thus, the problem domain contains two distinct tasks. The first task is to secure a location. Securing a location involves monitoring all objects and pathways in a bounded area and disabling any items as required. The second task is to recover items. To recover an item, it must be monitored and transported to a recovery area. As a prerequisite, any recovery operation in a location first needs to be secured.

In order to accomplish the tasks, the problem uses four domain operations: monitor, observe, transport and disable. The monitor operation requires an entity to observe all events within a sensory range of some fixed location. The observe operation requires continuous observation of a moving entity. The transport operation requires the physical movement of an item from one location to another. The disable operation causes an item to be neutralized, eliminated or otherwise arrested, e.g., a threat is eliminated. The transport and disable operations are mutually exclusive—an agent that possesses both of these abilities can only use one at a time. Similarly, the monitor and escort operations are mutually exclusive. Otherwise, all of the abilities of an agent are bound to each other to restrict separate components of one agent to work on two tasks that are not co-located.

The secure task is mapped to monitor and disable, and the recover task is mapped to observe, monitor and transport. Agents have any combination of operations according to the problem instance. The overall objective is to minimize the effort required to complete the recovery. This objective can be met by minimizing the total distance to move items to the recovery area. Additional details are provided in Section 6.3.1 for experimentation and simulation.

The general complexity of the problem is bounded below $\chi = 1$ since only two of the four types of abilities cause inhibition. A maximally inhibited problem accounts for a minimum complexity of $\chi = 0.5$. Also, since each recover task has only one prerequisite and there must be at least one secure task, the maximum number of

prerequisites in a problem with $|Y|$ agents is $|Y| - 1$. The constrain complexity for tasks is then $(|Y| - 1)/2^{|Y|} \to 0$ as $|Y| \to \infty$. This function is bounded above by $1/log(1/x + 1)$ and has a maximal value of $\frac{1}{4}$ at $|Y| = 2, 3$. So maximum prerequisites account for a minimum complexity of $\chi = 0.25$. This implies that the maximally constrained ASAR problem has a maximum complexity of $\chi = 0.75$. On the other hand, one unconstrained disabling agent for one secure task is a valid ASAR problem and has complexity $\chi = 0$, which shows that the lower complexity bound is zero.

3.5 Summary

The CMTS problem description represents a unified set of multiagent task assignment problems by combining key concepts of task allocation, project scheduling, and constraint satisfaction with elements of dynamic coalition formation. Solutions to the problem take the form of a directed acyclic graph, or alternatively, a binary matrix based on the parent-child relationships of the graph representation. The representation provides quantification for measuring the utility of an assignment of a coalition to a task by $u((C_y, y))$ in Equation 3.1, and for the overall utility of the solution $U(S)$ by Equation 3.2.

An analysis of the solution space shows a maximum problem size bound of $O\left((2^{|Y|(|X|+|Y|)})\right)$. Constraints placed on the problem significantly reduce the space complexity, but create odd discontinuous spaces that are problematic for solution-space algorithms to navigate.

The expressiveness of the CMTS model is demonstrated by translating several multiagent task assignment problems into CMTS form. The combined set of multiagent task assignment problems is vast but fails to adequately capture the full potential of the unified problem descriptor. Thus, a new problem class suitable for completely testing the framework, the autonomous search and recover problem, ASAR, is introduced.

IV. Solving Constrained Multiagent Task Scheduling Problems

The second part of the unified framework contributes a series of algorithms designed specifically to solve the constrained multiagent task scheduling (CMTS) problem. Although there has been success in implementing algorithms that solve various forms of the task assignment problem, as noted in Chapter II, the CMTS problem introduces complexities that are generally not addressed by other algorithmic approaches.

Algorithms solving the CMTS problem must produce both the coalition for a task as well as the ordering of tasks in order to generate a solution. Traditionally, this has been done separately as either a coalition formation problem approach [20, 27, 28, 71, 80, 95, 99, 105, 130] or a scheduling problem approach [66, 74, 77, 83]. The algorithms presented in this chapter build upon previously successful models, but are modified to concurrently solve both aspects of the CMTS problem in a single process. Particularly, they ensure support for solving any multiplicity of agents-to-task and tasks-to-agent identified in task assignment taxonomies.

4.1 Algorithm Domain Mapping of CMTS Problems

The CMTS problem domain descriptor identifies four key elements in a problem: the agents set needed for coalition formation, the task set identified by the problem, relational constraints between agent abilities and/or tasks, and value functions. The solution space is also well-defined by an acyclic directed graph (DAG) which contains nodes of assignments (tuples) representing agent ability coalitions performing a task and edges indicating the asynchronous, parallel performance of tasks.

The algorithm domain design for CMTS problem solving algorithms work by taking an instance of the problem domain as an input and producing, as an output, one or more solutions in the solution space. The operational input is the set of agents and associated capabilities (M, X), tasks (Y), relational constraints (H_x, B_x, R_y, P_y), and value and combination functions $(v_x, v_y, f^C, f^A, F^U)$ from the problem. The output operation produces solutions $\{S\}$ as DAGs based on algorithm-specific operations:
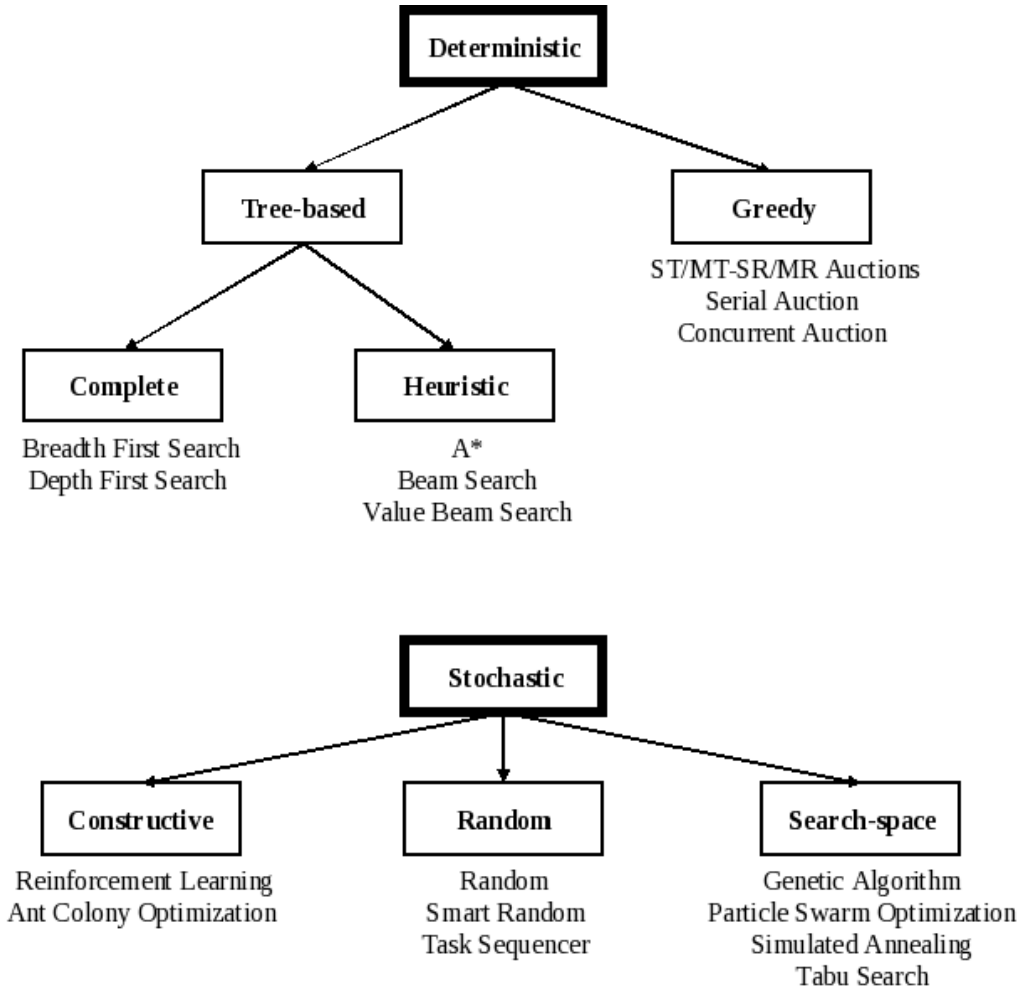
Figure 4.1: Select CMTS-Enabled Algorithms.

next state generation, selection, feasibility, solution and objective. The objective operation, however, remains consistent among all algorithms as the optimization of the utility function $U(S)$ based on the composition of value functions.

The remaining operations are discussed in more detail relative to the techniques used by the various algorithms modified to produce solutions to CMTS problems in this chapter. Both deterministic and stochastic algorithms, outlined in Figure 4.1, are modified to demonstrate how to apply the general concepts of a broad range of solution techniques to this problem class. Deterministic algorithms are further broken down into three categories: tree-based complete searches, tree-based heuris-

tic searches and greedy auction methods. Stochastic algorithms techniques include iterative construction, Monte Carlo techniques, and solution space searches.

4.2 Tree-based Complete Search Algorithms

The tree-based, complete search algorithms presented in the framework generate solutions by combinatorically constructing all solutions for a problem. Starting with an empty solution at the head of a tree, they iteratively grow branches by inserting one new assignment to the partial solution held by the parent node. The leaves of the tree hold the solutions, and the best solutions are selected based on optimizing the utility value function for the problem domain. This approach is guaranteed to find an optimal solution since all solutions are exposed, however, finding this solution is computationally infeasible for sufficiently large problems since any instance of the task allocation problem can be shown to be at least NP-Hard [75, 79].

Algorithm classes that fall into this category are based on non-heuristically driven breadth-first search (BFS) and depth-first search (DFS). This section presents the basic approach for using either algorithm to solve a CMTS problem.

4.2.1 Breadth-First Search. The breadth-first search (BFS) algorithm generates solutions by expanding each node of a layer of a search tree before expanding child nodes. This method opts to generate all leaf nodes last while fully expanding each branch. For task assignment problems, each depth in the search graph (most often a binary tree) up to the leaves represents a partial solution. The leaves of the tree represent completed assignment sets and constitute a solution.

The BFS algorithm is modified to use the CMTS definition with a graph-based solution. The technical implementation of the CMTS-ready BFS algorithm, shown in Algorithm 1, uses a queue initialized with nodes from the initial layer to perform the breadth search. The initial layer contains a node for every task in the problem and every possible coalition that could be assigned to the task. Remaining tasks are also tracked at each node.

Algorithm 1 Breadth-First Search

Require: CMTS problem (M, Y)

Ensure: A solution S^* that is optimal for (M, Y)

1: Let Q be a solution list
2: Initialize Q with an empty solution
3: Let S^+ be a solution queue initialized with an empty solution

4: **while** Q is not empty **do**
5:      Let S be the first element of Q, and $Q \leftarrow Q \setminus \{S\}$
6:      Let $\overline{Y}$ be the set of tasks y that are not assigned in S
7:      **if** $\overline{Y}$ is empty **then**
8:          **if** $U(S)$ is better than the utility of the best solution in S^+ **then**
9:              Let $S^+ = \{S\}$
10:         **else if** $U(S)$ equals the utility of the best solution in S^+ **then**
11:             $S^+ \leftarrow S^+ \cup S$
12:         **end if**
13:      **else**
14:         **for all** tasks y in $\overline{Y}$ **do**
15:             Let $P = $ all non-empty subsets of $\{y | y$ is assigned in $S\}$
16:             **for all** coalitions C_y assignable to y **do**
17:                 Let $S' = S \oplus (C_y, y)$ such that y is a child of all tasks in P
18:                 **if** isValid(S') **then**
19:                     Add S' to the end of list Q
20:                 **end if**
21:             **end for**
22:         **end for**
23:      **end if**
24: **end while**
25: Let S^* be any member of S^+
26: **return** S^+

The node on the front of the queue is dequeued and provides the partial solution needed to generate extended solutions. When expanding a partial solution, a series of new child nodes are created for new partial schedule solutions, each containing one additional assignment of a remaining task. The series includes all possible coalitions for the task inserted in the partial solution as a child node to every legal subset of existing assignments. For example, for a node with three tasks assigned and two tasks remaining, there are two sets of new child nodes, one for each remaining task. Each set of child nodes includes the insertion of the new node in one of seven child

positions: a child of the first task, of the second task, of the third task, of the first and second task, etc. Then, for each of those extensions, there is a node that represents every possible coalition that can perform that task in the assigned sequence. Thus, for a problem with three identical agent abilities, there would be seven child nodes for each of the seven extended positions totaling 49 new nodes for each one of the remaining tasks.

If a node is found to have an invalid partial solution due to a constraint enforcement from Table 3.1 during the expansion, it is not used in the search. This primarily occurs when coalitions contain abilities that have binding or inhibition violations are inserted into the partial solution, or the remaining tasks of an assignment contain a prerequisite task. Otherwise, each of the new search tree's child nodes are inserted into the tree and track which tasks are left by removing the newly inserted task from the parent node's remaining task list. The new nodes are then queued to the back.

When no remaining tasks are identified for a queued entry, the node holds a complete solution that is evaluated for a utility value (Equation 3.2). If the utility value of that current solution is better than the utility value of the best seen solution, then the set of best solutions is reset to hold only the current solution. If the utility values are equal, then the current solution is added to the set of best solutions. All possible solutions have been examined when the queue is empty and the set of best solutions holds the preferred solutions for the problem.

Since the BFS algorithm examines every possible solution to a problem, the result provides a set of equivalent optimal solutions for the problem. However, the queue used to track partial solutions grows as a function of the number of candidate assignments are valid for a partial solution. Since the growth is factorial and every node of a layer is in the queue at one time, resource consumption is high. The best, worst, and average search time is the complete size of the search space, $O(2^{|Y|(|X|+|Y|)})$, for abilities set X and task set Y. Using this search quickly becomes infeasible when either the number of tasks or agents grows. Thus, the complete searching nature of

47

Algorithm 2 Depth-First Search

Require: CMTS problem (M, Y)
Ensure: A solution S^* that is optimal for (M, Y)

1: Let Q be a solution stack
2: Initialize Q with an empty solution
3: Let S^+ be a solution queue initialized with an empty solution

4: **while** Q is not empty **do**
5:      Let $S = top(Q)$, and $Q \leftarrow Q \setminus \{S\}$
6:      Let $\overline{Y}$ be the set of tasks y that are not assigned in S
7:      **if** $\overline{Y}$ is empty **then**
8:          **if** $U(S)$ is better than the utility of the best solution in S^+ **then**
9:              Let $S^+ = \{S\}$
10:         **else if** $U(S)$ equals the utility of the best solution in S^+ **then**
11:             $S^+ \leftarrow S^+ \cup S$
12:         **end if**
13:      **else**
14:         **for all** tasks y in $\overline{Y}$ **do**
15:             Let $P = $ all non-empty subsets of $\{y | y$ is assigned in $S\}$
16:             **for all** coalitions C_y assignable to y **do**
17:                 Let $S' = S \oplus (C_y, y)$ such that y is a child of all tasks in P
18:                 **if** isValid(S') **then**
19:                     Add S' to top of stack Q
20:                 **end if**
21:             **end for**
22:         **end for**
23:      **end if**
24: **end while**
25: Let S^* be any member of S^+
26: **return** S^+

the BFS algorithm becomes a very time- and resource-consuming process useful only for very small problems.

4.2.2 Depth-First Search. The depth-first search (DFS) algorithm, contrary to a BFS, generates solutions by searching the children of a graph node before searching nodes of the same depth. This method opts to generate fully constructed solutions before expanding every possible partial solution. For task assignment problems, each depth in the graph (most often a binary tree) represents a partial solution with an

equal number of assignments. The leaves of the tree represent completed assignment sets and constitute a solution.

The DFS algorithm is modified to use the CMTS definition in the same manner as the BFS algorithm with a graph-based solution. The initial layer contains a node for every possible task in the problem and every possible coalition that could be assigned to the tasks with remaining tasks are tracked at each node. Child nodes are created as explained in Section 4.2.1 until every node on each layer of one branch has been expanded. The leaves hold all of the possible valid solutions to the problem with the preferred solutions being the ones which optimizes the utility value function $U(S)$.

The technical implementation of the CMTS-ready DFS algorithm, shown in Algorithm 2, uses a stack to perform the depth search. The stack is initialized with nodes from the initial layer. Then the node on the top of the stack is popped and all valid extensions are pushed back on top. When no remaining tasks are identified for the topmost entry, the node holds a complete solution that is evaluated for a utility value (Equation 3.2) to see if it belongs in the best solutions set. All possible solutions have been examined when the stack is empty, and the set of best solutions holds the preferred solutions for the problem.

Since the DFS algorithm also examines every possible solution to a problem, the set of solutions provided as an answer is the optimal solution set for the problem. This approach uses depth over breadth, solutions are generated quickly allowing this approach to be used as an anytime algorithm. These attributes represent the benefits of implementing a DFS. Although, there is no guarantee on solution quality at any intermediate iteration—the best solutions could be equally found first or last.

The best, worst, and average search time is the same as BFS, the complete size of the search space, $O(2^{|Y|(|X|+|Y|)})$, for abilities set X and task set Y. The stack used to track partial solutions grows factorially as a function of the number of candidate assignments causing high resource consumption. So, using this search also becomes

quickly infeasible when either the number of tasks or agents grows. However, nodes with complete solutions are found quickly, so the stack-based implementation uses substantially less memory that the queue-based BFS implementation since solution nodes are removed early and frequently during processing. Still, the complete searching nature of the DFS algorithm is a very time-consuming process with significant resource usage.

4.3   Tree-based Heuristic Search Algorithms

Complementary to tree-based complete searches are the generally more efficient heuristic-based tree searches. The basic premise remains the same: the head of the tree is an empty solution with branches being built by appending an assignment to the parent node's partial solution. The difference comes in the selection of which branches are selected for expansion. Whereas BFS and DFS expand every possible branch, heuristic searches select which branches to expand based on their predictive solution quality before all of the branches are explored.

This section presents a modified A* and two beam search algorithms to generate solutions to the CMTS problem. The A* approach takes advantage of the estimated utility $E[U(S)]$ to rank partial solutions that provide the best valued solution. Two types of beam search extend the A* paradigm by bounding the number of partial solutions that are examined in different ways.

4.3.1 A.*   The A* algorithm helps functions like a DFS, but uses heuristics to selectively choose which nodes offer the most promise for finding the best solutions. Branches with poor heuristic evaluation may not be expanded if an acceptable solution is found first. If the heuristic is admissible [103], then the A* algorithm is guaranteed to find an optimal solution. Thus, the CMTS-solving A* algorithm, implemented by Algorithm 3, provides the convenience of finding an optimal solution to a CMTS problem like the DFS algorithm, but possibly with considerably less graph searching.

Algorithm 3 A*

Require: CMTS problem (M, Y)
Ensure: A solution S^* that is optimal for (M, Y)

1: Let Q be a solution priority queue, prioritized by $E[U(S)]$, Eq 3.3
2: Initialize Q with an empty solution

3: **while** Q is not empty **do**
4:     Let $S = front(Q)$, and $Q \leftarrow Q \setminus \{S\}$
5:     Let $\overline{Y}$ be the set of tasks y that are not assigned in S
6:     **if** $\overline{Y}$ is empty **then**
7:         $S^* = S$; **return**
8:     **else**
9:         **for all** tasks y in $\overline{Y}$ **do**
10:             Let $P =$ all non-empty subsets of $\{y|y$ is assigned in $S\}$
11:             **for all** coalitions C_y assignable to y **do**
12:                 Let $S' = S \oplus (C_y, y)$ such that y is a child of all tasks in P
13:                 **if** isValid(S') **then**
14:                     Add S' by priority $E[U(S)]$ to Q
15:                 **end if**
16:             **end for**
17:         **end for**
18:     **end if**
19: **end while**

The CMTS A* scheduling algorithm constructs a set of agent abilities from the agents provided and inserts candidate assignments into partial solutions similar to the depth-first search algorithm (in Section 4.2.2), but uses a priority queue to manage which nodes are chosen for extension. The nodes are prioritized by the utility value of the associated partial solution ($g()$) plus a heuristic for estimating the value of scheduling the remaining tasks ($h()$) to form a priority value ($f() = g() + h()$).

Child nodes, including leaves with complete solutions, are ordered in the priority queue according to their priority value. Nodes with better priority values move closer to the front of the queue. The node with the best priority value sits at the front. When the front of the queue contains a node with a complete solution, that solution is provided as the best solution for the given problem.

The benefit of using this algorithmic approach is that the search graph is evaluated in an ordered manner with potentially better candidate solutions begin examined earlier. Also, if the heuristic element ($h()$) is admissible and consistent, then the solution found on the front of the queue is guaranteed to be an optimal solution. So, the A* approach can provide an intelligent search of the search space that does not require a complete search and still produces an optimal solution. This results in a faster delivery of the best solution.

However, unlike DFS, A* cannot serve as an anytime algorithm since complete solutions are not necessarily quickly developed. The algorithm backs-up to a previous partial solution if those assignments appear to have a better chance of providing a better solution (based on the utility value of the solution with the heuristic estimation). So, there is no guarantee that A* has generated a complete solution at any intermediate iteration.

The backup ability of A* also leads to another disadvantage, namely bookkeeping overhead. For problems where partial solutions tend to have a relatively uniform heuristic element, child nodes move to the back of the priority queue. This creates a situation where every possible partial solution is evaluated and tracked. This behavior effectively makes A* solve like a DFS and adds considerable memory overhead to store the backup information, e.g., heuristic values.

Another potential limitation is that A* only provides one solution, the first suitable solution discovered. Problems that use multi-objective values generally desire a set of solutions that represent optimization of a multi-objective function. This algorithm is not designed to meet the needs of multi-objective problems that need an optimal set of solutions.

4.3.2 Beam Search. Beam search [57] involves searching a tree using heuristic values to eliminate branches that do not seem to hold promising solutions. Branches can be heuristically eliminated, for instance, if the assignment solution utility value is

over a cut-off value for a minimization problem. The cut-off value is usually defined by elements of the domain.

This algorithm fortunately is a minor adaptation of the A* algorithm since that algorithm similarly involves building a tree structure to find solutions. But rather than having approximately $2my$ potential children for any given node, the beam search only chooses those within a top percentage of desirable values of the utility value of any solution using Equation 3.3 as the heuristic function. This process happens at each level of depth so that only a fraction of the tree is searched.

This technique significantly speeds up the time to generate a final solution, but unless the selection bound includes all children, it does not completely search the solution space. Therefore, beam search is not guaranteed to find an optimal solution. However, given the potentially large size of the trees, this search quickly finds good approximations given a suitable estimation of solution utility values.

Two types of beam search are presented in this section. The first CMTS-solving beam searching algorithm, Algorithm 4, is a derivative of the A* scheduling algorithm (4.3.1), differing only by bounding the number of nodes retained in the queue to a given size. It applies the same heuristic-based approach to order child nodes generated by inserting candidate assignments, but truncates the priority queue to a fixed size after all of the extended nodes are inserted into the queue. The solution on the front of the queue is the best solution for the problem.

The second CMTS-solving beam search algorithm, Algorithm 5, modifies the DFS scheduling algorithm (4.2.2) by replacing the stack with a priority queue and using a retention criteria based on the expected value of the best seen partial solution value. The algorithm requires a deviation percentage, ρ, to bound the maximum allowable difference of the expected utility value of a partial schedule solution from the best seen expected value. Child nodes are constructed identically to the DFS scheduling algorithm and inserted into the priority queue based on the expected value of their partial solution. After all child nodes are inserted into the queue, all nodes

---
Algorithm 4 Beam Search

Require: CMTS problem (M, Y)
Require: Beam size MAX
Ensure: A solution S^* for (M, Y)

1: Let Q be a solution priority queue, prioritized by $E[U(S)]$, Eq 3.3
2: Initialize Q with an empty solution

3: **while** Q is not empty **do**
4:     Let $S = front(Q)$, and $Q \leftarrow Q \setminus \{S\}$
5:     Let $\overline{Y}$ be the set of tasks y that are not assigned in S
6:     **if** $\overline{Y}$ is empty **then**
7:         $S^* = S$; **return**
8:     **else**
9:         **for all** tasks y in $\overline{Y}$ **do**
10:            Let $P =$ all non-empty subsets of $\{y | y$ is assigned in $S\}$
11:            **for all** coalitions C_y assignable to y **do**
12:                Let $S' = S \oplus (C_y, y)$ such that y is a child of all tasks in P
13:                **if** isValid(S') **then**
14:                    Add S' by priority $E[U(S)]$ to Q
15:                    **if** $size(Q) > MAX$ **then**
16:                        Remove the last (back) element of Q
17:                    **end if**
18:                **end if**
19:            **end for**
20:         **end for**
21:     **end if**
22: **end while**
---

with a partial solution utility over ρ percent of the partial solution utility on the front of the queue are removed. The remaining nodes form the basis of the next iteration. All solutions with equivalent value as the solution on the front of the queue are returned as the best solution set.

The benefit provided by both of these approaches is that they provide intelligent bounding of the search space ensuring only the most promising candidates are explored. The bulk elimination of candidate partial solutions allows reduced bookkeeping depending on the size or percentage limitations. In either algorithm, however, the limitations can be set high enough that the size-based beam search performs

---

Algorithm 5 Value-based Beam Search
Require: CMTS problem (M, Y)
Require: Beam value percentage R
Ensure: A solution S^* for (M, Y)

 1: Let Q be a solution priority queue, prioritized by $E[U(S)]$, Eq 3.3
 2: Initialize Q with an empty solution

 3: **while** Q is not empty **do**
 4:      Let $S = front(Q)$, and $Q \leftarrow Q \setminus \{S\}$
 5:      Let $\overline{Y}$ be the set of tasks y that are not assigned in S
 6:      **if** $\overline{Y}$ is empty **then**
 7:         $S^* = S$; **return**
 8:      **else**
 9:         **for all** tasks y in $\overline{Y}$ **do**
10:            Let $P = $ all non-empty subsets of $\{y | y$ is assigned in $S\}$
11:            **for all** coalitions C_y assignable to y **do**
12:               Let $S' = S \oplus (C_y, y)$ such that y is a child of all tasks in P
13:               **if** isValid(S') **then**
14:                  Add S' by priority $E[U(S)]$ to Q
15:               **end if**
16:            **end for**
17:         **end for**
18:         **for all** S' in the Q **do**
      $\triangleright$ Assuming a maximization check; use the appropriate comparitor
19:            **if** $|E[U(S')] - E[U(Q\text{'s front})]| < R \cdot E[U(Q\text{'s front})]$ **then**
20:               Remove S' from Q
21:            **end if**
22:         **end for**
23:      **end if**
24: **end while**

---

equivalent to A*, and the value-based beam search like an ordered DFS. For carefully selected limits, which very likely requires domain knowledge, either algorithm will readily outperform the algorithm they are derived from. The value-based beam search with a low percentage value can potentially search less of a graph than the efficient size-based beam search.

However, each algorithm relies on having a suitable limitation value (size or percentage) to balance memory overhead with sufficient graph searching. Finding such

a value, not within the scope of this work, is challenging without knowing domain-specific characteristics to guide the selection of these values. This can limit the effectiveness of the solutions provided as the best.

Also, neither algorithm is necessarily a complete search algorithm. The solutions generated by these algorithms are susceptible to being local maxima versus global. Even as a derivative of A*, the size beam search cannot guarantee an optimal solution with an admissible heuristic due to the elimination of candidate partial solutions.

4.4   Greedy Auction Algorithms

Greedy algorithms generate solutions by taking the most valuable choice, or the local optimum, at each stage of constructing a solution. Auction methods solicit bids for the value of a partial solution and generally selects the bid that provides the most value, similar to greedy algorithms. For CMTS problems, a greedy auction method can choose to locally optimize on the value of the coalition, assignment, task ordering, or any combination of these elements.

This section presents two modified versions of auctioning methods using greedy algorithm techniques for generating serial solutions similar to a serial project scheduler and concurrent solutions similar to a parallel project scheduler. First, the serial auction locally optimizes on only the assignment values to generate a solution. Complementary, the concurrent auction locally optimizes on the coalition membership relative to maximizing concurrent task ordering using a bin-packing approach. Algorithmic solutions to the multi-tasking agents on multi-agent tasks (MT-MR) category of the MTRA taxonomy [42, 45] combine the serial and concurrent auction methods to locally optimize on both assignment value and task ordering.

4.4.1 Serial Auctioning.   The serial auctioning algorithm, based on AL-LIANCE [96], creates a serial schedule of assignments based on a greedy selection criteria. Every task is performed in a given order depending on which coalition-task pairing offers the most optimal value. Initially, the assignment with the best value

Algorithm 6 Serial Auction
Require: CMTS problem (M, Y)
Ensure: A solution S^* for (M, Y)

1: Initialize S^* to the empty solution
2: Let P be a task, initially *null*
3: Let $\overline{Y} = Y$
4: **while** $\overline{Y}$ is not empty **do**
5:     Let A = the set of assignments $\{(C_y, y)\}$, $\forall y \in \overline{Y}, \forall C_y \ni C_y$ performs y
6:     Choose $(C_y, y) \in A$ such that $U(S^* \oplus (C_y, y))$ is the best utility value with y
    a child of P
7:     $S^* \leftarrow S^* \oplus (C_y, y)$ such that y is a child of task P
8:     $P \leftarrow P \cup y$
9:     $\overline{Y} \leftarrow \overline{Y} \setminus \{y\}$
10: **end while**
11: **return**

is scheduled first. Subsequent assignments are ordered similarly until each task is arranged in a solution.

The implementation of the CMTS-ready serial auctioning algorithm, shown in Algorithm 6, begins with an empty partial solution and initializes the set of remaining tasks to the complete set of tasks in the problem. Then, for every task in the remaining set, a derivative set of solutions is generated with each new partial solution having one new assignment for every possible coalition to each remaining task. The partial solution with the best utility value is selected the best seen solution and the task of the assignment given to the solution is removed from the remaining tasks list. This process continues until the remaining tasks list is empty. The resulting solution is the final solution for the serial auctioning algorithm.

This process generates the best possible sequential schedule for the set of problem tasks. But, it is not necessarily optimal since it fails to consider concurrently performed tasks that can be performed concurrently. However, if constraints shape the problem such that only a serial solution is valid, then this algorithm is optimal.

Although it represents a qualitative improvement over the randomly selected sequential algorithm in Section 4.6.2, it requires the construction of every possible

---
Algorithm 7 Concurrent Auction

Require: CMTS problem (M, Y)
Ensure: A solution S^* for (M, Y)

1: Initialize S^* to the empty solution
2: Let P be a set of parent tasks, and R a set of sibling tasks, each initially empty
3: Let X be the set of all agent abilities, $X = \bigcup C_m \forall m \in M$
4: Let $\overline{Y} = Y$
5: Let $\overline{X} = X$
6: **while** $\overline{Y}$ is not empty **do**
7:      Let $A =$ the set of assignments $\{(C_y, y)\}$, $\forall y \in \overline{Y}, \forall C_y \in 2^{\overline{X}} \ni C_y$ performs y
8:      **if** A is not empty **then**
     $\triangleright$ Assignments are available from the remaining tasks and abilities
     $\triangleright$ So, add the best but don't descend
9:          Choose $(C_y, y) \in A$ such that $U(S^* \oplus (C_y, y))$ is the best utility value with y a child of every task in P
10:          $S^* \leftarrow S^* \oplus (C_y, y)$ such that y is a child of every task in P
11:          $R \leftarrow R \cup y$
12:          $\overline{Y} \leftarrow \overline{Y} \setminus \{y\}$
13:          $\overline{X} \leftarrow \overline{X} \setminus C_y$; reduce the eligible abilities for the next task
14:      **else**
15:          $\triangleright$ Otherwise, descend to the next layer of the solution
16:          $P \leftarrow R$; reset the parent set to the siblings
17:          Empty the sibling set R
18:          $\overline{X} \leftarrow X$; Reset the abilities set to all available members
19:      **end if**
20: **end while**
21: **return**
---

assignment for each step of the process. This is a significant increase in processing time, but since it is theoretically only necessary to save two partial solutions at any given moment, the best seen and the one-extended solutions, there is no additional memory overhead. Thus, this algorithm scales very well to problems of any size in a memory-constrained environment. However, the implementation of Algorithm 6 does store the combinatorial list of assignments, so memory usage becomes an issue for large problems.

4.4.2 Concurrent Auctioning. The concurrent auctioning algorithm, based on the Broadcast of Local Eligibility (BLE) [121] algorithm, takes the serial auc-

tioning algorithm one step further by accounting for tasks that can be performed concurrently. Instead of necessarily scheduling each task alone in a sequence, the concurrent auction allows multiple tasks per depth in a solution. Whereas the serial auction algorithm's solutions are a linear graph with an average degree of one, the solutions to this algorithm allow nodes with degree greater than one. Essentially, the concurrent algorithm creates as many assignments as possible per step rather than optimize for just one assignment.

The CMTS-ready concurrent auctioning algorithm, shown in Algorithm 7, starts with an empty solution and a set of remaining tasks equal to the initial set of problem tasks. Added are a set of remaining abilities equal to the initial set of abilities available in the problem and an initially empty set of parent and child assignments.

Then, while the remaining task set is non-empty, a new partial solution is created for every possible assignment covering the remaining tasks and remaining abilities. If there are valid partial solutions, the assignment of the best partial solution becomes a member of the child assignment set, and the abilities used in the assignment are removed from the remaining abilities set. The assignment becomes a child of every assignment held in the parent assignment set. The task in the assignment is removed from set of remaining tasks. However, if there are no possible assignments for the remaining entity sets, then the remaining abilities set is reset to the initial set of abilities and the set of parent assignments equals the set of child assignment. The child assignment set is then emptied. This forces subsequent assignments into the next layer of the solution starting with a fresh set of agent abilities. This process continues until the set of remaining tasks is empty. The resulting solution is the final answer for the concurrent auctioning algorithm.

This method continually evaluates every possible assignment in the problem but fails to provide an optimal solution since assigning the remaining agent abilities to multiple coalitions is not considered. Instead the algorithm takes a greedy bin-packing approach. Otherwise, if the algorithm did check every distribution of the remaining

agents over the remaining tasks then it would find an optimal solution. As a matter of fact, this is exactly how the DFS algorithm works.

Since concurrent tasks are accounted for in this algorithm, it represents an improvement over the serial algorithm for problems where concurrent task execution is beneficial. Since the size of the additional data structures used to solve this version of the auction are fixed and based solely on the number of entities in the problem, only a trivial amount of overhead is added compared to the serial scheduler. But since it necessarily searches for concurrent opportunities, problems which penalize concurrent execution may cause the concurrent auction to generate solutions with less quality than a serial approach.

4.5 Constructive Stochastic Algorithms

Constructive stochastic methods involve building solutions from initially empty solutions using non-deterministic methods, principally random selection and other Monte Carlo methods. This is one of the key differences between the algorithms described in this section and deterministic algorithms. As with deterministic algorithms, there are several types of stochastic methods that are used to construct solutions.

This section describes two stochastic approaches. First is the biologically-inspired ant colony optimization (ACO) which presents an alternative heuristic construction to solve the multiagent problem using digital pheromone trails updated by multiple search agents. Alternately, reinforcement learning (RL) presents a way for agents to learn how to build the best assignment solution by making step-by-step construction choices with a reward feedback function to guide subsequent assignment selections.

4.5.1 Ant Colony Optimization. Another paradigm to generating multiagent task assignment solutions uses ant colony optimization (ACO) [11, 29, 30]. ACO, outlined in Algorithm 9, uses a biologically inspired approach to solving combinatorial optimization problems.

Algorithm 8 ACO Graph Building Method

1: **procedure** BUILDGRAPH(M, Y)

2:   Let $X = \bigcup C_m \forall m \in M$, the set of all abilities

3:   ▷ Build a set of all coalitions that perform a given task

4:   Let $W_y = \{(C_y, y) \ni C_y \subseteq X \text{ and } C_y \text{ performs } Y\}$

5:   ▷ Build a set of all assignments involving concurrently performed tasks

6:   Let Z be all non-empty subsets of Y

7:   $\forall z \in Z$, let $A_z = \{(C_y, y) \ni y \in z \text{ and } C_{y_i} \bigcap C_{y_j} = \emptyset \forall y_i, y_j \in z\}$

8:   ▷ Construct an assignment graph for ACO

9:   Construct G with nodes from $\bigcup_{z \in Z} A_z$

10:   Edge(A_{z_i}, A_{z_j}) exists iff $\{y \in A_{z_i}\} \bigcap \{y \in A_{z_j}\} = \emptyset, \forall z_i, z_j \in Z$

  return Graph G

11: **end procedure**

In ACO algorithms, ants (agents) navigate a graph representation of a problem in order to construct a solution. Ants deposit pheromone on graph edges used in their solutions relative to the quality of the solutions they construct. Over time, edges involved in high-quality solutions contain higher pheromone concentrations than edges used in good or low-quality solutions. As such, the pheromone serves as a collective memory about which edges contribute to the best solutions. During solution construction, an ant's decision to move from one node to another is directly influenced by the amount of pheromone on each edge and a heuristic desirability of each edge. Outgoing edges with high pheromone concentrations and/or high heuristic values are more likely to be selected than other outgoing edges. The ACO approach for building CMTS scheduling solutions incorporates the WoLF Ant ACO algorithm [26], which incorporates a policy-based decision matrix (see Section 4.5.2) into the heuristic component to guide the selection of edges chosen for a solution.

The nodes of a graph contain sets of assignments that represent a layer of the search graph. The nodes are constructed by selecting a subset of a every possible assignment for each task based on matching agent abilities. Every legal set of assignments has a node in the graph. Directional edges between nodes indicate that every assignment of the first node serve as parents to every assignment of the second node, and conversely, every assignment of the second node is a child of every assignment

Algorithm 9 WoLF Ant Algorithm

Require: CMTS problem (M, Y)
Require: A random walk criteria $q_0 > 0$
Ensure: A schedule S^* holding a schedule that is best for (M, Y)

1: Build graph $G = \text{BUILDGRAPH}(M, Y)$
2: Let Z be all non-empty subsets of Y
3: $\forall z \in Z$, let $A_z = \{(C_y, y) \ni y \in z \text{ and } C_{y_i} \bigcap C_{y_j} = \emptyset \forall y_i, y_j \in z\}$

4: Initialize pheromones $\tau_{ij} = \tau_0$, policy $\pi_{ij} = 1/|E_{ij}|$, $\Delta_{ij} = \Delta_{ij}^2 = 0$
5: Let heuristic $\eta_{ij} = E[U(S_k)]$, the expected value of ant k's solution where A_{z_i}
    parents A_{z_j}
6: **for** $i \leftarrow 1, MAX\_ITERATION$ **do**
7:     For each ant k, initialize solution S_k to random assignment set $v_{k,0} = A_{z_{random}}$
8:     $\triangleright$ Ants build solutions by adding assignments sets from a subsequent node
9:     **while** some ant is still building a schedule **do**
10:       **for** each ant k **do**
11:         Let $v_{k,i}$ be the ant's latest assignment set
12:         **if** there are unassigned tasks in S_k **then**
13:           $\triangleright$ Choose assignment set as valid child to $v_{k,i}$
14:           Let $A = \{A_z \ni \{y \in S_k\} \bigcap \{y \in A_z\} = \emptyset$
15:           **if** $random \leq q_0$ **then**
16:             select $v_{k,j} \in A$ according to Eq 4.2
17:           **else**
18:             select $v_{k,j} \in A$ according to Eq 4.1
19:           **end if**
20:           $\forall (C_y, y) \in v_{k,j}$, set (C_y, y) as child to $(C_y', y') \in v_{k,i}$ in S_k
21:           If isValid(S_k), update pheromone τ_{ij}
22:         **end if**
23:       **end for**
24:     **end while**
25:     $\triangleright$ Each ant now has a solution, so choose the best then update values
26:     Choose S^* so that $U(S^*) = \text{ argmax } \{U(S_k) \forall k, U(S^*)\}$
27:     $\triangleright$ Elitist update the pheromones and policy
28:     **for all** $v_{*,i}, v_{*,j} \in S^*$ **do**
29:       update pheromones $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau$ (from [26])
30:       calculate policy change $\Delta\pi_{ij}$ (from [26])
31:       update policy $\pi_{ij} \leftarrow \pi_{ij} + \Delta\pi_{ij}$
32:       normalize π_i, update $\Delta_{ij}^2 \leftarrow \Delta\pi_{ij} - \Delta_{ij}$, $\Delta_{ij} \leftarrow \Delta\pi_{ij}$
33:     **end for**
34: **end for**
35: **return** solution S^*

---

in the first node. Edges that cause constraint violations, i.e., violates inhibitions, prerequisites or solution form rules, are not allowed in the graph.

The ACO approach of the development of a best solution occurs over a fixed number of iterations. Each iteration, ants develop a complete solution based on the values on the edges of the problem graph. The overall best solution is tracked over each iteration as the best solution developed during any iteration by an ant. The number of ants that are used to generate solutions is based on a given percentage. Empirical studies show a value of $60\% - 100\%$ of the graph size seem sufficient to produce highly effective solutions [11].

During an iteration, each ant is initialized to a randomly selected node which serves as the first layer of assignments in the ant's solution. Each ant iteratively builds its solution by choosing an edge to follow to a subsequent set of assignments which form the next layer of the solution graph. The selection of an edge is biased by a random walk value. If a randomly selected value $q \in [0,1]$ is less than a given bound, q_0, then the ant chooses the edge E_{ij} that maximizes the pheromone selection rule in Equation 4.1. Otherwise, the ant randomly chooses an edge to form its extended partial solution through weighted selection based on Equation 4.2. Pheromone values are referred as τ, policy by π, and heuristic by η.

$$P_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta \cdot [\pi_{ij}]^\phi}{\sum_{l \in J_i} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta \cdot [\pi_{il}]^\phi} \tag{4.1}$$

$$j = \text{argmax}_{j \in J_i} \{ [\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta \cdot [\pi_{ij}]^\phi \} \tag{4.2}$$

After an ant selects an edge, it increases the pheromone value (τ) by some given rate. This encourages other ants to use the same edge while extending their partial solutions. This process continues until each ant has a complete solution. The best solution from the group is identified and compared to the best overall solution. Finally, the pheromone and policy values of the edges in the overall best solution are updated to guide ants for next iteration solution development.

The main benefit to using WoLF Ant is the fast convergence rate [26] to finding a solution. The efficiency of the WoLF Ant algorithm generally offsets the additional computational overhead required to find high quality solutions. ACO is known to explore the solution space very well looking for alternatives that lead to better, perhaps optimal, solutions. Because ants develop solutions as quickly as possible, the WoLF Ant approach is an anytime algorithm like the DFS scheduling algorithm.

Also, since the nodes and edges of the problem graph are created ahead of time and do not change, there is a fixed, determinable memory requirement. The ants generate as many solutions as desired without requiring any more or less bookkeeping. However, large problem sizes can be problematic since the problem graph size increases exponentially based on the number and capabilities of each agent in the problem. And increasing the number of tasks also results in a combinatorial increase of problem nodes. Problems may easily require extensive amounts of resources to represent the search graph used to produce a solution.

4.5.2 Reinforcement Learning.   Another means to approximate optimal solutions is policy-based reinforcement learning (RL) [112]. RL provides a structure where agents are rewarded for doing tasks. This reward is used to learn which tasks are most appropriate for an agent. A value function captures the influence of rewards given for taking actions over time to generate a policy. The policy is used by the agent to determine which tasks to select given a state of the partial solution.

The basic approach for using policy-based reinforcement learning with CMTS problems, presented in Algorithm 10, is to implement a hybrid reinforcement learning value function with Q-learning [119] and shared policy cooperation. The expected value of an assignment solution, $E[U(S)]$, serves as the reward-generating value function, $V(t)$ for Q-learning. Each agent has a policy, π, of state-action pairs where states are represented as a combination of agents and unassigned tasks and actions are the selection of a task to perform.

Algorithm 10 Reinforcement Learning Algorithm

Require: CMTS problem (M, Y)
Require: PDWoLF and reinforcement learning constants, $\alpha, \gamma, \delta_w, \delta_l$
Require: Exploration constant $q_0 \in [0, 1]$
Ensure: A solution S^* that is optimal for (M, Y)

1: Let S be the set of all possible assignment set $\{(C_y, y)\} \forall y$ in every subset of Y
2: Let A_s be the set of all possible child tasks for a state $s \in S$
3: Initialize $Q(s \in S, a \in A_s) \leftarrow 0$, $\pi(s \in S, a \in A_s) \leftarrow 1/|A_s| - 1$, PDWoLF, WSS
4: Initialize solution S^* to the empty solution
5: Initialize assignment set P to the empty set
6: **while** Converging... **do**
7:     Observe state s as P
8:     **if** random $\geq q_0$ **then**
9:         Randomly choose action $a = \{(C_y, y)\} \in A_s$
10:     **else**
11:         Choose action $a = \{(C_y, y)\} \leftarrow textrmargmax\pi(s)$
12:     **end if**
13:     Update $S^* \leftarrow S^* \oplus a$ such that all $(C_y, y) \in a$ is a child of all tasks in P
14:     Update $P \leftarrow a$
15:     Observe reward $r = E[U(S^*)]$ and next state s' as P
16:     Update $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha [r + \gamma argmax_{a'\ inA_s} Q(s', a')]$
17:     Update $\pi(s, a) \leftarrow \pi(s, a) + \Delta\pi(s, a)$ where $\Delta\pi(s, a)$ comes from Equation 4.3

18:     **if** time to cooperate **then**
19:         $\triangleright$ Cooperate policy π using weighted strategy sharing (WSS)
20:         Update π by Equation 4.5
21:     **end if**
22:     **if** S^* is a complete solution **then**
23:         Reset S^* to the empty solution, and P to the empty set
24:     **end if**
25: **end while**

26: $\triangleright$ Reconstruct the final solution based on maximal policy values
27: Reset S^* to the empty solution, and P to the empty set
28: Initialize s to the no-task state
29: **while** $\exists y \in Y$ and $y \notin S^*$ **do**
30:     Choose action $a = \{(C_y, y)\}$ so that $\pi(s, a)$ is maximal
31:     Update $S^* \leftarrow S^* \oplus a$ such that all $(C_y, y) \in a$ is a child of all tasks in P
32:     Update $P \leftarrow a$
33:     Set $s \leftarrow P$
34: **end while**
35: **return** S^*

Each agent determines which task to perform as an action based on its policy. Agents then deconflict actions using a variety of methods such as voting or auctioning to settle conflicts in a potential solution. The cumulative result of the agents selecting tasks results in a series of assignments that fit into an existing solution. The expected value of the new partial solution drives the computation of the reward values for the Q-learning process, which then updates the policy. This process iterates with agents continually selecting assignments until the policy stabilizes to an equilibrium. Equation 4.3 shows the update rule for agent m transitioning from one state (partial solution) to another.

$$\pi(s_t, a_t) \leftarrow (1 - \alpha)\pi(s_t, a_t) + \alpha\left(r + \gamma \text{argmax}_{a' \in A} V(s_{t+1}, a')\right) \tag{4.3}$$

During the iterative process, agents cooperate policies using weighted strategy sharing (WSS) [2] in order to converge solutions to a global optimum. Each agent builds an expertness metric [1,2,34] based on the rewards received for constructing a partial schedule. This metric, ε^m, is used to weight the qualitative value of the policies of other agents solving the problem. The weights, ω, are computed by Equation 4.4 for a group of agents that only incorporate policy information from agents that are performing better than itself. The weight is biased by an influence parameter, ρ, to control how sensitive an agent is to changing its policy. Then for every state that an agent needs updated in its policy, it applies the weight formula to combine the collective values of all equal or better performing agents into its own policy, as shown in Equation 4.5.

$$\omega_{mn} = \begin{cases} 1 - \rho, & \text{if } m = n \\ \rho \cdot \frac{\varepsilon_n - \varepsilon_m}{\sum_{z=1}^{|M|} \varepsilon_z - \varepsilon_m}, & \text{if } \varepsilon_n \geq \varepsilon_m \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

$$\pi_s^m = \sum_{n=1}^{|M|} \omega_{mn} \cdot \pi_s^m \tag{4.5}$$

A final solution is available when agents complete several iterations of solution building, possibly resulting in the convergence to a Nash equilibrium indicating an optimal policy. The solution is derived by following the series of actions taken by an agent from the converged policy to build a complete schedule of assignments.

When a proper value function is implemented in RL, the policy converges optimally. This implies that optimal solutions can be found, but only over infinite time. However, convergence to an optimal policy is aided by applying fast gradient ascent methods, such as "Win or Learn Fast" (WoLF) [15, 16] and policy-dynamics WoLF (PDWoLF) [8], to the value functions feeding the policy. These methods provide meta-information about how the policy is converging and allow agents to select tasks based on the quality of the solution's utility over time versus selecting tasks based directly on the utility. Using the gradient information of a policy to make the decision often results in faster convergence [24, 26], better solutions and better response to dynamic environments where the utility of a solution can change over time.

The most significant drawback to using policies is that the state space must be stored in the policy. The state space for a sufficiently large problem can readily overwhelm the resources of any machine. One approach to deal with this issue is to approximate the state space with a function approximation [112], but this can lead to slower convergence rates or higher deviation from optimal solutions.

4.6   Random Algorithms

Random methods provide a generally quick way to construct a solution without regard to the quality of the solution. CMTS problem solutions are randomly constructed similar to the methods used for greedy auctioning, i.e., stepwise selection of coalitions and sequencing, but without necessarily selecting the local optimum. Additionally, random methods can directly produce complete solutions by simply selecting an index that maps to a solution in the solution space.

Algorithm 11 Random Solution Method

 1: **procedure** MAKERANDOM(M, Y)
 2:      Initialize S to the empty solution
 3:      **for all** $y \in Y$ **do**
 4:          Randomly choose P as a subset of $\{y | y$ is assigned in $S\}$
 5:          Randomly choose coalition $C_y \subseteq \bigcup C_m \forall m \in M$ such that C_y performs y
 6:          $S \leftarrow S \oplus (C_y, y)$ such that y is a child of all tasks in P
 7:      **end for**
 8:      **return** S
 9: **end procedure**

Algorithm 12 Random Search

Require: CMTS problem (M, Y)
Ensure: A solution S^* for (M, Y)

 1: Initialize S^* to the empty solution
 2: **while** not $isValid(S^*)$ **do**
 3:     Let $S^* = MAKERANDOM(M, Y)$
 4: **end while**
 5: **return**

This section presents three types of random algorithms. The first two are related random algorithms that produce any type of solution graph either by selecting a random solution from the complete solution space until a valid solution is chosen, or by selecting a random solution from the solution space and "smartly" modifying it until it is valid. Alternately, the third random approach only builds serial sequences similar to the serial auction method, but without necessarily choosing the local optimum as the preferred extension.

4.6.1 Random Search. The random search provides a simple approach to solving a task assignment problem. The search produces solutions by a random selection without using any information about a problem other than knowing which items are selectable. There are no general parameters used to configure this algorithm for developing a solution.

The simplest random search approach solves a CMTS problem in two phases. First, for every task in the problem, a randomly selected subset of agent abilities

Algorithm 13 Smart Random Search
Require: CMTS problem (M, Y)
Require: Number of iterations N
Ensure: A solution S^* for (M, Y)

1: **for** $I = 1$ **do** N
2:     Initialize S to the empty solution
3:     **while** not $isValid(S)$ **do**
4:       Let $S = MAKERANDOM(M, Y)$
5:       $S \leftarrow REPAIR(S)$
6:     **end while**
7:     **if** $U(S)$ is better than $U(S^*)$ **then**
8:       $S^* \leftarrow S$
9:     **end if**
10: **end for**
11: **return**

that are able to perform a task are formed into a coalition for that task. Then, the assignment is randomly placed as a child to any number of other assignments present in a partial solution to determine when the task is to be performed. This method is defined in Algorithm 11. Since the method does not necessarily construct several branches of partial solutions, the matrix representation is used. The resulting matrix is the solution to the problem. An algorithm to perform a random search for a CMTS problem is provided in Algorithm 12.

Clearly, without some checking, the random search is likely to generate invalid solutions when the problem involves constraints. One way to mitigate this concern is to throw away the invalid solution and perform subsequent iterations until a valid solution is produced. Invalid solutions are tracked so that they are not repeated to avoid infinite looping. Or, the selection of random elements is restricted as tasks are assigned so that invalid solutions are less likely to form. However, it is still possible to generate an invalid partial solution which would again require invalid solution tracking and subsequent iteration to find a valid solution.

Alternately, a "smarter" random algorithm, provided in Algorithm 13, performs some checks through a repair mechanism used for the genetic algorithm described in

Section 4.7.1. The smart-random algorithm is still subject to invalid solutions, but generally requires far fewer iterations to produce a solution.

Since this search relies on random selection, there is no guarantee for generating an optimal solution. Quite often, the solutions are mediocre at best. However, the solution generation time has a best case time of $O(1)$. The worst case is $O(2^{|Y|(|X|+|Y|)})$, the maximum size of the solution space, provided invalid solutions are tracked to avoid infinite looping. The average case depends on the number of constraints involved in the problem. With no constraints, the average case is the same as the best case, $O(1)$; with constraints the average case approaches the LUB (Equation 3.7) as the number of constaints increase.

The poor performance in the presence of constraints and likely probability of mediocre solution quality are the major drawbacks for using the random search. However, there are benefits to performing this search. In lightly constrained problems, the solutions come quickly, i.e., few iterations are required to generate a valid solution. This can be exploited by other algorithms that need a seed set of solutions, such as the genetic algorithm (Section 4.7.1) or particle swarm optimization (Section 4.7.2). Also, the results of this search can provide a benchmark to compare the quality of other computationally intensive algorithms, such as the genetic algorithm, ant colony optimization (Section 4.5.1) and reinforcement learning (Section 4.5.2). If the solution quality of another algorithm does not statistically outweigh the quality of a random selection, then the algorithm could be rejected based on increased computational complexity.

4.6.2 Task Sequencer. The task sequencer provides another uninformed search technique that is constructive in nature. It behaves more like a traditional serial scheduling algorithm where all able agents perform tasks in some order. Similar to the random search, this search also produces solutions without using any information about a problem. With some heuristic information, this algorithm performs on

70

Algorithm 14 Task Sequencing Algorithm

Require: CMTS problem (M, Y)
Ensure: A solution S^* for (M, Y)

 1: Initialize S^* to the empty solution
 2: Let P be a task, initially *null*
 3: Let $\overline{Y} = Y$
 4: **while** $\overline{Y}$ is not empty **do**
 5:     Let $A =$ the set of assignments $\{(C_y, y)\}$, $\forall y \in \overline{Y}, \forall C_y \ni C_y$ performs y
 6:     Choose $(C_y, y) \in A$ at random
 7:     $S^* \leftarrow S^* \oplus (C_y, y)$ such that y is a child of task P
 8:     $P \leftarrow P \cup y$
 9:     $\overline{Y} \leftarrow \overline{Y} \setminus \{y\}$
10: **end while**
11: **return**

---

par with greedy Monte-Carlo or typical bin-packing methods. There are no general parameters used to configure this algorithm for developing or choosing a solution.

The task sequencer approach solves a CMTS problem by producing a linear graph where every task is assigned a complete coalition of non-conflicting agent abilities that are able to perform that task. The order of the tasks is arbitrary determined by random choice, provided task prerequisite constraints are met. This approach uses the graph-based solution representation. The resulting linear graph is the solution to the problem. An algorithm to perform task sequencing for a CMTS problem is provided in Algorithm 14.

Similar to random search, the random sequencer is likely to generate invalid solutions when the problem involves constraints. This is mitigated by tracking invalid solutions and performing subsequent iterations until a valid solution is produced. If possible, a permutation set of task orderings can be produced where invalid task sequences are removed, and a final solution can be selected from the remaining candidates.

Since this search relies on random selection, there is no guarantee for generating an optimal solution. The solution generation time has a best case time of $O(1)$. The worst case is $O(|Y|!)$, the number of possible task orderings, provided invalid solutions

are tracked to avoid infinite looping. The average case depends on the number of prerequisite constraints involved in the problem. With no constraints, the average case is the same as the best case, $O(1)$; with constraints the average case approaches $O(|Y|!)$ as the number of constaints increase.

The drawbacks for sequential scheduling include potentially poor solution quality and performance in the presence of constraints. The linear graph solutions also fail to take advantage of concurrent task performance where two coalitions perform independent tasks at the same time. This is only significant if the problem domain allows for such situations. However, the benefits mirror those of the random algorithm: a valid solution is generated in fewer iterations and the sequencer can be exploited by other algorithms that need a seed set of solutions.

4.7 Solution-Space Algorithms

The next set of algorithms use domain value instances to guide solution search, but do not construct a solution. Rather, they start with a set of complete, valid solutions and modify them with a set of operators. Modifying a solution graph is not particularly challenging, but incorporating the concepts behind these algorithms into graph modifying operators is not easy. Thus, these algorithms use the matrix form of a solution which is far easier to manipulate, and often is a better suited representation for the defined operations.

Two algorithms presented here use evolutionary algorithm techniques to search the multiagent task assignment solution space for best solutions. The first uses a genetic algorithm (GA) to perform genetic crossover and mutation operations on a set of solution-based chromosomes. The second, particle swarm optimization (PSO), moves a set of solution-based particles in a field by vector-based attraction forces.

4.7.1 Genetic Algorithm. Evolutionary algorithms [6] provide many approaches to solving task assignment problems, such as the genetic algorithm (GA). The GA converges an initial set of complete solutions to an optimal set through

Algorithm 15 Genetic Algorithm
***
Require: CMTS problem (M, Y)
Require: A population size N
Require: An iteration maximum T
Require: Mutation constant $q_0 \in [0, 1]$
Ensure: A solution S^* that is optimal for (M, Y)

1: Initialize population P_0 with randomly constructed solutions
2: **for** $t = 0$ to T **do**
3:     Rank P_t such that $\forall p_i, p_j \in P_t, U(p_i)$ is better than $U(p_j)$
4:     Select elitist population set $E = \left\{ p_1, \ldots, p_{\frac{1}{4}N} \right\}$
5:     Select reproducing population $R = \left\{ p_1, \ldots, p_{\frac{3}{4}N} \right\}$
6:     Initialize new population P_{t+1} to E
7:     **for all** member p_i in R **do**
8:         **if** $random > q_0$ **then**
9:             $\triangleright$ Perform a mutation to p
10:             $P_{t+1} \leftarrow P_{t+1} \cup MUTATE(p_i)$
11:         **else**
12:             $\triangleright$ Else, cross p_i with another member
13:             Randomly select $p_j \in R, i \neq j$
14:             $P_{t+1} \leftarrow P_{t+1} \cup CROSSOVER(p_i, p_j)$
15:         **end if**
16:     **end for**
17:     Rank P_{t+1} such that $\forall p_i, p_j \in P', U(p_i)$ is better than $U(p_j)$
18:     $P_{t+1} \leftarrow \{p_1, \ldots, p_N\} \in P_{t+1}$

19:     $\triangleright$ Check for stagnation
20:     **if** best $p* \in P_t$ is the same as the best $p* \in P_{t+1}$ **then**
21:         increment stagnation count
22:     **else**
23:         reset stagnation count
24:     **end if**
25:     **if** stagnation count is above stagnation threshold **then**
26:         Reset $p_k \in P_{t+1}, k > N/2$ to a random solution
27:     **end if**
28: **end for**
29: **return** $S^* =$ a solution in P such that $U(S^*)$ is best
***

biologically-equivalent genetic methods based on genetic chromosomes. A chromo-
some, for a CMTS problem, represents an assignment solution. Through mutation

---

Algorithm 16 Genetic Algorithm Crossover Method

1: **procedure** CROSSOVER(p_1, p_2)
2:     Randomly choose crossover point x
3:     Create child solution c_1 as $p_1 \otimes p_2$
4:     Create child solution c_2 as $p_2 \otimes p_1$

    $\triangleright$ Using a matrix solution form, the crossover is implemented by
5:     Copy $p_1 \to c_1$ and $p_2 \to c_2$
6:     Randomly choose $1 \leq i \leq |X| + |Y|$ and $1 \leq j \leq |Y|$
7:     **for** $m = i$ to $|X| + |Y|$ **do**
8:         **for** $n = j$ to $|Y|$ **do**
9:             $\triangleright$ Crossover a block from alternate parent
10:             $c_{1,(m,n)} \leftarrow p_{2,(m,n)}$ and $c_{2,(m,n)} \leftarrow p_{1,(m,n)}$
11:         **end for**
12:     **end for**

13:     $\triangleright$ There is a good chance c_1 and c_2 are invalid, so repair them
14:     $c_1 \leftarrow REPAIR(c_1)$
15:     $c_2 \leftarrow REPAIR(c_2)$
16:     **return** $\{c_1, c_2\}$
17: **end procedure**

---

and crossover operators, new chromosomes are formed from an existing population. The fitness of a chromosome is measured through a given domain function.

The CMTS-ready genetic algorithm, listed in Algorithm 15, works identically to existing GA implementations. The genetic chromosome represents an matrix-based assignment solution. The fitness of a chromosome is measured through a function of the utility value, $U(S)$. The general process of the standard algorithm does not change to be CMTS capable since the work is performed by defined genetic operators.

To begin, an initial population is required to seed the genetic process. Fortunately, the random search and task sequencer algorithms, respectively in Sections 4.6.1 and 4.6.2, provide quick, albeit not very good, solutions to seed the population. For some number of iterations or other termination criteria, members of the population, P, are subject to crossover and mutation operations to produce candidate offspring for the next population generation, P'.

74

The crossover operation, listed in Algorithm 16, selects two members of the population P (the parents), and creates two new members (the children), which each contain chromosomal information from the selected parents. Since the chromosomes represent matrix-based solutions, the crossover operation defines a 2D point in the matrix solution to define an upper section and a lower section. The operation creates the children by copying the upper portion of the first parent to the first child, the lower portion of the first parent to the second child, the upper portion of the second parent to the second child, and the lower portion of the second parent to the first child. This way, each child contains a portion of each parent.

Mutation, listed in Algorithm 18, is much simpler. One form of mutation is to add or remove an ability from a task coalition. Alternately, the sequence of tasks is changed by a rearrangement or change of parent assignments. Either method only requires a single digit flip in the matrix-based representation.

Unfortunately, the operations can generate invalid solutions due to constraint violations. Mutation causes potential coalition conflicts where a newly assigned ability causes an inhibitor violation or bound abilities to be present in overlapping tasks, or potential prerequisite errors caused by changed task sequences. Crossover operations can cause task duplication errors as well as any of the errors created through mutation. Since two members of P are independent assignment solutions, swapping portions of the schedule does not guarantee that the agents of a newly acquired sequence are free for assignment nor that tasks in the new sequence still require assignment. This causes task duplication errors such as a task being assigned more than once.

This is mitigated by incorporating a pessimistic genetic repair operator, defined in Algorithm 19, which chooses to repair errors by non-deterministically changing the offending assignment or sequence order. The repair function implements a four-step process to correct violations of the constraints in Table 3.1. First, any cycles that have formed are broken by randomly removing a link in the cycle. Then, missing prerequisites sequentially swap into the ancestry of any task with prerequisites that are

Algorithm 17 Task Sequence Parent-Child Swap Method

1: **procedure** SWAP(y, y', S)
Require: (C_y, y) is parent of $(C_{y'}, y')$
2:     Let S' be a copy of S
3:     ▷ Step 1: Clear all sequences
4:     Remove all sequence edges $(C_{y_i}, y_i) \rightarrow (C_{y_j}, y_j)$ in S'

5:     ▷ Step 2: Set child as parent
6:     Add edge $(C_{y'}, y') \rightarrow (C_y, y)$ to S'

7:     ▷ Step 3: Restore relationships not starting from y and y'
8:     **for all** edges $(C_{y_i}, y_i) \rightarrow (C_{y_j}, y_j)$ in S **do**
9:         **if** $y_i \neq y$ or y' and $y_j \neq y$ or y' **then**
10:            Add edge $(C_{y_i}, y_i) \rightarrow (C_{y_j}, y_j)$ in S'
11:        **end if**
12:    **end for**

13:    ▷ Step 4: Set parents of (C_y, y) to parent children of (C_y, y)
14:    **for all** $(C_{y_i}, y_i) \in parent((C_y, y))$ **do**
15:        **for all** $(C_{y_j}, y_j) \in child((C_y, y))$ **do**
16:            Add edge $(C_{y_i}, y_i) \rightarrow (C_{y_j}, y_j)$ in S'
17:        **end for**
18:    **end for**

19:    ▷ Step 5: Set parent (C_y, y) as parent of child $(C_{y'}, y')$'s children
20:    **for all** $(C_{y_j}, y_j) \in child((C_{y'}, y'))$ **do**
21:        Add edge $(C_y, y) \rightarrow (C_{y_j}, y_j)$ in S'
22:    **end for**
23:    **return** Solution S'
24: **end procedure**

not met. Third, it repairs inhibitor and binding errors in coalitions. Inhibitor errors are fixed by removing any agents that become inhibited. Binding errors are repaired by moving an assignment randomly into the descendancy of the overlapping assignment. Finally, every task that does not have its requirements met by the assigned coalition gains a larger coalition that can perform the task.

Unfortunately, some measures used to correct an invalidated constraint cause another invalidation. For example, moving an assignment to fix a binding error may cause a new cyclic error. The repair process is then selectively reapplied until it is

---
Algorithm 18 Genetic Algorithm Mutation Method

 1: **procedure** MUTATE(p)
 2:      Initialize p' to p
 3:      Choose task y in p' to. . .
 4:      **if** randomly choose to change coalition **then**
 5:           ▷ This mutation changes the coalition assignment of a random task
 6:           Option 1: Remove any coalition member that duplicates a requirement
 7:           Option 2: Add any available ability that duplicates a requirement
 8:      **else**
 9:           ▷ This mutation changes the order of tasks
10:           Option 1: Remove any child if y has children
11:           Option 2: Add any other task as a child
12:           Option 3: Swap a parent with a child
13:      **end if**

14:      ▷ There is a good chance p' is invalid, so repair it
15:      $p' \leftarrow REPAIR(p')$
16:      **return** p'
17: **end procedure**
---

unable to correct a solution. Then, the solution is rejected from the candidate pool for the next population generation.

A new population of offspring can be produced in a variety of ways [6]. This approach uses a fixed size population where the new population consists of highly fit members of the original population and genetically altered members of the top 75% of members. The bottom 25% of the prior generation are not allowed to contribute to future solutions, and the middle 50% contribute genetic building blocks only through genetic operators. The best suited members represent the next generation P' which becomes the standard population P at the end of each iteration.

This process iterates for some fixed number of iterations or until the population shows excessive stagnation. A stagnant population occurs when the values of the best and worst members are equal. When some number of iterations occurs with a stagnant population, the lower-valued half of the population is replaced by new randomly generated chromosomes. This forces exploration of the solution space. The algorithm is converged once stagnation is not effectively relieved by a new population.

Algorithm 19 Genetic Algorithm Repair Method

1: **procedure** REPAIR(p)
2:     Perform steps 1 - 4 for elements of member p
3:     ▷ Step 1: Break any sequence cycles/self-loops
4:     **while** exists $P = path((C_y, y), (C_y, y))$, any path from a task to itself **do**
5:         **if** $len(P) > 1$ **then**
6:             Choose $(C_{y_k}, y_k) \in P) \ni 2 \leq k \leq len(P)$
7:             Remove edge between $(C_{y_{k-1}}, y_{k-1})$ and (C_{y_k}, y_k)
8:         **else**
9:             Remove edge between (C_y, y) and itself
10:         **end if**
11:     **end while**

12:     ▷ Step 2: Ensure prerequisites are met
13:     **for all** tasks y **do**
14:         **while** $\exists path(\ldots, y, \ldots, y' \in P_y, \ldots)$ **do**
15:             Swap $(C_{y'}, y')$ sequentially with a preceding parent task using SWAP
16:         **end while**
17:     **end for**

18:     ▷ Step 3: Ensure no inhibiting or separated abilities
19:     **for all** coalitions $C_y \ni \exists x_i \in C_y \ni H_{x_i} \bigcap C_y \neq \emptyset$ **do**
20:         $C_y \leftarrow C_y \setminus H_{x_i} \bigcap C_y$
21:         ▷ This may need repair handled by step 4
22:     **end for**
23:     **while** Exists a separated bound ability concurrently performing task $C_{y'}$ **do**
24:         Move $(C_{y'}, y')$ as a child of (C_y, y) which contains the binding ability
25:         Break any loops that may be created with step 1
26:     **end while**

27:     ▷ Step 4: Ensure all task requirements are met
28:     **while** exists C_y that does not meet the requirements of y **do**
29:         Let A be the set of all abilities in the ancestor set of (C_y, y)
30:         **if** $A \neq \emptyset$ **then**
31:             ▷ Add every available ability that performs the missing requirement
32:             $C_y \leftarrow C_y \bigcup \{x \in A \ni x$ performs an unassigned requirement of $y\}$
33:         **else**
34:             ▷ Add every ability that performs the missing requirement and fix
35:             $C_y \leftarrow C_y \bigcup \{x \in X \ni x$ performs an unassigned requirement of $y\}$
36:             Repair bounded ability errors with step 3
37:         **end if**
38:     **end while**
39: **end procedure**

The drawbacks to using a GA are the complexity of the repair function and potentially slow convergence rates [89]. The continual repair of members when using a large population greatly increases the computational processing time required to iterate solutions. And, GAs sometime require several tens of thousands of iterations to approach the optimal Pareto front within a specific error bound. However, a key benefit of using GAs is that multiple correct solutions are available at any given moment. So, a supervised task assignment GA algorithm offers a choice of solutions at any time, allowing the GA to perform as an anytime algorithm. Also, the GA produces optimal solutions over infinite time, so there is little concern for divergence.

4.7.2 Particle Swarm Optimization. Particle swarm optimization (PSO) [61] provides another evolutionary-based solution-space search. PSO, like a GA, begins with a population of solutions and creates new solutions by modifying the existing set. However, the modifications are based on vector algebra or potential fields. Essentially, each particle represents an assignment solution which occupies a position in n-dimensional field. The particle is moved in this space by adding calculated vectors to the best seen overall solution position and best seen solution by the particle to the particle's own velocity in the field.

Ideally, each particle begins with a zero velocity component and uses its initial position to represent its best seen solution. The initial global best solution is determined by selecting the best ranked solution based on their utility value. Then the particle is moved by adding the weighted difference between the current position with the best seen particle position, the weighted difference between the current position and the global best position, and the current particle's velocity. Equation 4.6 formally shows the update equation for moving a particle A_k given the local best A^+ and global best A^*.

$$A'_k = A_k + v_k + c \cdot r_l \left(A^+ - A_k \right) + (1 - c) \cdot r_g \left(A^* - A_k \right) \tag{4.6}$$

Algorithm 20 PSO Algorithm

Require: CMTS problem (M, Y)
Require: A population size N
Require: An iteration maximum T
Ensure: A solution S^* that is optimal for (M, Y)

1: Initialize population P with randomly constructed solutions
2: Rank P such that $\forall p_i, p_j \in P_t, U(p_i)$ is better than $U(p_j)$
3: Let $S^* =$ the solution in P such that $U(S^*)$ is best
4: **for** $t = 0$ to T **do**
5:    **for all** member p in P **do**
6:       $\triangleright$ Move the particle by reducing the Hamming distance from p to S^* by 1
7:       Randomly select element in p different in value from same element in S^*
8:       **if** no elements differ **then**
9:          Set p to a random solution
10:       **else**
11:          Change that element in p to match the value in S^*
12:       **end if**
   $\triangleright$ Using the matrix form
13:       Let $\Delta = \{(i, j) \in p \ni p(i, j) \neq S^*(i, j)$
14:       **if** $\Delta = \emptyset$ **then**
15:          Set p to a random solution
16:       **else**
17:          Randomly select $(i, j) \in \Delta$
18:          Set $p(i, j) = S^*(i, j)$
19:       **end if**
20:    **end for**

21:    Rank P such that $\forall p_i, p_j \in P_t, U(p_i)$ is better than $U(p_j)$
22:    Let $S^* =$ the solution in P such that $U(S^*)$ is best
23: **end for**
24: **return** $S^* =$ a solution in P such that $U(S^*)$ is best

---

The difference between solutions is weighted by two parameters each, a cognitive parameter c and a random fluctuation parameter r. The cognitive parameter, $c \in [0, 1]$, allows a particle to be more influenced by the group or by itself. The higher the parameter, the more it prefers the group influence. Two random fluctuation parameters, r_l and r_g, provide for exploration in the local neighborhood of best seen solutions to find potential improvements.

Particles continue to move within the field until the majority are within a pre-defined radius of the global best solution. The global and local best solutions are reevaluated at each iteration using $U(S)$ and the ranking metric. Finally, the global best solution then represents the final solution for the problem.

A CMTS-ready PSO algorithm, Algorithm 20, operates similarly using the matrix-based solution representation. The matrix solution represents a high dimension binary field that particles occupy. However, there are some significant limitations due to the way relational constraints shape the search space. First, the velocity vector representing the difference in solution positions is not necessarily a binary matrix because negative entries are possible. Likewise, multiplication by real-valued weights c and r place the particle solution in real space versus binary. This would force constant corrections by rounding values to the nearest appropriate binary value. Unfortunately, the rounding process loses significant information that causes solutions to get "stuck" in positions that require dramatic velocity changes in order to move in the search space.

In order to alleviate this limiting effect, velocity for the binary matrix in this algorithm is represented by Hamming distance instead of Cartesian distance. A velocity of one indicates a change of one value in the particle's position. Thus, the distance from one particle to another particle is the number of different elements in the two solutions. As long as the Hamming distance between a particle's solution and the global solution is decreasing, the particle is considered to be moving towards the global solution. Likewise for the local best seen solution.

Since the only rational change in the solution has a necessary magnitude of one, the cognitive and random weights lose their meaning as scalar weights. The algorithm accommodates the concepts behind the weights by setting the cognitive parameter to zero and allowing particles to randomly select which element to change to reduce the Hamming distance to the global solution. With a cognitive parameter of zero, all

particles are directed to move only towards the global best solution and disregard their own best position.

This behavior is acceptable since, by simple trials, most velocity changes made by a particle create an invalid solution due to some type of constraint. Instead of attempting to repair these solutions, the particle is allowed to exist in invalid space since it is guaranteed to return to the valid solution space in subsequent moves. Though, this may not happen until the particle reaches the global best solution. Otherwise, the particle may find pockets of valid solution space to report new solution utility values.

Finally, the traditional PSO has the particle swarming around the global solution, but this is not acceptable for a constrained binary space. Instead, once a particle reaches the global best solution, it is reset to another position in the solution space, potentially invalid, similar to the way a particle fountain system works. This approach is consistent with other PSO implementations for different problems [50, 61, 127]. A final solution is provided after a maximum number of iterations are performed or when the global best solution has not changed for some number of iterations.

The PSO algorithm operates quickly using relatively few resources. Since it deals with a pool of solutions, it can provide solution quickly making it an anytime algorithm. However, its most significant drawback is managing particles stuck in invalid solution subspaces since problem constraints have a significant effect on the shape of the field for matrix-based solutions (Section 3.3). This effectively limits the amount of valid searching the PSO system provides, possibly making the algorithm equivalent to trial and error searching.

4.8 Alternative Approaches

The algorithms described in the previous sections represent a wide range of approaches to solving the multiagent task assignment problem. However, they are not a comprehensive set of approaches to solving the problem. This section presents a few

more approaches that share similar characteristics to the featured implementations, but not enough difference to warrant thorough testing.

Integer linear programming (ILP) [89], closely related to dynamic programming, is heavily used in the operations research field to solve task assignment problems. The techniques behind ILP for task assignment generally work similar to the class of quadratic assignment algorithms and use a binary matrix structure to indicate assignments. This approach is similar to methods used by particle swarm optimization, genetic algorithms and ant systems. Values are built up through an iterative process until feasible solutions emerge while minding different types of constraints. ILP, like its evolutionary counterparts, is not guaranteed to find the optimal solution. Further, it is resource conservative, but most implementations do not perform in the role of an anytime algorithm.

Alternately, a genetic programming (GP) [6] approach can be used since the native solutions are graphs. A narrow-width beam search algorithm seeds the initial population with solutions, and the GP applies genetic operations directly on the graph representation of a solution in a tree representation rather than matrix representation. There is less concern for invalid solutions since the genetic operators work within the context of the solution instead of an abstract representation. However, a repair function is still needed for a graph represention to correct some constraint-violating changes. However, designing graph-based genetic operations that adequately produce a variety of offspring is technically challenging.

Finally, simulated annealing (SA) [23] and tabu search (TS) [47] perform solution-space searches using the matrix-based representation of a solution similar to PSO. A SA or TS approach makes minor alterations to a solution similar to the genetic operators used for the GA to generate a new set of solutions. Based on the utility values of neighboring solutions, these searches accept the better solution as a new search point. TS would keep a set of search points versus SA, which only keeps the best. The tricky aspect about performing these searches is in clearly defining what is a neighbor. If

defined as any change in the assigned coalition or task sequencing, then there are a factorial number of neighbors as identified by breadth first search node expansion (Section 4.2.1). For a large problem, selecting the appropriate neighbors would need to be streamlined to make either SA or TS viable for solving CMTS problems. This does occur in PSO, which modifies its solution by one element, but it does not keep a list of potential neighboring solutions, nor does it select the modification that results in the most beneficial move since it operates in invalid solution space.

4.9 Solving Problems With Uncertain Information

One important approach to solving multiagent task assignment problems is to be able to handle uncertainty in the environment. A common approach in task assignment is to use a partially-observable Markov decision process (POMDP) model [110]. Some of the methods mentioned in Section 2.2 show how to apply the POMDP to the multiagent problem. Generally, the models only apply to either the coalition structure or the task sequencing, but not both. Since the CMTS representation encapsulates both coalition formation and task sequencing into one representation, a POMDP model for the CMTS problem or algorithms then handles uncertainty for both aspects of the problem.

The POMDP model, derived from [60], is defined as the tuple $(P_S, P_A, T, \Omega, R, O)$, for a finite set of states, P_S, and discrete actions, P_A, that transition from one state to another. T is a transition function defining the probability of moving to a state from some state based on a selected action represented as $T(s_{t+1}, a_t, s_t) = \Pr(s_{t+1}|a_t, s_t)$. Ω is a finite set of observations related to states in P_S where members of the observations set, $o \in \Omega$, are representative of states that have incomplete or noisy information. O defines the probability of having some observation based on the state generated by some action, represented as $O(s_{t+1}, a_t, o_{t+1}) = \Pr(o_{t+1}|a_t, s_{t+1})$. R defines a reward for being in some state after having taken an action.

Essential to the POMDP model, as related to the multiagent problem, are agent beliefs about the state that it is in. The belief states for a probability distribution

over the state space P_S are derived by calculating the probability that the agent is in a state given the history of observations $o \in O$ and actions $a \in P_A$ taken. The belief probability distribution defining $b_t(s_t)$ is given in Equation 4.7.

$$b_{t+1}(s_{t+1}) = \Pr(s_{t+1}|o_t, a_t, b_t) \rightarrow \frac{O(s_{t+1}, a_t, o_t) \sum_{s \in P_S} T(s, a_t, s_{t+1}) b_t(s)}{\Pr(o_t|a_t, b_t)} \qquad (4.7)$$

The POMDP model is applied to the CMTS problem by defining the finite set of states P_S as the set of all partial and complete solutions. This set is finite as it is bounded by the maximum upper bound (UB) given in Equation 3.6. The set of actions is represented by the addition of either one coalition member or setting one assignment as a child in the task sequence. These actions are represented by the set of all matrices having a Hamming distance of one from the empty solution. Since the matrix representation of a solution is bound by the number of agent abilities and tasks, the number of matrices having a Hamming distance of one is finite. Precisely, there are $|Y|(|X|+|Y|)$ such matrices for a problem containing $|X|$ agent abilities and $|Y|$ tasks. The action set also includes the NULL action which is useful for indicating solution terminal points.

The transition probability T for CMTS defines the probability of reaching a valid (partial) solution from an action as a uniform distribution over the number of valid actions. The transition probability to invalid solutions is zero since it should not be considered, or even believed, to be reachable. So, define T by Equation 4.8.

$$T(s_t, a_t, s_{t+1}) = \begin{cases} \frac{1}{\#\text{ valid (partial) solutions}} & \text{if } s_{t+1} \text{ results in a valid solution} \\ 0 \text{ otherwise} \end{cases} \qquad (4.8)$$

The set of observations Ω can be defined in many ways. For uncertainty in the coalition set, the observation set includes only the portion of the coalition where an agent can contribute an ability over every possible task sequence. Complementary, for uncertainty in the task sequencing, the observation set includes every possible

coalition set for each task, but no task sequencing. The most encompassing approach is to use the intersection of these observation sets which is each agent accounting only for its abilities in a coalition and having no task sequence information. The last two definitions are somewhat problematic in an implementation since no task sequencing in a matrix representation can appear the same as the solution where all tasks are concurrently executed. The implementation for this research includes a special NULL task sequence to differentiate the representations.

The probability transition function, O, for CMTS problems then depends on which observation set is chosen. For the encompassing option, the transition function measures the probability of making an observation as the uniform probability distribution over the number of solutions containing a non-empty intersection between each assigned coalition and the abilities an agent has. For example, if an agent has two abilities, but only one is assigned in a coalition, then the probability of observing a solution with only the one ability assigned is the number of solutions that have the one ability assigned and the other not assigned over the total number of coalitions. Thus, the observation transition probability function, expressed in Equation 4.9, reduces to the number of solutions that contain the specific subcoalition offered by an agent $m \in M$ as the total number of coalitions made by other agent abilities X. The observation transition probability functions for uncertainty only in the coalition set or task sequence are similarly derived, but not used in this research.

$$O(s_{t+1}, a_t, o_{t+1}) = \Pr(o_{t+1}|a_t, s_t + 1) = \frac{1}{2^{|X|-|C_m|}} \qquad (4.9)$$

The reward for a POMDP solution to a CMTS problem is the utility value of the solution represented by the state if the solution is complete. If the state represents a partial solution, then the reward is zero. However, since it is possible to extend a complete solution into another complete solution, a non-zero reward is only offered when having selected the NULL action.

The final POMDP component for CMTS is the belief state. Modifying Equation 4.7 by replacing T and O with Equations 4.8 and 4.9 respectively derives the proper belief state probability. The factor in the denominator, $\Pr(o_t|a_t, b_t)$, is a normalization value used to ensure that the belief state remains a valid probability distribution over the entire state space. Using the encompassing observation set, this probability is defined as one over the number of actions that can change the observation, or $1/\left(2^{|Y||C_m|} - 1\right)$ for $|Y|$ tasks.

With these formulas, the POMDP version of a CMTS is solved using value iteration [60, 110]. The value function used for this research uses the policy-dynamics "Win or Learn Fast" (PDWoLF) [8] gradient descent based policy value iteration function, equivalent to Equation 4.3 described in Section 4.5.2. The value iteration converges to an optimal policy over infinite time based on stochastic action selection. The solution is the policy state that is maximal over all states.

However, the computational requirements of the model, in addition to maintaining a policy that grows asymptotically with the power set of the number of capabilities and tasks, can be overwhelming. Alternate approaches have managed to control the size of the policy using state space folding [41, 55] with experiments containing hundreds of millions of states to solve the coalition assignment component. When incorporating task sequencing, only very small problems contain less than 10^9 states which is roughly equivalent to a moderately constrained four-ability, six-task problem. Other techniques bound the computational complexity with correlation [10] or through other types of value iteration methods [99].

More importantly, the main benefit to using this approach is the incorporation of uncertainty in deriving solutions, under the Markov assumption. This is often the case in real world problems: agents may not know what the abilities of another agent are, or that actual order of tasks may not be resolved until the solution is completely assembled. But, this approach also fundamentally breaks the problem down into agent-centric views of the solving problem, as though the problem were decomposed

Table 4.1: Summary of CMTS-ready Algorithms.

Algorithm	Search Method	Iterates	Heuristic	Optimal	Resource Usage
Smart Random	construct	yes	no	no	$O(1)$
Sequencer	construct	no	no	no	$O(1)$
Serial Auction	construct	no	no	no	variable
Concurrent Auction	construct	no	no	no	variable
Breadth-first Search	construct	no	no	yes	variable
Depth-first Search	construct	no	no	yes	variable
A*	construct	no	yes	yes†	variable
Beam Search	construct	no	yes	no	fixed, $O(n)$
Value Beam Search	construct	no	yes	no	variable
Ant Colony Opt.	construct	yes	yes	no	fixed, $O(2^n)$
Reinforcement Learning	construct	yes	yes	no‡	fixed, $O(2^n)$
Genetic Algorithm	soln-space	yes	yes	no‡	fixed, $O(n)$
Particle Swarm Opt.	soln-space	yes	yes	no‡	fixed, $O(n)$
Value Iteration	soln-space	yes	yes	no‡	fixed, $O(2^n)$

†- Algorithm finds optimal solution with an admissible heuristic.
‡- Algorithm finds optimal solution with infinite time.

and solved individually by the agents. This hints that the CMTS problem, in general, can be solved using a distributed process.

4.10 Summary

The algorithms presented in this work, as shown in Figure 4.1, extend algorithms previously applied to the multiagent task assignment problem to use the CMTS problem description. The methods include generation of optimal solutions through the use of tree-based searches such as DFS and A*, or apply heuristic techniques to generate approximated solutions such as beam search, ACO, RL. Also included are economic-based auctioning methods and evolutionary approaches, including GA and PSO. The CMTS problem is also solvable by using a POMDP model when uncertainties are present in the problem domain.

Each algorithm provides a unique way of generating solutions either by con-struction of a solution one task at a time, or by using complete solutions to generate new solution points. The impact of the different search techniques used by these algo-

rithms only becomes apparent through testing. However, many of the algorithms are only applicable for small problems since their resource requirements are high. Table 4.1 summarizes the characteristics of these algorithms, including their resource usage. In general, the algorithms that find the better solutions tend to have the worst scalability. This leaves the ultimate trade-off as reducing solution quality in favor of solving larger problems. Unfortunately, the larger the problem, the lower the solution quality. The most appropriate way to leverage this trade-off is to use a distributed system.

V. Distributed Methods

To improve the efficiency of solving complex multiagent task assignment problems, the unified framework includes distributed processing approaches. The approaches are based upon applying one of two decomposition strategies to reduce the size of the solution space: solution space or problem space decomposition. Then, for either strategy are a number of techniques for managing the solution process for finding the best solution. Collectively, these processes demonstrate how to generate approximate and optimal solutions to CMTS problems with opportunities to increase efficiency using one or a combination of the algorithms from Chapter IV.

5.1 Decomposition Strategies

There are two decomposition strategies associated with CMTS problems. First, the finite-sized solution space associated with the problem is decomposed into granular subspace chunks with search algorithms applied to each subspace. This only increases the efficiency of the overall process if the subspaces are being searched concurrently. For properly sized subspaces and enough available processors, the increased efficiency can be significant. However, this strategy does not necessarily take advantage of the efficiency of the algorithms searching the subspaces. So it is possible that a heuristic algorithm could search the whole space more efficiently than several subspaces concurrently. Section 5.5 discusses the advantages to using this approach.

The alternate strategy is to decompose the CMTS problem space into smaller subproblems and solve those independently. This involves selecting subsets of agents and/or tasks to form a subproblem within the domain. Solutions to these subproblems are recombined through another algorithmic technique to generate an answer to the overall problem. There are a number of ways to perform problem-space decomposition and the associated recombination process as explained in the next couple of sections.
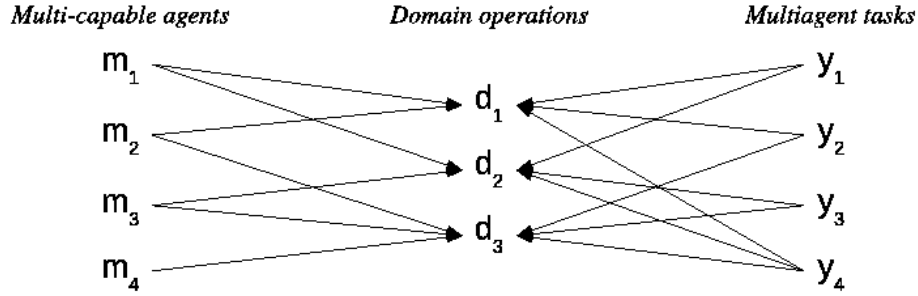
Figure 5.1: Problem Instance For Decomposition.

5.2 Problem Decomposition

The constrained multiagent task scheduling (CMTS) problem description provides several opportunities for decomposition. Since the problem is organized in terms of agents and tasks mapped to a set of domain operations, any one of these three elements forms a basis for decomposition. These breakdowns create a number of subproblems dependent on the number of elements in the set used to break down the problem. For example, a decomposition of a problem with five agents creates five subproblems; if there were only three agents, then three subproblems. Since the number of subproblems may not equal the number of distributed nodes in the cluster, a fourth type of decomposition is based on creating an equal number of subproblems as there are processors.

5.2.1 By Agent. The first approach to decomposing a CMTS problem is to create subproblems that single out the abilities of an agent. For every agent, a new subproblem is created where only that agent is available. Every task that requires one of the domain operations provided by an ability of the agent belongs in the subproblem task set. The resulting problem is based on a one-to-many agent-task decomposition.

The inherent problem with this decomposition is that possible agent inhibition or binding constraints are lost. To correct this, any other agent ability that performs any of the domain operations as the chosen agent are included. This potentially expands the problem back into a many-to-many agent-task subproblem where many

other agents perform the same operations. If every other agent performs at least one of the same operations, then the subproblem is not reduced. However, if no other agent performs any of the operations of the chosen agent, then the decomposition successfully reduces the problem space complexity of the overall problem by one agent.

The problem shown in Figure 5.1 decomposes into four subproblems, one for each agent. The problems before additional relational constraints are reintroduced:

$$[m_1] \times [y_1, y_2, y_3, y_4]$$
$$[m_2] \times [y_1, y_2, y_3, y_4]$$
$$[m_3] \times [y_1, y_2, y_3, y_4]$$
$$[m_4] \times [y_3, y_4]$$

5.2.2 By Task.   Alternately, the problem can be decomposed by task. This decomposition requires a new subproblem for each task resulting in many-to-one agent-task based subproblems. Every agent that has an ability that performs a requirement the task is included in the agent set. Since this is equivalent to coalition assignment (there is only one task so pick the best coalition), recomposing this problem is equivalent to the serial auction method.

The central issue with this decomposition is that possible task ordering constraints (prerequisites) are lost. This is corrected by including every prerequisite task in the task set, potentially expanding the problem to a many-to-many agent-task problem. Prerequisites of the prerequisites are not needed since resolution occurs with the recomposer. The only case where one of the subproblems matches the original problem is if the task has a prerequisite set containing every other task and requires every agent. Although there is only one location to place the task, in this case, at the end of the sequence, the algorithm designated to solve this problem is still faced with sequencing the remaining tasks.

Figure 5.1 also decomposes into four subproblems for the task decomposition method, one for each task. The problems before additional relational constraints are reintroduced:

$$[m_1, m_2, m_3] \times [y_1]$$

$$[m_1, m_2, m_3] \times [y_2]$$

$$[m_1, m_2, m_3, m_4] \times [y_3]$$

$$[m_1, m_2, m_3, m_4] \times [y_4]$$

5.2.3 By Operation. The third decomposition method divides a problem based on its domain operations. A new subproblem is created for each domain operation. Every task that requires an operation belongs in the task set, and every agent that has an ability that performs the operation also belongs in the agent set. This decomposition is more likely to create smaller many-to-many agent-task subproblems rather than isolate an agent or task. If optimally decomposed, this method isolates completely independent operations that can be concurrently performed.

Since the selection of agents is based on operations, the problem seen with agent decomposition is already managed for operation-based decomposition. However, there is still the opportunity to lose prerequisite information. The corrective action is to add the prerequisites of each task into the problem as prescribed by the task decomposition method.

Figure 5.1 only decomposes into three subproblems for the operation-based decomposition method, one for each operation. The problems before additional relational constraints are reintroduced:

$$d_1 : \quad [m_1, m_2] \times [y_1, y_2, y_4]$$

$$d_2 : \quad [m_1, m_3] \times [y_1, y_2, y_3, y_4]$$

$$d_3 : \quad [m_2, m_3, m_4] \times [y_3, y_4]$$

5.2.4 By System Size. The final approach for decomposing a problem involves decomposing both agents and tasks into subsets based on the number of processing nodes used by the distributed system. To create k subproblems (one for each of k processing nodes) from a problem with agent set M and task set Y, $|M|/k + \delta_M$ agents and $|Y|/k + \delta_Y$ tasks, δ_M, $\delta_Y \geq 0$, are sequentially selected for each subproblem. The δ bias controls how much overlap there is between problems. A bias greater than

one ensures that there is some overlap between subproblems which forces some task assignment decisions to be examined by more than one processing algorithm. This is useful for getting second or more "opinions" about an assignment since entities are not likely to appear in another subproblem if the bias is zero.

This decomposition produces a fixed-size number of many-to-many problems with overlap. However, it has all of the troubles from the other three decompositions: it can lose agent inhibition information, binding information, and task ordering constraints. To correct this, all prerequisites of tasks in the subproblem are added, and an agent ability that can perform a missing operation requirement is chosen at random to cover the task that requires the operation. After the additions, the resulting subproblems may grow back to original size. Such a situation primarily occurs when a task has a prerequisite set that contains all other tasks.

For a three- or four- node system with $\delta_M = \delta_Y = 2$, Figure 5.1 decomposes identically into three subproblems. The problems are $\{[m_1, m_2] \times [y_1, y_2, y_4]\}, \{[m_1, m_3] \times [y_1, y_2, y_3, y_4]\}$, and $\{[m_2, m_3, m_4] \times [y_3, y_4]\}$ before additional relational constraints are reintroduced. When extended to a five-node system, the problem decomposes into duplicate pairs of $\{[m_1, m_2] \times [y_1, y_2]\}$ and $\{[m_3, m_4] \times [y_3, y_4]\}$ subproblems. On the other hand, with $\delta_M = 2$ and $\delta_Y = 3$, the five-node system decomposes into a more interesting set of subproblems:

$$[m_1, m_2] \times [y_1, y_2, y_3]$$
$$[m_3, m_4] \times [y_4, y_1, y_2]$$
$$[m_1, m_2] \times [y_1, y_3, y_4]$$
$$[m_3, m_4] \times [y_2, y_3, y_4]$$
$$[m_1, m_2] \times [y_1, y_2, y_3]$$

Also note, that for a two-node system, the problem decomposes into duplicate pairs of the original problem.

5.3 Problem Recomposition

Once each of the distributed solvers generates a subsolution to their subproblem, the results are combined into a complete solution through a majority-takes-all approach. Three statistics are collected about the subsolutions in reference to the complete solution based on every coalition membership and task sequence entry of a matrix representation: the number of subsolutions that want the relationship (indicated by a one in the subsolution), the number of subsolutions that do not want the relationship (indicated by a zero in the subsolution), and the number of subsolutions that do not consider the relationship (indicated by the coalition ability or task not being in the subsolution).

Let W, NW and NC be integer matrices equivalent in structure to the complete solution respectively representing the number of subsolutions wanting, not wanting and not considering elements of the problem domain. The total number of votes across each (i,j) in the matrices must sum to the total number of subsolutions; i.e., for k subsolutions, $W(i,j) + NW(i,j) + NC(i,j) = k$.

The status of a relationship in the final solution is determined by the higher vote between W and NW. If the number of subsolutions wanting a relation is greater than the number not wanting a relationship, $W(i,j) > NW(i,j)$, then the relationship is established. Conversely, if $W(i,j) < NW(i,j)$ then the relationship is not established. But, if there are equal numbers of votes, then the relationship is in dispute.

There are many ways to resolve disputed relationships. One is to qualify each subsolution with a metric that can uniquely rank the subsolutions. One such metric is to use the utility value $U(A_i)$ of the subsolutions. This value is immediately available, but is heavily influenced by the decomposition of the problem. Another metric is to track how many votes were accepted in the non-disputed relationships. Yet, a better measure is to accept the result of a subsolution from a larger subproblem since that solution has likely searched a broader section of the solution space. The subsolution with the better metric is considered a better source to resolve the dispute, and

that subsolution's relationship is the accepted relationship for the complete problem. Random selection can be used to break further ties.

Alternately, the elements not involved in a dispute are frozen in place, and each of the distributed solvers re-accomplishes a subsolution for only the components that are in dispute. This effectively gives hints to the algorithms on how to build a better schedule with several additional assignment constraints put into place from frozen portions of a solution. The additional constraints significantly reduce the problem graph size making subsequent scheduling processes faster. And, the hints may help guide algorithms towards better assignment selection similar to a local search option.

The updated subsolutions are then submitted to the voting process to recombine a final solution. If disputes continue to arise, then all non-disputes in the new subsolutions are frozen and a new set of subsolutions is generated for the decreased set of disputed elements. This process continues until there are no more disputes, or that some relationships cannot be settled, in which case a random or informed selection settles the matter.

5.4 A Distributed Processing Meta-Heuristic

The process of generating a solution differs from traditional methods where distributed processes use a distributed form of a single algorithm to develop solutions. The method used here allows for any collection of CMTS algorithms to provide a scheduling solution to a distributed process. The approach still supports using traditional methods by having each distributed process use the same algorithm to develop solutions, but using multiple algorithms across the processes aims to exploit the characteristics that make the algorithms excel in generating solutions. For example, a narrow-width beam search with a random search minimizes memory consumption while finding solutions quickly with decent exploration of the search space. Then, the sub-solutions developed by each is combined to provide a single solution.

Algorithm 21 Distributed Scheduling Process

Require: CMTS problem (M, Y)
Ensure: S^* as the solution

1: Let P represent a set of subproblems to (M, Y)
2: Let S represent a set of subsolutions to problem (M, Y)
3: Let S^+ represent a complete solution to problem (M, Y), initially empty
4: Decompose problem (M, Y) into a set of subproblems $P = \{P_1, P_2, \ldots, P_n\}$ (Section 5.2)
5: **while** Solution S^+ is not completely locked or maximum iterations exceeded **do**
6:     **for all** subproblems $P_i \in P$ **do**
7:         Generate solution S_i by solving P_i locked by S^+ with any CMTS algorithm
8:     **end for**
9:     Recompose solution S^+ from solutions in S (Section 5.3)
10:     Optional: Score solutions S_i based on recomposition S^+
11:     Lock agreed elements of S^+
12: **end while**
13: **return** $S^* \leftarrow S^+$ as the final solution

Based on the methods used for decomposing and recomposing solutions, the distributed meta-heuristic, shown in Algorithm 21, is as follows. For a set of k solvers, decompose the CMTS problem P into k subproblems using any one of the four decomposition methods presented in Section 5.2, making adjustments as necessary. Issue subproblem P_i to solver S_i for a subsolution. In turn, each solver returns subsolution A_i. The subsolutions are recombined with the method described in Section 5.3 using the best-voting heuristic as the ranking strategy. Non-disputed elements are locked and the resulting solution is iteratively redistributed to the problem solvers until no further disputes appear. This basic approach generates an agreed, complete solution for problem P.

Furthermore, minor additions can be made to potentially improve the solution process. One improvement is to pass along a subsolution metric to the solver that generated it. By using the metric as a measure of comparable performance, underperforming algorithms reconfigure their parameters in order to generate better subsolutions for future problems. Alternately, the metrics of the entire group are shared so that algorithms can match parameters with better performing algorithms.

For example, if a value beam search of 50% is more agreeable than a value beam search of 20%, the latter search algorithm should increase its percentage parameter. This approach is used in weighted strategy sharing algorithms [1,2,24,34] for reinforcement learning while optimizing multiagent problems.

5.5   Distributed Methods For Generating Optimal Solutions

To get an optimal solution to a CMTS problem in a distributed system, the system must be able to evaluate every possible solution and compare the utility values of those solutions. The distributed meta-heuristic operates by generating a solution from decomposing the problem into subproblems. This approach does not necessarily reach every point of the solution space, so the distributed process is not guaranteed to provide an optimal solution.

In order to guarantee an optimal solution, a CMTS-ready distributed approach must instead partition the solution space, not the problem space. Since the solution space effectively is arranged on a two-dimensional discrete grid, any mechanism of partitioning the grid into k subspaces allows for parallel processing of the problem on k processors. Essentially, a processor evaluates every solution in a subspace and reports the best utility value to a master process. The master process then compares the best value of each process to determine which holds the optimal solution to the problem.

One partitioning scheme forms subspaces by assuming the first assignment in the solution and constructing the rest of a solution tree with a non-empty parent node. For instance, given a set of tasks Y, every solution will begin with some subset of these tasks. So, one approach is to send $2^{|Y|}-1$ subproblems to a processing cluster where each problem starts with a unique, fixed set of root assignments. An algorithm such as A* could finish the subsolution for each subproblem on each processor. Then, in the same manner as before, the processor with the best utility holds the optimal solution.

Alternately, from a matrix-based perspective, solution subspace partitions are formed based on selecting regions of the matrix, such as sets of rows, sets of columns, or submatrix "blocks". The rest of the solution matrix is fixed to some arrangement, and the identified submatrix areas are solved by concurrent processes. The assumed first assignment partitioning scheme, discussed in the previous paragraph, is equivalent to this approach where only a small region of the matrix is fixed. Otherwise, this approach potentially assumes several assignments before distribution. However, the fixed portion of the matrix still needs to be enumerated to ensure the optimal solution is discovered.

Unfortunately, these processes scale poorly since every solution must still be evaluated using a complete search. Furthermore, every processing node must have the entire problem context in order to find the optimal solution regardless of the partition scheme. At a minimum, for every agent and task in a processing node's subproblem, every other agent and task identified as a relational constraint must also be in the problem. However, if, for example, task ordering affects the overall utility value, partitioned problems become dependent. The optimal value of such a problem can be missed if separate clusters independently solved the partitions without evaluating the subsolution in the context of the entire problem.

5.6 Summary

Distributing the solution generation process for large CMTS problems increases the efficiency of solving the problem. The distributed meta-heuristic presented in this research uses any CMTS-ready algorithm to solve subproblems of a larger domain problem to capture the basic process for solving a problem with a distributed system. The basic process requires problem-space decomposition and recomposition methods that maintain the integrity of the problem constraints on solution form. Four different methods to perform decomposition are proposed along with subsolution recomposition methods to provide flexibility in applying a distributed solution.

CMTS problems are decomposed by agent, task, operation, or a fixed number of subproblems. Each method divides the problem space based on problem domain elements or the number of processing nodes. Depending on relational constraints, the methods produce smaller subproblems that have smaller search space sizes. However, when accounting for relational constraints, it is possible to not be able to decompose a problem. Another limitation is that this partitioning does not guarantee generation of optimal solutions since any one of the methods can remove an element from the problem that is required for an optimal solution. In order to guarantee finding an optimal solution using a distributed system, the solution space must be partitioned with each subspace completely searched. This approach searches sections of one solution space for one problem by distributed processes instead of searching overlapping sections of independent solution spaces for several subproblems. Furthermore, each processing node must evaluate solutions in the context of the entire problem in order to ensure relational constraints are properly handled.

Subsolutions are recombined into a single overall solution using a mediating voting process. Essentially, the more frequently an assignment appears in various overlapping subsolutions, the greater the chance of the assignment belonging to the final solution. Assignments in contention, where there are an equal number of votes to have it as there are to not have it, are resolved through tie-breaking techniques such as selecting the decision of the highest ranked subsolution or the most informed subsolution offering a vote. Alternately, the assignments are chosen through reiterating the disputed parts of a solution to the distributed system until there is complete agreement. This improves the overall solution by only using solutions from the best solvers, or by allowing solvers to adjust their search parameters to potentially improve solution development.

VI. Experimentation and Simulation

Three experiments demonstrate aspects of the effectiveness and efficiency of algorithms designed to solve constrained multiagent task scheduling (CMTS) problems. The first experiment involves a simple gathering problem domain with no constraints in order to comparatively test the effectiveness of taxonomy-driven approaches in solving multi-capable agent, multiagent task problems. Four auction algorithms based on the multi-robot task allocation taxonomy (MRTA) [42, 45] provide insight into the solution trade-offs of solving multiagent task assignment problems with single-tasked agents and multi-tasked agents.

The second set of experiments demonstrates the effectiveness and efficiency of some of the direct search algorithms, such as depth-first search (DFS) and A*, for an alarm handling domain [25]. This unconstrained domain involves multi-capable agents cooperatively handling spatially separated alarms and requires multiagent, multi-tasking solutions. CMTS tree-based approaches are measured for effectiveness as compared to optimal and efficiency when solved using distributed methods in this domain.

The third experiment employs lower-resource-using CMTS algorithms in the autonomous search and recovery (ASAR) domain. This domain, described in Section 3.4.5, presents several relational constraints while requiring multiagent, multi-tasking solutions. Algorithms capable of performing larger scale problems are measured to determine which methodologies provide effective approaches to solving higher complexity CMTS problems. These methods are also tested in a distributed process to measure processing efficiency and error rates when solving complex problems decomposed by the methods discussed in Section 5.2.

The assumptions that hold during experimentation are that the multiagent approach provides higher quality solutions than single-agent or single-task approaches, but that there should not be a single approximation algorithm identified in Chapter IV that consistently outperforms all other algorithms for every single problem. Such an algorithm would contradict the No Free Lunch Theorem [122], the widely accepted

position that no single approximation algorithm solves every problem better than all other algorithms.

6.1 Gathering Domain Experiments With Auction Methods

The value added in allowing multiple agents to perform multiagent tasks is realized through several experiments using a common solution method, auctioning. Auction methods solving each class of the multi-robot task allocation (MRTA) taxonomy [42,45] are compared based on solution quality to determine which taxonomy class algorithm provides the highest quality solutions. The experiment domain is simplified gathering, and it is represented using the constrained multiagent task scheduling (CMTS) problem descriptor.

6.1.1 Gathering Problem Domain. In gathering, there are mixed pools of resources that agents have to gather in minimum time. Each agent has one or more abilities that collects a resource type at some rate. For instance, an agent may be able to collect three red blocks and two green blocks at a time. A task in this problem is simply to have agents collect all of the resources in a given pool, e.g., agents must collect 15 red blocks, 10 blue blocks and 7 green blocks from some location. Distance between locations is not considered, only the total work rate. There are no constraints involved in these experiments in order to differentiate the effectiveness of the preferred many robots per multi-robot task, MT-MR, auctioning method versus ST-SR (one-to-one assignment), MT-SR (many tasks per robot, single-robot task assignment), and ST-MR (many robots per task, single-task robot assignment).

The CMTS description for this problem involves three independent operations: gather red blocks, gather blue blocks and gather green blocks. An agent has any number of abilities that performs one of these operations at a certain work rate r_x. This rate is the value function of the ability, $v_x(y, s) = r_x$ iff $d_x \in R_y$, otherwise zero. Tasks have a pool of size b_y blocks of one specific color. The requirement of a task,

R_y is only the one corresponding domain operation, e.g., $R_y = \{$ gather red blocks $\}$ for a pool of red blocks. The value of a task is the pool size, i.e., $v_y(C_y, s) = b_y$.

For cooperative gathering, agents work together to perform the task. The coalition combination function, f^C, is a summation of the work rates of each of the abilities in the coalition. Conversely, in competitive gathering, only a certain number of agents are allowed to perform any specific operation of a task. Extra abilities degrade the value of performing the task. The coalition combination function in this case changes to reflect that any number of abilities above a threshold $\xi > 0$ performing a specific operation causes a work rate of zero. The cooperative form assumes $\xi = \infty$. The value of an assignment $u((C_y, y), s)$, is then the total time for a coalition to perform a task. This is fully expressed in Equation 6.1 where the assignment combination function, f^A is $f^A \equiv \lceil \frac{v_y}{f^C(v_x)} \rceil$. The state provided s is the full or partial solution involving the assignment.

$$u((C_y, y), s) = \left\lceil \frac{v_y(C_y, s)}{f^C_{x \in C_y}(v_x(y, s))} \right\rceil = \begin{cases} \left\lceil \frac{b_y}{\sum_{x \in C_y} r_x} \right\rceil \text{ if } \exists d \in R_y \ni |\{x \in C_y | x = d\}| > \xi \\ 0 \text{ otherwise} \end{cases}$$

$$(6.1)$$

Then, the utility $U(S)$ for a set of assignments is the maximum of the sum of assignment utilities (performance times) over every possible path in the solution S. This is expressed in Equation 6.2.

$$U(S) = F^U((C_y, y) \in S) \equiv \text{argmax}\{\sum u((C_y, y), S) \in path(S) \forall \text{paths } \in S\} \quad (6.2)$$

This representation remains consistent for every instance of a gathering domain problem. Each of the auction methods used to solve an instance of the problem use the same set of equations. The only difference is the value of ξ for the cooperative and competitive versions of the domain.

6.1.2 Auction Methodology. Four different auction-style methods generate solutions to a cooperative gathering problem based on the MRTA class representation. In principle, the auction methods are equivalent to greedy algorithms. The basic approach is to form all possible assignments for a task and select the one that minimizes partial solution utility values until all tasks are assigned. So, the algorithms are at least assignment-complete searches since they test every possible assignment. This approach is outlined, for the MT-MR taxonomy class, by the concurrent auctioning algorithm in Section 4.4.2 using Algorithm 7.

The differences in algorithmic approaches are as follows. For the ST-SR and MT-SR auctions, only one-agent is allowed in a coalition. So modify Algorithm 7 so that only coalitions of one member are allowed in a task assignment. For ST-SR and ST-MR, ensure that assignments having coalitions with common agent abilities are serially ordered, similar to Algorithm 6.

The difference in instantaneous assignment (IA) versus time extended (TE) assignment depends on both the auction class and the problem. An ST class auction is TE if there are more tasks requiring an operation than there are abilities to perform that task. Otherwise, the auction is IA. The MT class auction is likely to always be instantaneous for this problem domain. This is because agents can work on all tasks concurrently in one step since there are no constraints forcing a separation. When constaints are introduced, the MT class auction divides into IA and TE based upon the the number of abilities that can perform a task given the constraints.

6.1.3 Experiments and Results. Fifty-six random instances of cooperative and competitive gather domain problems are used to quantify the effectiveness of the four auction approaches. Each instance represents a problem size of m agents and y tasks. Each agent is assigned one to three abilities that perform a unique domain operations with a rate $r_x \in [1, 5]$. Tasks randomly select one domain operation as its only requirement with a capacity $b_y \in [1, 20]$. The problem sizes range from two to six agents performing three to six tasks. There are no relational constraints for any of

Table 6.1: Gather Problem Instance Characteristics

M×Y	X	Size (LUB)	M×Y	X	Size (LUB)
2×3	4	$7.37 \cdot 10^{04}$	5×7	8	$1.07 \cdot 10^{29}$
2×4	2	$6.55 \cdot 10^{04}$	5×8	10	$3.68 \cdot 10^{40}$
2×5	4	$1.74 \cdot 10^{11}$	5×9	10	$2.48 \cdot 10^{48}$
2×6	2	$6.87 \cdot 10^{10}$	5×10	8	$3.94 \cdot 10^{50}$
2×7	3	$7.21 \cdot 10^{16}$	6×3	16	$1.64 \cdot 10^{16}$
2×8	4	$4.08 \cdot 10^{24}$	6×4	13	$1.47 \cdot 10^{19}$
2×9	5	$5.54 \cdot 10^{33}$	6×5	11	$2.55 \cdot 10^{22}$
2×10	5	$3.49 \cdot 10^{40}$	6×6	11	$4.69 \cdot 10^{28}$
3×3	7	$7.71 \cdot 10^{07}$	6×7	13	$7.65 \cdot 10^{39}$
3×4	5	$9.06 \cdot 10^{08}$	6×8	12	$3.41 \cdot 10^{45}$
3×5	6	$2.67 \cdot 10^{14}$	6×9	13	$4.84 \cdot 10^{56}$
3×6	5	$2.05 \cdot 10^{17}$	6×10	12	$8.63 \cdot 10^{62}$
3×7	5	$1.34 \cdot 10^{22}$	7×3	11	$4.23 \cdot 10^{11}$
3×8	6	$2.03 \cdot 10^{30}$	7×4	15	$4.16 \cdot 10^{21}$
3×9	6	$6.39 \cdot 10^{36}$	7×5	13	$2.80 \cdot 10^{25}$
3×10	8	$3.37 \cdot 10^{50}$	7×6	15	$1.10 \cdot 10^{36}$
4×3	9	$5.75 \cdot 10^{09}$	7×7	16	$1.95 \cdot 10^{46}$
4×4	7	$4.74 \cdot 10^{11}$	7×8	18	$1.42 \cdot 10^{60}$
4×5	6	$2.67 \cdot 10^{14}$	7×9	12	$8.57 \cdot 10^{53}$
4×6	8	$1.36 \cdot 10^{23}$	7×10	18	$1.62 \cdot 10^{81}$
4×7	5	$2.02 \cdot 10^{22}$	8×3	10	$4.93 \cdot 10^{10}$
4×8	8	$3.36 \cdot 10^{35}$	8×4	17	$1.14 \cdot 10^{24}$
4×9	10	$2.16 \cdot 10^{48}$	8×5	17	$3.69 \cdot 10^{31}$
4×10	11	$6.85 \cdot 10^{59}$	8×6	14	$1.72 \cdot 10^{34}$
5×3	11	$3.95 \cdot 10^{11}$	8×7	18	$3.35 \cdot 10^{50}$
5×4	10	$2.64 \cdot 10^{15}$	8×8	15	$7.43 \cdot 10^{52}$
5×5	8	$5.91 \cdot 10^{17}$	8×9	11	$1.27 \cdot 10^{51}$
5×6	11	$4.69 \cdot 10^{28}$	8×10	12	$8.63 \cdot 10^{62}$

the problem instances. Competition is set for maximum coalition per operation sizes of $\xi = 2$. Table 6.1 lists problem size characteristics for each of the problems used in the experiment. The complexity of each problem, χ, is zero.

Figures 6.1 and 6.2 show the rank-ordered solution quality, measured by averaged solution utility value, of each auction method for the cooperative and competitive problems. The better performing methods appear to the right on the bottom axis having an average lower utility value in a minimization problem. The results show
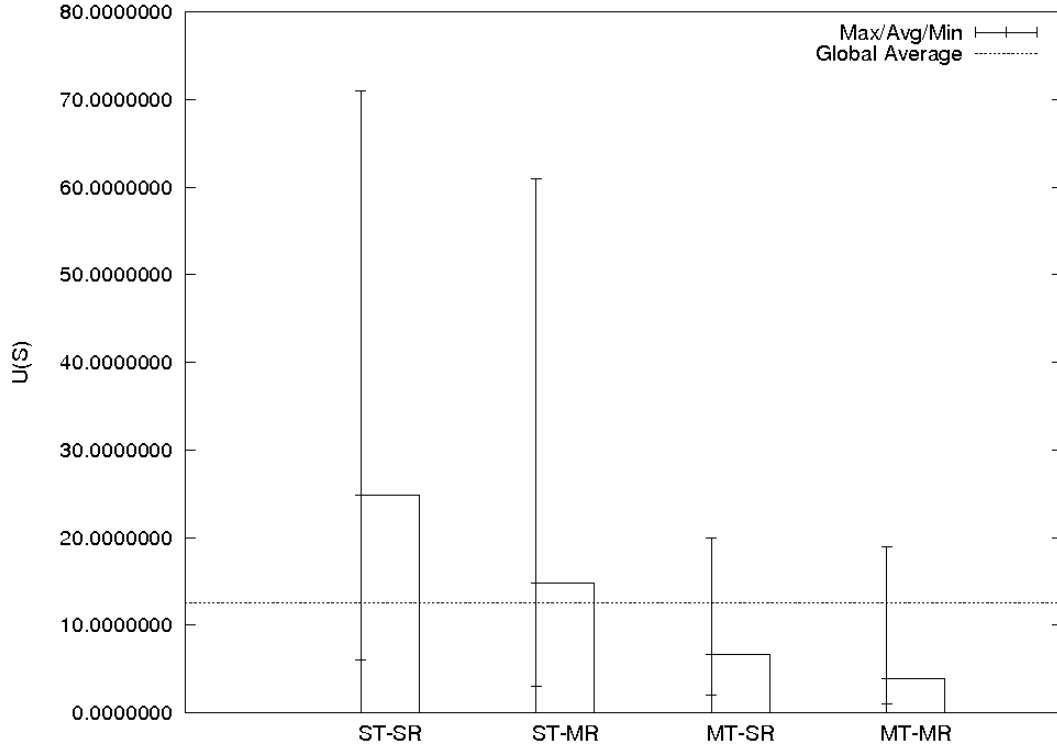
Figure 6.1: Average Utility Value Per Algorithm in Cooperative Gathering.

that the MT-MR auction method provides a significantly smaller utility value than the other algorithms. Statistical analysis using the Wilcoxen signed rank (WSR) [56] significance test show that the solutions from MT-MR are statistically significantly different from each of the other methods in both problem classes. The analysis shows that each algorithm has a less than 0.1% probability of being from the same population distribution from any other distribution.

Figures 6.3 and 6.4 illustrate the average error for each algorithm from the average of the best solutions found for each problem. The MT-MR algorithm has a total error percentage of 0% which implies that it found the best solution every time. The data indicates that ST-SR performs the worst, requiring over four times the problem solution time to perform a set of tasks than the MT-MR approach. The coalition-based methods, MT-SR and ST-MR, both provide better solutions than a one-to-one
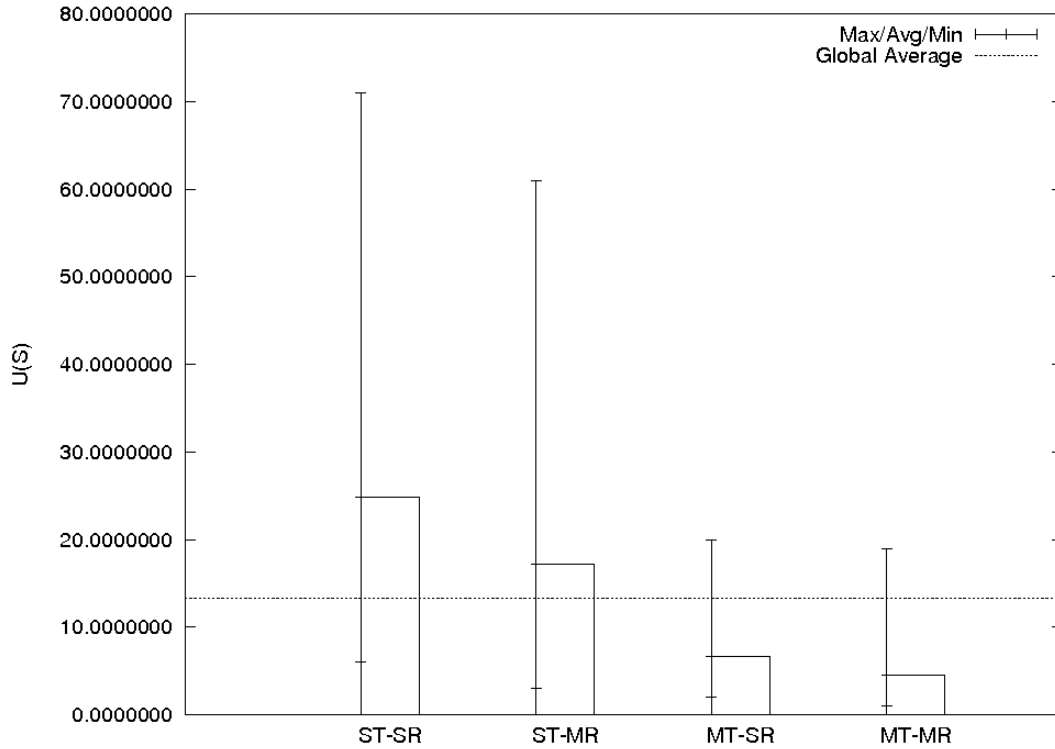
Figure 6.2: Average Utility Value Per Algorithm in Competitive Gathering.

approach, but underperform the concurrent auction by 71.0% and 278.0% respectively on cooperative performances, and 47.0% and 276.4% on competitive tasks.

As expected, the error margin decreases for competitive problems because the number of agent abilities per task is limited. This effectively eliminates potential multi-agent solutions which increases the minimum value of the MR-based auctions thereby making the SR auctions produce identical, yet closer results. The ST-MR approach has an average utility value elevated by 15.9%, and the MT-MR is elevated by 16.4% but still produces the best results.

These results suggest that using multiple agents is more effective for single-agent and multiple-agent tasks regardless of whether the the agents are cooperating or competing for gathering problem tasks. In both cooperative and competitive problems, the MT auction algorithms tend to generate completely instantaneous assignment solutions since agents perform all tasks at once. So, the multi-tasking agents provide

107
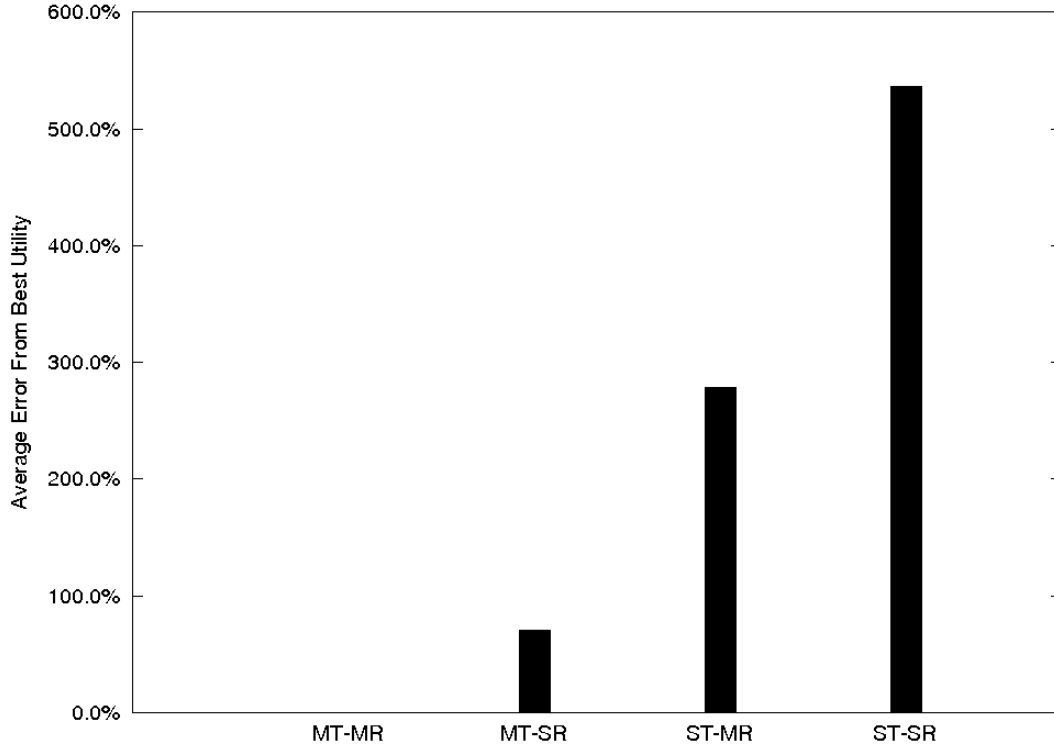
Figure 6.3: Average Error Per Algorithm in Cooperative Gathering.

better results than non-multi-tasking agents. The ST auction algorithms generate poorer solutions since they require planned sequences when there are more tasks than available agent operations for the requirements of the tasks. This results in longer solution times, thus less effective results.

6.2 Alarm Domain Experiments With Search Tree Methods

The second experiment builds upon the first by examining the alarm problem domain as a CMTS problem domain [25] where multiagent, multitasking solutions are required. The problem size for the experiments is kept relatively small to demonstrate the effectiveness of the search-tree methods. However, two experiments test 1) serial performance and three-node distributed processing using identical and mixed algorithmic solutions and 2) additional distributed methods using tree-based search methods and auction methods.
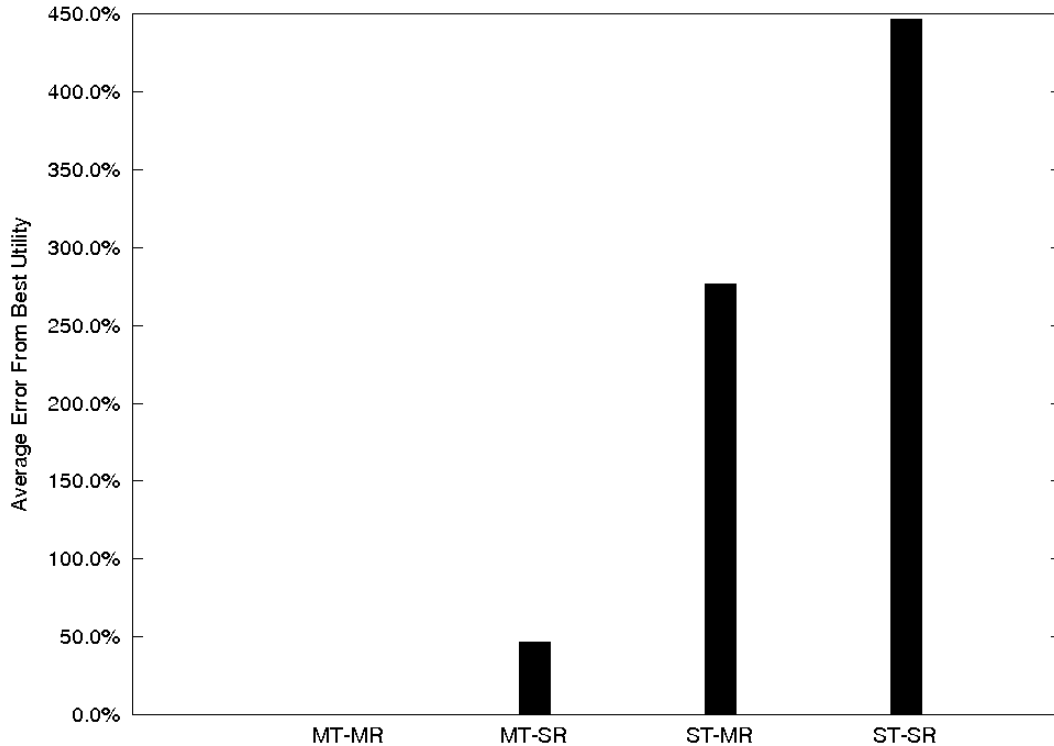
Figure 6.4: Average Error Per Algorithm in Competitive Gathering.

6.2.1 Alarm Handling Problem Domain. The multiagent problem domain used for experiment is a multi-capable agent, multi-agent task alarm response where alarms require one to three distinct domain operations to be deactivated. A problem instance involves a group of agents scattered across a two-dimensional area containing some number of activated alarms. Each alarm identifies a task requiring assistance from one or more agents. Depending on the number of alarms, the agents may be require to handle the alarms in stages rather than all at once. The goal is for the agents to efficiently service all of the alarms.

Solutions to this problem must optimize two objectives: minimize the total distance traveled to deactivate the alarms and minimize the number tasks any agent performs. The first objective implies that agents are to complete the problem as quickly as possible while the second encourages coalitions that balance the workload—although having one agent do it all is a valid solution, having several others sit

109

idle is generally not very operationally efficient. The second objective incorporates ideas similar to the behavioral methods identified in Section 2.2.4.

Using the CMTS problem descriptor, the agent ability value function, v_x, measures the distance of an agent to an alarm task from its last working position (or an initial position if it has not yet performed a task) for the first objective. This provides its portion of the total contribution towards a task, yielding the function $v_x(y, s) = (dist(x, y), \frac{1}{|C_y|})$, where C_y is the coalition performing task y. The coalition combination function sums both the distances required by the abilities to travel to the alarm task and the steps to the task, which comes to $(\sum_{x \in C_y} dist(x, y), 1)$. The task value function quantifies the objectives by denoting a distance value of zero and a step value equal to the depth of the assignment in the solution, one plus the number of tasks that precede it in the graph, yielding $v_y(C_y, s) = (0, depth(y))$. The state, s, used for these functions is the currently built schedule.

The assignment combination function merges the ability and task value functions as a sum over the agent distances and a product of task steps for each ability $\left[f^A = (\sum_{x \in C_y} dist(x, y) + 0, \sum_{x \in C_y} \frac{1}{|C_y|} \cdot depth(y)) \right]$, ultimately simplifying the assignment utility value to $u((C_y, y)) = \left(\sum_{x \in C_y} dist(x, y), depth(y) \right)$. The utility combination function combines the assignment utility values into the final utility value $U(S)$ by producing a single two-dimensional value by summing over each of the distances and taking the maximum of the task depths. The value is quantified into a single real value for ranking by taking the ratio of coalition balance to distance.

The heuristic chosen for the alarm problem domain is to choose the smallest distance to all remaining tasks and assume that all remaining tasks are completed within $\lceil |Y'|/|X| \rceil$ steps of a partial solution, where Y' is the set of unassigned tasks. Using these values to estimate the utility of an assignment consistently underestimates (at most equals) the true assignment utility value. Likewise, the heuristic is consistent since the distances are summed and the estimated number of remaining steps is monotonically decreasing. Thus the heuristic component to the prob-

lem yields $E[v_y(X, s)] = \left(argmin\{dist(x, y)\}, \lceil \frac{|Y - \{y \in A\}|}{|X|} \rceil\right)$, per the estimated utility value shown Equation 3.3.

6.2.2   Algorithm Selection.   In the first experiment, three serial algorithms provide the base algorithms for the distributed tests: A* (Section 4.3.1), beam search (Section 4.3.2) and random selection (Section 4.6.1). The A* algorithm is chosen because, for the given heuristic, it generates optimal solutions. Beam search performs similarly to A* (heuristically driven), but can only approximate solutions since it limits the number of potential solution candidates it evaluates. The experiments test beam search using pool sizes of 50, 100, 150, 200, 250 and 500 candidate solutions. The A* and the beam search algorithms use the same heuristic to determine which partial solutions are the most competitive for further expansion. Finally, the random selection provides a way to sample the quality of the solutions generated by the distributed approach. The DFS (Section 4.2.2) is also tested, but does not provide timely nor memory-efficient solutions to majority of the problem set. It is not distinctly reported in detail since results do not exist for it.

The distributed tests involve three mixtures of the serial algorithms over three processing nodes using decomposition by three nodes from Section 5.2. The first configuration has all three processors using the A* algorithm. Although the A* algorithm generates optimal solutions, the distributed algorithm using only A* solvers is not guaranteed to produce an optimal solution due to the fixed subproblem size decomposition method. The second configuration generates solutions with three processors using the beam search algorithm set to a pool size of 200. The final distributed test mixes the algorithms with one processor running A* and the other two running beam search with 200 candidates. The distributed portion of this experiment shows how efficient each algorithm is in searching a reduced solution space using common or different distributed solvers.

The second experiment expands the number and types of algorithms and distributed methods. The serial auction (Section 4.4.1) and concurrent auction (Section

111
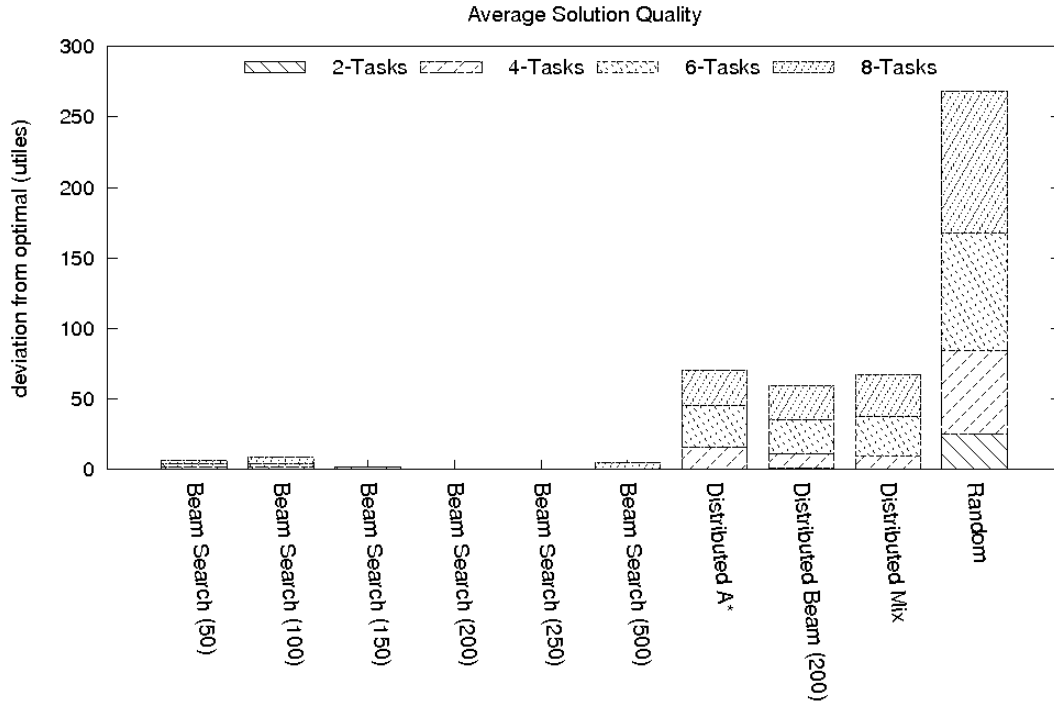
Figure 6.5: Comparison of Solution Quality for Alarm Handling.

4.4.2) are included and comparably tested with the A* results from the first experiment. Distribution is performed using each of the problem decomposition methods in Section 5.2 using matching algorithms. Decomposition by size is performed for four and five processing nodes versus three.

6.2.3 Experiment and Results. Each of the CMTS algorithms and distributed approaches are tested on alarm domain problem instances having two to five agents, with each agent having two or more abilities, collectively perform a problem consisting of 2, 4, 6, or 8 alarm tasks, many requiring more than one ability. The A* algorithm serves as the baseline for optimal solutions, and the random selection algorithm provides comparative sample quality.

Figures 6.5 and 6.6 plot head-to-head comparisons of the algorithms to gauge their effectiveness and efficiency in serial and distributed processing. The serial A* algorithm provided baseline optimal results for each of the problems except the eight-
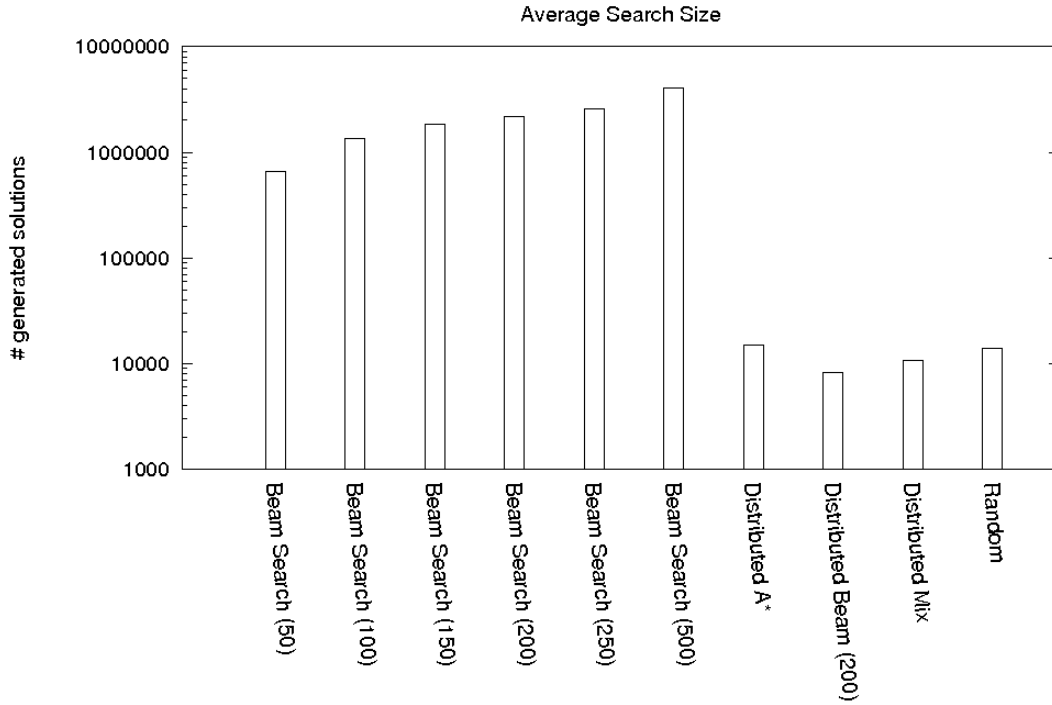
Figure 6.6: Comparison of Algorithm Performance for Alarm Handling.

task problems which it was unable to complete due to resource limitations. The results in Figure 6.5 show how the quality of solutions deviated in unit steps (metric distance of the distance and number of steps of agents in the problem domain) from the optimal results found by A* for 2, 4 and 6-task problems and the best solution found for the eight-task problems. As expected, the beam searches generally improve as the beam width grows, but the potentially negative effects of approximating the solution by limiting candidate pool sizes is apparent for a pool size of 500, where the search result has a higher error than smaller beams. This occurs when the algorithm is evaluating too many nodes with similar values early in the search and removes potentially good paths to explore a broader section of the upper tree.

The distributed algorithms offer an average 51% improvement over randomly generated solutions which brings some merit to the distributed approach in light of the straight-forward decomposition approach. However, each version of the distributed

113

test fails to outperform its serial counterpart. As expected, the distributed A* algorithm is non-optimal compared to serial A* because of the problem decomposition method. Interestingly, the distributed beam and distributed mix both provide less error to optimal solutions than the distributed A*. This is most likely due to the individual A* algorithms optimizing solutions to subproblems that do not have enough information to reach the global optimum. The recombination of the optimized subsolutions then fails to produce better solutions than approximation methods. Thus, using a distributed system with fixed-size decomposition is not as effective for solving the alarm problem.

On the other hand, the results in Figure 6.6 show the efficiency of the algorithms when solving the same problem set by identifying precisely the number of solutions generated for each algorithm. The beam search algorithms generate fewer solutions for the same set of problems as A* during serial solving, but distributed approach requires significantly fewer solutions to produce results that are within 20% of the optimal solution. The solution generation rates show that the distributed approach requires nearly two orders of magnitudes less generation than stand-alone algorithms. Specifically, the distributed beam search required less than 0.4% of the number of generations to find it solution; the distributed A*, at worst, needed less than 0.7%.

6.2.4   Additional Distributed Experiments.   Based on these results, additional solution methods and problem decomposition techniques are applied to the problems to test their effectiveness on smaller, unconstrained CMTS problems. Table 6.2 shows how many subproblems are created after applying the problem decomposition methods. A decomposition of one in the table indicates that the method is not able to provide a set of subproblems where each are smaller than the original problem. Only the agent and task based methods are able to decompose every problem after relational constraints resize the problem set. But, on average, task decomposition produced the most subproblems which contributes to making it the most efficient approach for the alarm problem domain.

114

Problem	Agent	4 nodes	5 nodes	Operation	Task	Average
002x002	2	1	1	2	2	1.6
002x004	2	4	2	1	4	2.9
002x006	2	2	2	3	6	3.6
002x008	2	2	5	3	8	4.7
003x002	3	1	1	1	2	2.5
003x004	3	4	2	3	4	3.7
003x006	3	2	2	3	6	3.9
003x008	3	2	5	3	8	5.0
004x002	4	1	1	3	2	3.2
004x004	4	4	4	3	4	4.4
004x006	4	3	3	3	6	4.7
004x008	4	4	5	3	8	5.7
005x002	5	3	4	3	2	4.5
005x004	5	4	5	3	4	5.1
005x006	5	4	5	3	6	5.6
005x008	5	4	5	3	8	6.1
Average	3.5	2.8	3.3	2.7	5.0	

Table 6.2: Number of Alarm Subproblems Generated Through Decomposition

Figure 6.7 shows the rank-ordered effectiveness of A*, beam search, serial auction and concurrent auction when used as a serial algorithm, which has no subscript, and when used in a distributed approach, showing a subscript. The subscript denotes which decomposition method applies: D_a for agent, D_t for task, D_o for operation, and $D_{f/n}$ for n number of nodes (four and five in this experiment). The better performing algorithms are ordered to the left where a lower average utility value best optimizes the minimization objective function, $U(S)$, as specified in Section 6.2.1.

The results indicate that the A* algorithm performs the best, as it should being the sole algorithm that generates optimal solutions. Likewise, four of the five distributed A* algorithms performed better than any other distributed approach although none of the distributed A* algorithms outperformed serial beam search. This is in line with the results discussed in the previous section. However, the best performing problem decomposition method is task-based decomposition.

Table 6.3 shows the averaged utility values and associated search space size relative to the algorithm and decomposition method, corresponding to Figure 6.7. The
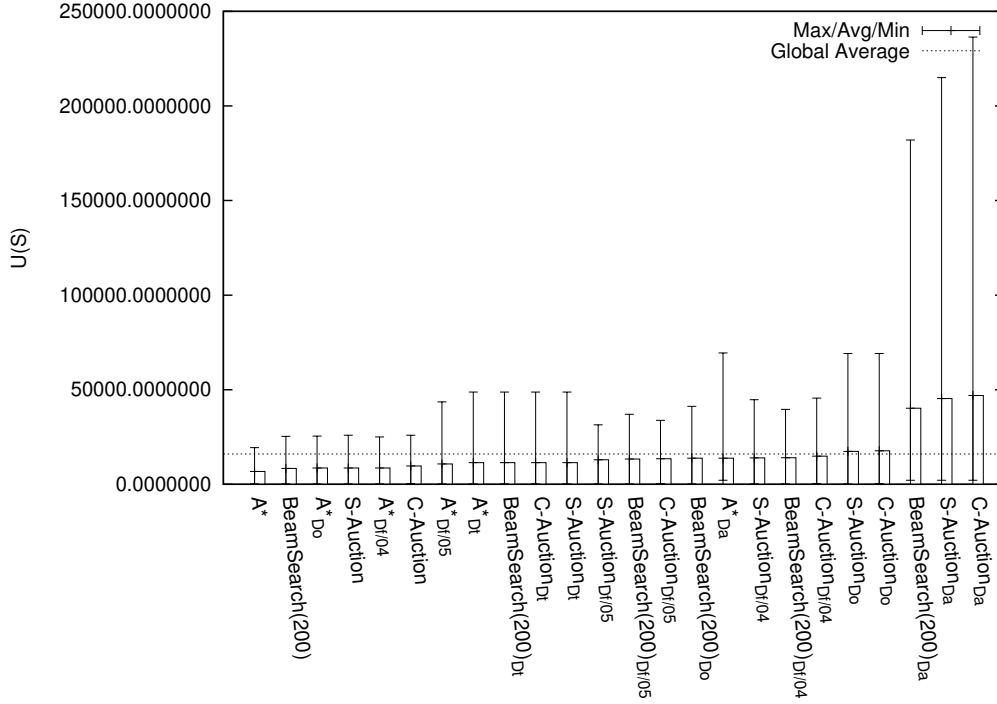
Figure 6.7: Average Utility Value Per Algorithm With Distributed Processing for Alarms, Ranked Best First.

best utility values for serial and distributed processing belong to the serial auction method, and the task decomposition method is identified as the best performing decomposition method except for A*. The last row of the table identifies the average size (LUB) of the solution space used to perform the search. Task decomposition uses the smallest search space of all of the methods by several orders of magnitude. This is key to efficiently solving much larger problems.

6.3 Autonomous Search and Recovery Domain Experiments

Additional algorithms described in Chapter IV are implemented and tested with autonomous search and recovery (ASAR) problem instances. The ASAR problems in this experiment also require multiagent, multi-task solutions, but involve varying

116

Table 6.3: Averaged Results for Utility Value and Space Size for Alarms.

		Decomposition Method				
Algorithm	Serial	Agent	4-Node	5-Node	Operation	Task
A*	6766.09	13822.08	8625.93	10729.21	8609.81	11454.61
Beam Search	8405.34	40192.86	14019.45	13387.29	13815.65	11454.6
S-Auction	8615.13	45337.19	13980.26	13008.27	17431.29	11454.6
C-Auction	9718.47	46969.39	14891.32	13544.55	17716.62	11454.6
LUB Size	4.6×10^{41}	1.6×10^{19}	3.0×10^{14}	5.1×10^9	9.3×10^{31}	7.9×10^2

degrees of relational constraints for significantly larger problems than in the alarm and gathering experiments. This experiment complements the alarms domain experiment by testing the algorithms for serial processing to measure effectiveness, then performing distributed processing to determine if the solution space savings provides quality solutions.

6.3.1 Autonomous Search and Recovery Problem Details. The ASAR problems used in these experiments are static instances of the domain problem where there are a known number of agents, tasks and domain entities. Gunmen and hostages compose the domain entities. A set of locations, to include a recovery point, provide points of reference for identifying the locations of agents, tasks and domain entities. Initially, all of the agents begin at the recovery point. Gunmen and hostages are randomly distributed at different locations. The positions of each of the entities, including agents, are globally known.

There are two tasks associated with the ASAR problem: secure and recover. Every location in the problem represents two tasks, one to secure a location and another to recover the location. Each recover task has the complementary secure task as a prerequisite. Coalitions are formed to complete each task. The goal is to recover all of the hostages to the recovery point as efficiently as possible. This is measured by minimizing the total time to perform the problem and maximizing the effect of each operation performed.

In order to secure a location, a coalition of agents must monitor the location and disable each of the gunmen. In order to monitor a location, at least one agent

must provide a *monitor* ability at the specified location. The effect of monitoring is quantified by a value of one if an agent with a monitor ability is present at the location, otherwise zero. To disable a gunman, agents must provide *disable* abilities at the specified location. The effect of disabling gunmen is measured by maximizing the probability of success of a group of disabling abilities such that if there are more disablers than gunmen, the probability is one, or if there are twice as many gunmen as shooters the probability is zero. The probability is the ratio of gunmen to disablers for all other events. This is mathematically expressed by the cases 0 if $n \geq 2x$, 1 if $n \leq x$, $\frac{n}{x}$ otherwise, where x is the number of disable abilities and n is the number of gunmen at the location. Essentially, the effect is maximized when there are at least an equal number of disabling abilities as there are gunmen, otherwise, the effect is linearly minimized.

The other task is to recover a location. To recover a location, a coalition of agents must monitor the location and move every hostage under observation to the recovery point. The monitoring of a location for a recovery task is the same for the secure task. However, moving a hostage also involves an agent with an ability to *observe*, as well as agents that can *transport*. The effect of transporting a hostage is measured by a simplified diminishing gains formula. To maximize the effect, two agent abilities are needed. Anything other than two represents a diminished effect. This is governed by the formula $\frac{x}{2n}$ if $0 < x \leq 2n$, else $\frac{n}{x}$.

In order to minimize the total time to complete the tasks, agents must minimize the total distance traveled to complete their tasks, assuming each agent travels at the same rate. When performing a *disable* ability, the distance is measured from the agent's last known location to the location of the task. The distance of *observe* operations are measured identically. However, the distance of the *transport* ability is the total distance of the agent from its current location to the task location plus the distance from the task location to the recovery point for each involved agent.

The problem prerequisites include task ordering such that each location is secured before recovered. Solutions involve locations to be secured then recovered iteratively, or to have all locations secured before any one is recovered. Both forms are acceptable solutions, though the optimal solution likely lies between these extremes. Also, having the abilities transport and disable together on an agent causes the two to inhibit each other. This implies that an agent cannot simultaneously disable a gunman while moving a hostage. Finally, every ability of an agent is bound to each other to ensure that parts of an agent are not separated to perform spatially separated tasks.

The components of the utility function $U(S)$ for the ASAR problem uses the distance and effect formulas to form the basis of the agent ability and task value functions, v_x and v_y respectively. More precisely, the agent ability value function is the distance the agent must travel in order to perform the task expressed as $v_x(y, s) = (0, distance(x, y))$ plus any travel to the recovery location if needed. The ability combination function f^C sums the distances that each member of the coalition moves, even if two abilities belong to the same agent. The task value function v_y presents the effect values of the coalition assigned to perform the task. The value of the task value function is the product of the operation effect functions associated with the entities. So, $v_y(C_y, s) = (eff_{disable} \cdot eff_{monitor} \cdot eff_{transport}, 0)$.

The value of an assignment, $u\left((C_y, y), s\right)$ combines the agent and task value functions through the assignment function combiner f^A. This is formally expressed as $u\left((C_y, y), s\right) \equiv f^A\left(f^C(v_x(y, s)), v_y(C_y, y)\right) = (\Pi_{d \in D} eff_d(C_y, y), \sum distance(x, y))$. Finally, the utility of the solution $U(S)$ combines the assignments with the assignment utility combination function F^U. For the ASAR problem, $F^U \equiv tan^{-1}(+, +)$, implying that the value of the solution is a ratio of the sum over all distances to perform tasks and the sum of all effect products. The higher the ratio, as provided by the inverse tangent, the better the solution maximizes the effect and minimizes the

time. Equation 6.3 states the equation in its entirety.

$$U(S) \cong tan^{-1}\left(\Pi_{d \in D} effect_d(C_y, y), \sum distance(x, y) \right) \qquad (6.3)$$

The expected utility for completing a task, $E[v_y]$ in the ASAR problem domain is the shortest distance of each agent capable of performing the task to that task, and assumes an efficiency rating of zero for any assigned coalition. This is expressed as $E[U(S)] = \left(0, \sum_{m \in M} min\{dist(x \in C_m, y)\} \forall y \notin S\right)$, using the form of the expected utility value shown Equation 3.3. Using these values to estimate the utility of an assignment consistently underestimates the true assignment utility value. This allows the expected value $E[U(S)]$ to be used as an admissible heuristic. The heuristic is consistent since the distances are summed and the estimated number of remaining steps is non-increasing. This enables appropriate algorithms to perform better searches, especially for the branch-and-bound techniques used in A* and beam search.

6.3.2 Problem and Algorithm Selection. The ASAR problem instances used in this experiment are randomly generated by the method described in Appendix B. The problems are generated on a 100x70 Cartesian space with all agents starting at a recovery point located at (0,-20). Tasks in the problem are randomly spread throughout a 100x50 Cartesian space with the x-axis covering the range $[-50, 50]$. Each of the ASAR problems used for preliminary and experimental testing are listed in Appendix B as an initial repository.

Only original ASAR problems are tested in these experiments. Benchmark problems from various repositories can provide problem domain instance data, but since the objectives differ between ASAR and other problems, optimal solutions provided for the library problems are not necessarily optimal for nor compatible with the ASAR problem. A separate series of tests could be run for other benchmark problems, but they do not exploit the complexity involved in a fully-defined CMTS problem and
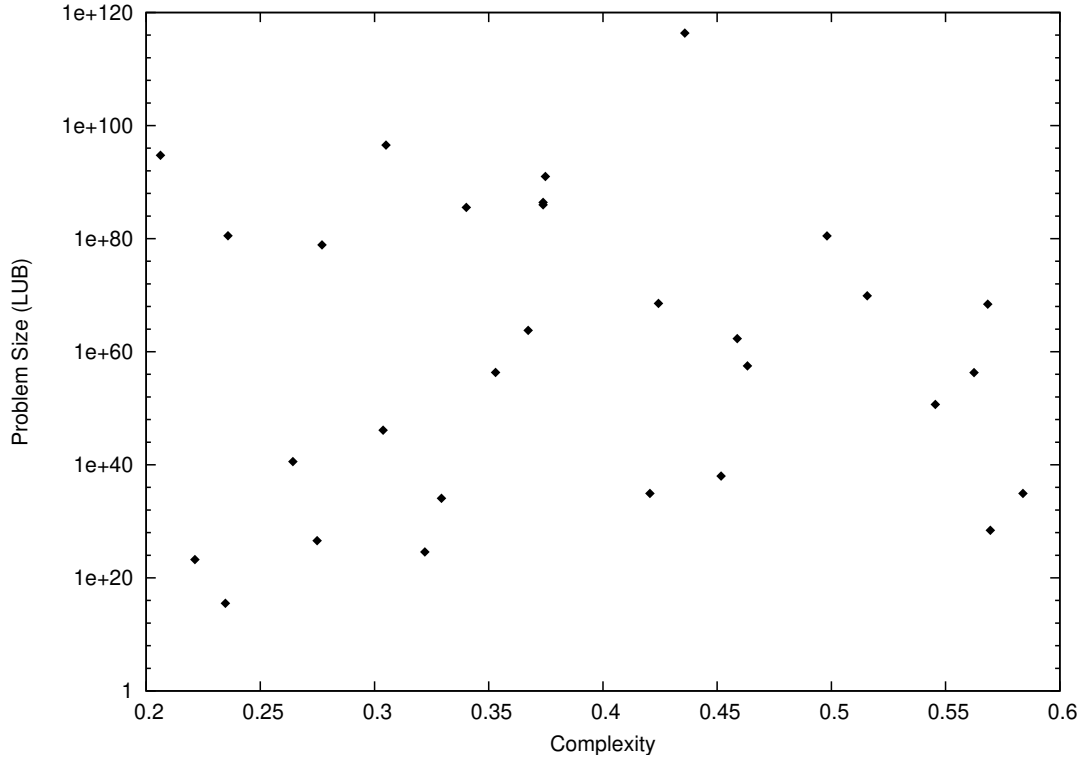
Figure 6.8: Experiment Problems—Size Mapped By Complexity.

Table 6.4: ASAR Problem Instance Characteristics.

M×Y	X	Size (LUB)	χ	M×Y	X	Size (LUB)	χ
3×6	6	$3.21 \cdot 10^{15}$	0.2347	4×12	10	$3.44 \cdot 10^{68}$	0.4243
3×7	9	$2.65 \cdot 10^{28}$	0.5696	4×13	11	$3.30 \cdot 10^{85}$	0.3402
3×8	9	$8.68 \cdot 10^{34}$	0.5838	4×14	9	$9.72 \cdot 10^{85}$	0.3738
3×9	7	$1.16 \cdot 10^{34}$	0.3293	4×15	9	$5.45 \cdot 10^{94}$	0.2063
3×10	8	$1.32 \cdot 10^{46}$	0.3037	5×6	11	$3.77 \cdot 10^{26}$	0.2749
3×11	9	$3.13 \cdot 10^{57}$	0.4633	5×7	7	$1.72 \cdot 10^{23}$	0.2213
3×12	8	$2.13 \cdot 10^{62}$	0.4588	5×8	9	$8.68 \cdot 10^{34}$	0.4206
3×13	10	$3.06 \cdot 10^{80}$	0.4980	5×9	14	$2.07 \cdot 10^{56}$	0.5625
3×14	9	$2.57 \cdot 10^{86}$	0.3738	5×10	11	$2.15 \cdot 10^{56}$	0.3530
3×15	6	$8.00 \cdot 10^{78}$	0.2770	5×11	11	$5.96 \cdot 10^{63}$	0.3673
4×6	10	$3.71 \cdot 10^{24}$	0.3220	5×12	15	$1.00 \cdot 10^{91}$	0.3748
4×7	13	$9.97 \cdot 10^{37}$	0.4517	5×13	10	$3.23 \cdot 10^{80}$	0.2359
4×8	11	$3.70 \cdot 10^{40}$	0.2643	5×14	11	$3.47 \cdot 10^{96}$	0.3050
4×9	12	$4.74 \cdot 10^{50}$	0.5455	5×15	13	$2.29 \cdot 10^{116}$	0.4358

the CMTS algorithms presented in this work do not exploit the heuristics involved in solving producing the known results.

121

Problem instances intentionally use a relatively low number of agents to emulate a team concept, but each agent can have up to four unique abilities. A team of five agents can potentially present a problem requiring the scheduling of 20 abilities which is a significantly increased number of entities to assign from the gathering and alarm domains. The number of tasks is kept at a minimum of six to ensure that only time extended solutions are generated in order to test the multiagent, multitasking, time-extended capabilities of CMTS-ready algorithms.

The purpose behind using randomly generated problems is to create problems with sufficiently uniformly distributed complexity over problem size. The size and complexity of the problems generated for this work is listed in Table 6.4. Figure 6.8 visually confirms the uniform distribution of problem sizes to complexity where each diamond shape represents a problem from Table 6.4. The generated set of problems from this experiment range in complexity from $\chi \in [.206, .584]$ which is well-centered in the ASAR complexity range of $[0, 0.75]$ derived in Section 3.4.5. Problems involve solution spaces larger than 10^{116} with the average of the lower upper bounds for the problems in 10^{59}. For comparison, chess is 10^{43} [106].

Thus, this experiment is designed to test the efficiency and effectiveness of low-resource algorithms from Chapter IV since the complexity and size of the ASAR problems are significant increases over the gathering and alarm problems. Although conceptually sound, algorithms such as breadth-first search, DFS and A* consume resources far too quickly to be a viable approach for even the smallest of problems. This is unfortunate since those algorithms provide optimal solutions that can be used to benchmark the quality of solutions generated by additionally evaluated CMTS-enabled algorithms. Ant colony optimization (ACO) and reinforcement learning (RL) also quickly consume resources and become bottlenecks too early when increasing problem sizes and cannot be reported in detail.

So, the experiment uses random search (using smart random), serial auctioning, concurrent auctioning, the genetic algorithm (GA), and particle swarm optimization

(PSO). The tree-based searches are not included for reasons listed above, but the MT-MR auction is included to provide a comparative analysis of how these other CMTS algorithms perform. The task sequencer algorithm, a random sequencer similar to ST-MR auctioning, is also not reported in detail since it, through preliminary testing, provided significantly worse results than any other algorithm in this test. Including the results of the task sequencer unfairly biases averaging techniques applied to comparative solution analysis.

It should be noted that algorithmic approaches other than the ones presented in Chapter IV either fail to handle the relational constraints expressed in CMTS without substantial modification, or completely fail to solve the problem. The latter case is evident when multiple agents are required to perform a task yet the algorithm only performs single-agent tasking such as the one-to-one auction method used in the gathering experiments. For this reason, only the serial, concurrent and MT-MR auction methods are used for the ASAR problem.

6.3.3 Serial Algorithm Comparison. Each of the algorithms are tested on several randomly-generated ASAR domain problem instances ranging in various sizes as identified in Table 6.4. The averaged results indicate the effectiveness of each algorithm for solving ASAR problems. The results of the average utility value, from Equation 6.3, for each algorithm are shown in Figure 6.9 where better performing algorithms are ranked to the left. The highest and lowest found values of any problem are plot on the line with a histogram box showing the average utility over all of the problems for each algorithm tested. Numbers next to an algorithm in parentheses indicate the number of iterations followed by the population size as applicable.

From the chart, the auction methods appear to have an advantage in finding better solutions over the other methods, although, the GA is capable of discovering higher-valued solutions than any other. The average solution quality of the algorithms form three groups based on significant statistical difference ($p < 0.05$). The serial auction is generating better results than any other algorithm as the top algorithm. The
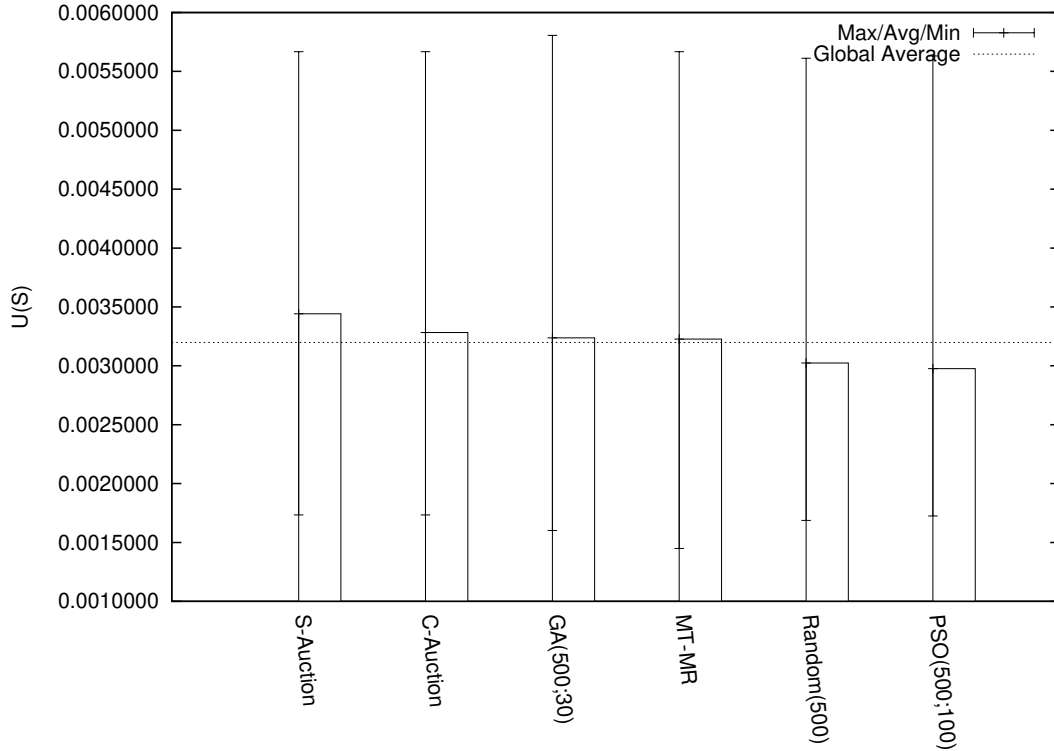
Figure 6.9: Average Utility Value Per Algorithm for ASAR, Ranked Best First.

GA, concurrent auction and MT-MR auction share similar results as the second tier group. The PSO and random algorithms trail all other algorithms, but share similar solution quality. Tables 6.7 and 6.8 shows the statistical differences and highlights in bold which solutions sets are statistically different using the WSR statistical test. From the utility figure and statistics table, it is clear that the PSO algorithm has difficulty generating solutions of the same quality of any other algorithm except smart random selection.

One of the reasons is because the PSO has a much slower convergence rate for finding better solutions since particles may iterate through large invalid solution subspaces. The rate of convergence for algorithms that perform iterative processes is shown in Figure 6.10 by annotating the value of the best held solution over time. The data shown is averaged over the total problem set for the GA, PSO and the random algorithms. Comparatively, the GA employs different types of repair methods to avoid
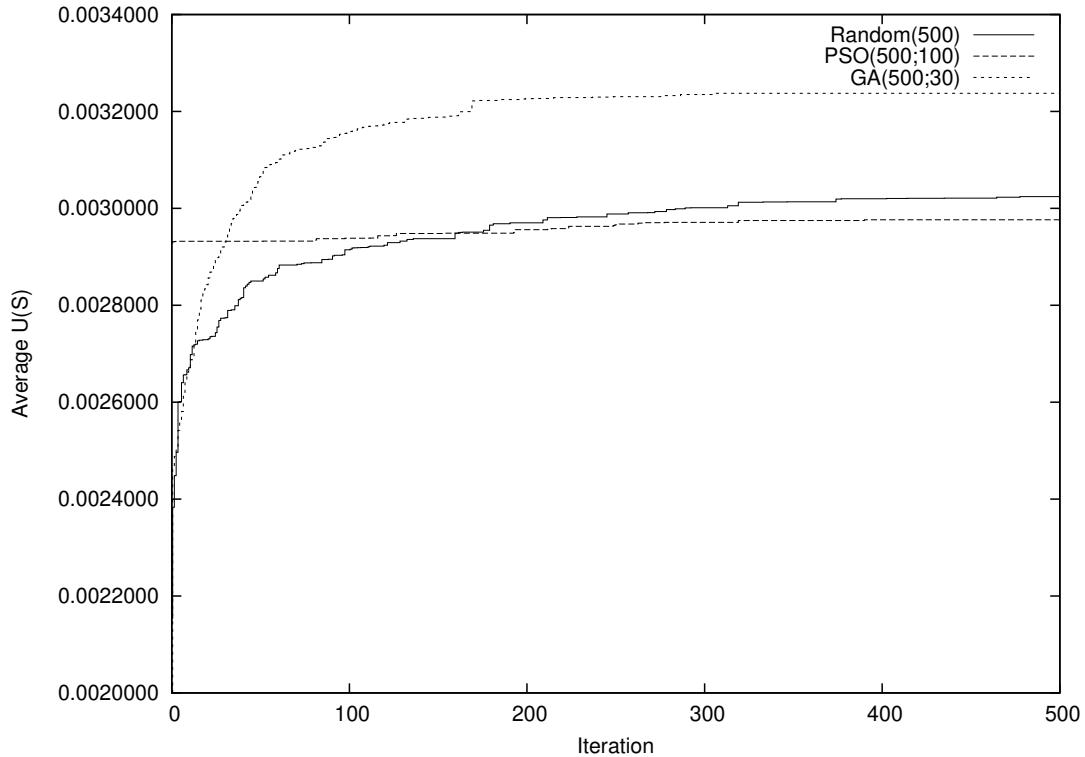
Figure 6.10: Convergence Results of Iterative Algorithms for ASAR.

consistently generating invalid solutions. The smart random algorithm only chooses from valid assignment sequences in order to avoid invalid solutions altogether.

While all three algorithms show iterative improvement over time, the GA shows the fastest convergence rate and quickly terminates. Figure 6.11 shows the convergence trends in the early iterative stages for each of the three iterative algorithms for a subset of ASAR problems over a longer span of iterations, up to 3,000. The GA and PSO have increasing population sizes to measure the effects of adding more solution points. As shown in the ASAR experiment solution set shown in Figure 6.10, the GA demonstrates the fastest convergence and finds better solutions with more members. However, the random algorithm does not appear to stagnate in these iterations, and shows a convergence rate that quickly overtakes various sized PSOs. With PSO showing incremental improvement throughout the iteration span, a faster converging random algorithm indicates that the choice of operators for the PSO algorithm does not explore the solution space very effectively or efficiently. Furthermore, since

125
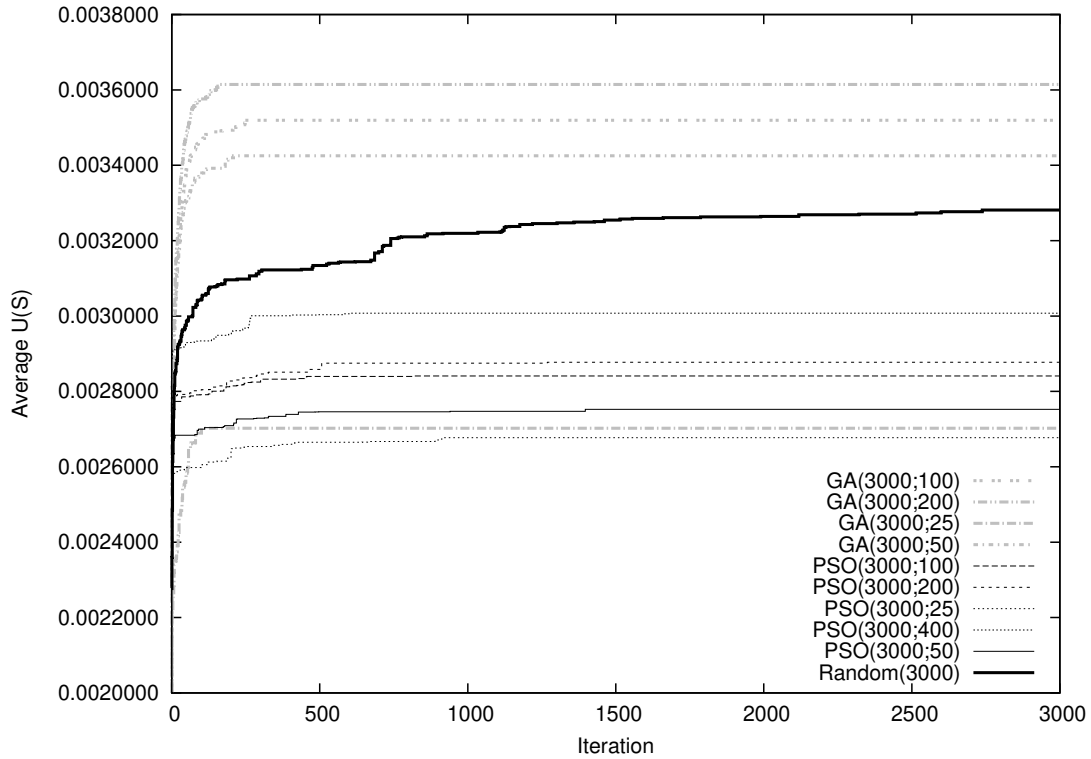
Figure 6.11: Iterated Algorithm Convergence Comparison.

each algorithm can reproduce identical results, duplicate solutions further degrade the ability to explore the solution space.

The serial algorithm experiment results support using the serial auction method for solving the ASAR problem. The serial auction is a variant of MT-MR which prefers to build longer sequences of assignments rather than short sequences. This result is consistent with the fact that the ASAR domain uses prerequisite task constraints which requires longer sequences. However, a GA can be used for off-line solution generation over long iteration periods since it is generally better at finding higher solution values than the other algorithms. The PSO, unfortunately, fails to excel over a random selection which implies that the approach outlined in Section 4.7.2 is possibly deficient for CMTS algorithms in general.

6.3.4 Distributed Analysis.   The same set of problems in Table 6.4 are tested using a distribution process with identical serial algorithms. The problems are

Problem	Agent	4-Node	5-Node	Operation	Task	Average
3×6	1	2	2	1	6	2.4
3×7	1	4	5	4	7	4.7
3×8	1	2	5	1	8	4.3
3×9	1	4	3	1	9	4.5
3×10	1	4	5	3	10	5.5
3×11	1	4	5	1	11	5.5
3×12	1	4	3	1	12	5.3
3×13	1	4	5	1	13	5.9
3×14	1	4	5	1	14	6.2
3×15	1	3	3	1	15	5.8
4×6	1	2	2	1	6	3.6
4×7	1	4	5	1	7	4.3
4×8	1	2	5	4	8	4.9
4×9	1	4	3	4	9	5.2
4×10	1	4	5	1	10	5.2
4×11	1	4	5	4	11	6.0
4×12	1	4	3	1	12	5.4
4×13	1	4	5	4	13	6.5
4×14	1	4	5	1	14	6.3
4×15	3	3	5	3	15	7.1
5×6	1	4	5	4	6	5.4
5×7	1	4	5	1	7	4.7
5×8	1	4	5	1	8	4.7
5×9	1	4	5	4	9	5.5
5×10	1	4	5	4	10	5.9
5×11	1	4	5	3	11	6.0
5×12	1	4	3	3	12	5.8
5×13	1	4	5	4	13	6.6
5×14	1	4	5	4	14	6.9
5×15	1	3	5	4	15	7.0
Average	1.1	3.6	4.4	2.4	10.5	—

Table 6.5: Number of ASAR Subproblems Generated Through Decomposition.

decomposed by each method discussed in Section 5.2. Table 6.5 shows how many subproblems are actually created after correcting the subproblems for relational constraints. A decomposition of one in the table indicates that the method is not able to provide a set of subproblems each of which are all smaller than the original problem.

Table 6.6: Averaged Results for Utility Value and Space Size for ASAR.

		Decomposition Method				
Algorithm	Serial	Agent	4-Node	5-Node	Operation	Task
C-Auction	0.003283	0.003269	0.003224	0.003086	0.002970	0.003235
GA(500;30)	0.003237	0.003233	0.003130	0.003095	0.002883	0.003251
MT-MR	0.003226	0.003212	0.003163	0.003028	0.002823	0.003241
PSO(500;100)	0.002977	0.002954	0.003067	0.003044	0.002734	0.002954
Random(500)	0.002977	—	—	—	—	—
S-Auction	0.003443	0.003428	0.003337	0.003215	0.003034	0.003254
LUB Size	7.6×10^{114}	7.6×10^{114}	7.0×10^{58}	7.7×10^{41}	3.6×10^{100}	3.7×10^{8}

From this data, it is clear that task decomposition provides the highest average number of subproblems and an optimal number of subproblems. Each task in a decomposed problem is placed in its own subproblem. The fixed node decompositions also generate an average number of subproblems close to their intended cluster size. Similarly, the operations-based decomposition provides an intended number of subproblems (based on the four operations of an ASAR problem) roughly one-half of the time. The agent decomposition, however, fails to provide a set of subproblems that does not include a problem equal to the original except in one case. In this case, the agent decomposition effectively nullifies the distributed approach by generating subproblems that are equal to the original, which forces at least one solution node to solve the problem serially.

Once the set of generated subproblems are solved, recombination merges the solutions, settling disputes by preferring the solutions of larger subproblems. No subsequent redistribution of partially locked solutions are performed in these experiments. The utility value results of the distributed process using the merged solutions are shown in Figure 6.12 according to how it was distributed. A distributed process is denoted by the algorithm involved in the distribution with a subscript 'D' followed by the decomposition method used—'a' for decomposition by agent, 'o' for operation, 't' for task, and 'f' for a fixed number of nodes with a number indicating exactly how many nodes. Serial tests are included for comparison (from Figure 6.9) with those algorithms not having a subscript. (Results specific to individual problems are listed in Appendix A.) The algorithms are ranked with the best performing algorithm, in
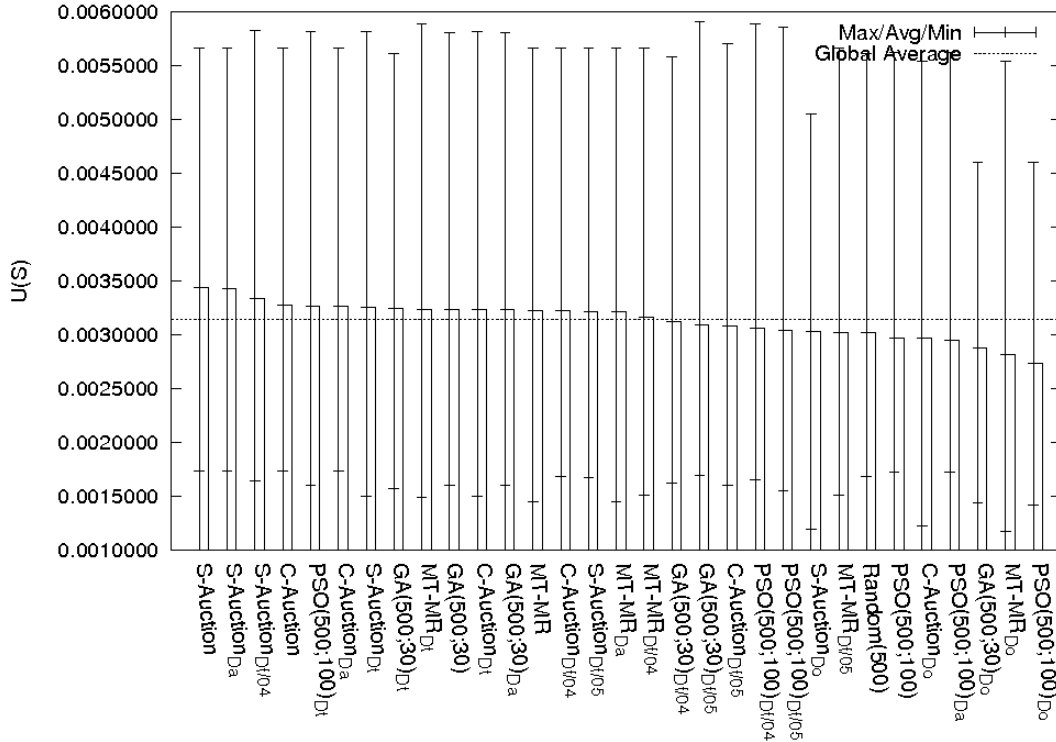
Figure 6.12: Average Utility Value Per Algorithm With Distributed Processing for
ASAR, Ranked Best First.

terms of average solution, appearing to the left. Each value is unique, so statistically
significant differences between algorithms implies a performance difference that makes
one algorithm "better" than the other.

Table 6.6 provides the actual average solution utility values corresponding to
Figure 6.12 for each algorithm broken down by decomposition method. The best
utility values for serial and distributed processing belong to the serial auction method,
and the task decomposition method is identified as the best performing decomposition
method. The last row of the table identifies the average size (LUB) of the solution
space used to perform the search, corresponding to Figure 6.13. As discovered in the
alarm domain, the best method is task decomposition with a relatively very small
search space.

In terms of solution quality, the decomposition methods show varied success in
general and per algorithm type. Disregarding agent-based decomposition, which per-

129

formed serial solutions, task-based decomposition provides the highest utility values for ASAR problems. Each of the five task decomposition approaches made the top one-third of ranked solutions with four of the five outperforming all serial performers except serial and concurrent auctioning. This demonstrates that the task-based decomposition is capable of improving the serial solution approach of the GA, MT-MR auction and PSO. In a very surprising result, task decomposition improved the performance of the PSO above the GA and MT-MR, and also the task decomposition approaches for serial and concurrent auction, ranking it fifth among all solution approaches. It is only one of two subproblem-producing decomposition that outperformed serial counterparts—the other being four-node decomposition for serial auction.

Otherwise, within each algorithm type, fixed node processing, a nearly perfect decomposition technique, provides mid-level solution quality with the four-node cluster size performing better than five nodes. Operation-based decomposition, the worst subproblem generating method, provides below average solution quality on every distributed test with four out of five tests performing worse than iterative random selection.

By algorithm type, the serial and concurrent algorithms hold six of the top seven ranked positions, with the serial algorithm appearing in four of those spots. This indicates that the auctioning methods are best at generating quality solutions in a distributed environment for the ASAR problem domain. However, the three highest scoring tests belong to fixed sized distributed approaches, the five-node GA, four-node PSO and five-node PSO. This shows that the decomposition/recomposition methodology is very capable of providing improved solutions to ASAR problems over a serial approach.

Statistically, however, the non-distributed serial auction approach ranks as the top algorithm and has a statistically significant difference from all other algorithms making it the preferred algorithm choice for this sample of ASAR problems. Compar-

Table 6.7: Right-tailed WSR Test Results (C-Auction - MT-MR).

Algorithm	$C-Auction$	$C-Auction_{Da}$	$C-Auction_{Df/04}$	$C-Auction_{Df/05}$	$C-Auction_{Do}$	$C-Auction_{Dt}$	$GA(500;30)$	$GA(500;30)_{Da}$	$GA(500;30)_{Df/04}$	$GA(500;30)_{Df/05}$	$GA(500;30)_{Do}$	$GA(500;30)_{Dt}$	$MT-MR$	$MT-MR_{Da}$	$MT-MR_{Df/04}$	$MT-MR_{Df/05}$
$C-Auction$	-	1.0	**.03**	**.00**	**.00**	.24	.18	.18	**.00**	**.00**	**.00**	.11	.22	.20	**.02**	**.00**
$C-Auction_{Da}$	1.0	-	.07	**.00**	**.00**	.35	.30	.24	**.01**	**.01**	**.00**	.26	.37	.22	**.05**	**.00**
$C-Auction_{Df/04}$	**.03**	.07	-	**.01**	**.05**	.73	.64	.64	**.01**	**.01**	**.00**	.95	.75	.69	.12	**.00**
$C-Auction_{Df/05}$	**.00**	**.00**	**.01**	-	.77	.11	.09	.10	.82	.61	**.02**	**.05**	.27	.29	.95	.09
$C-Auction_{Do}$	**.00**	**.00**	**.05**	.77	-	.15	.06	.07	.63	.70	.40	.07	.17	.21	.61	.70
$C-Auction_{Dt}$	.24	.35	.73	.11	.15	-	.99	.93	.11	.07	**.01**	.19	.58	.79	.72	**.00**
$GA(500;30)$	.18	.30	.64	.09	.06	.99	-	1.0	.06	**.02**	**.00**	.72	.61	.87	.29	**.00**
$GA(500;30)_{Da}$	.18	.24	.64	.10	.07	.93	1.0	-	.06	**.03**	**.00**	.63	.54	.86	.34	**.00**
$GA(500;30)_{Df/04}$	**.00**	**.01**	**.01**	.82	.63	.11	.06	.06	-	.44	**.03**	**.04**	.29	.33	.52	.25
$GA(500;30)_{Df/05}$	**.00**	**.01**	**.01**	.61	.70	.07	**.02**	**.03**	.44	-	.09	**.03**	.25	.33	.66	.15
$GA(500;30)_{Do}$	**.00**	**.00**	**.00**	**.02**	.40	**.01**	**.00**	**.00**	**.03**	.09	-	**.00**	**.01**	**.02**	**.02**	.50
$GA(500;30)_{Dt}$	.11	.26	.95	**.05**	.07	.19	.72	.63	**.04**	**.03**	**.00**	-	.58	.76	.26	**.00**
$MT-MRAuction$	.22	.37	.75	.27	.17	.58	.61	.54	.29	.25	**.01**	.58	-	1.0	**.00**	**.00**
$MT-MRAuction_{Da}$	.20	.22	.69	.29	.21	.79	.87	.86	.33	.33	**.02**	.76	1.0	-	.06	**.00**
$MT-MRAuction_{Df/04}$	**.02**	**.05**	.12	.95	.61	.72	.29	.34	.52	.66	**.03**	.26	1.0	.06	-	**.03**
$MT-MRAuction_{Df/05}$	**.00**	**.00**	**.00**	.09	.70	**.00**	**.00**	**.00**	.25	.15	.50	**.00**	**.02**	**.00**	**.03**	-
$MT-MRAuction_{Do}$	**.00**	**.00**	**.00**	**.00**	.10	**.00**	**.00**	**.00**	**.01**	**.00**	.32	**.00**	**.00**	**.00**	**.01**	**.03**
$MT-MRAuction_{Dt}$	.27	.39	.92	.08	.13	.51	.95	.90	.08	.07	**.01**	.16	.58	.82	.78	**.00**
$PSO(500;100)$	**.00**	**.00**	**.00**	.08	.84	**.00**	**.00**	**.00**	**.03**	**.05**	.31	**.00**	**.00**	**.01**	**.01**	.64
$PSO(500;100)_{Da}$	**.00**	**.00**	**.00**	**.04**	.64	**.00**	**.00**	**.00**	**.02**	**.03**	.54	**.00**	.06	**.00**	**.00**	.44
$PSO(500;100)_{Df/04}$	**.00**	**.00**	**.00**	.53	.81	**.05**	**.04**	**.05**	.25	.99	.16	**.01**	.06	.09	.36	.42
$PSO(500;100)_{Df/05}$	**.00**	**.00**	**.00**	.27	.92	**.01**	**.00**	**.00**	**.05**	.26	.35	**.00**	**.05**	.10	.06	.54
$PSO(500;100)_{Do}$	**.00**	**.00**	**.00**	**.00**	**.03**	**.00**	**.00**	**.00**	**.00**	**.00**	**.02**	**.00**	**.00**	**.00**	**.00**	**.00**
$PSO(500;100)_{Dt}$	.38	.56	.99	**.03**	**.05**	.10	.50	.45	**.04**	**.01**	**.00**	.95	.72	.99	.28	**.00**
$Random(500)$	**.00**	**.00**	**.00**	.21	.89	**.00**	**.00**	**.00**	**.00**	.08	.23	**.00**	**.00**	**.01**	**.01**	**.00**
$S-Auction$	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	.90
$S-Auction_{Da}$	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**	**.00**
$S-Auction_{Df/04}$	.35	.21	**.01**	**.00**	**.00**	**.00**	**.02**	**.02**	**.00**	**.00**	**.00**	**.02**	.16	.13	**.00**	**.00**
$S-Auction_{Df/05}$	.22	.32	.41	.51	.10	.92	.78	.90	.20	**.02**	**.00**	.92	.92	.75	.27	**.00**
$S-Auction_{Do}$	**.05**	.10	.31	**.05**	**.05**	.64	.49	.50	.95	.57	**.04**	.52	.44	.52	.86	.36
$S-Auction_{Dt}$	.40	.61	.90	**.05**	.09	.06	.70	.63	**.04**	**.03**	**.01**	.49	.86	.95	.43	**.00**

Bold values indicate statistically significant differences between solution values ($p \leq 0.05$).

Table 6.8: Right-tailed WSR Test Results (MT-MR - S-Auction).

Algorithm	$MT-MR_{Do}$	$MT-MR_{Dt}$	$PSO(500;100)$	$PSO(500;100)_{Da}$	$PSO(500;100)_{Df/04}$	$PSO(500;100)_{Df/05}$	$PSO(500;100)_{Do}$	$PSO(500;100)_{Dt}$	$Random(500)$	$S-Auction$	$S-Auction_{Da}$	$S-Auction_{Df/04}$	$S-Auction_{Df/05}$	$S-Auction_{Do}$	$S-Auction_{Dt}$
$C-Auction$	.00	.27	.00	.00	.00	.00	.00	.38	.00	.00	.00	.35	.22	.05	.40
$C-Auction_{Da}$	.00	.39	.00	.00	.00	.00	.00	.56	.00	.00	.00	.21	.32	.10	.61
$C-Auction_{Df/04}$	.00	.92	.00	.00	.00	.00	.00	.99	.00	.00	.00	**.01**	.41	.31	.90
$C-Auction_{Df/05}$	.00	.08	.08	**.04**	.53	.27	**.03**	**.03**	.21	.00	.00	**.00**	**.02**	.51	**.05**
$C-Auction_{Do}$	.10	.13	.84	.64	.81	.92	**.03**	.05	.89	.00	.00	.00	.10	.05	.09
$C-Auction_{Dt}$	.00	.51	.00	.00	**.05**	**.01**	.00	.10	.00	.00	.00	.00	.92	.64	.06
$GA(500;30)$	.00	.95	.00	.00	**.04**	.00	.00	.50	.00	.00	.00	**.02**	.78	.49	.70
$GA(500;30)_{Da}$	.00	.90	.00	.00	.05	.00	.00	.45	.00	.00	.00	**.02**	.90	.50	.63
$GA(500;30)_{Df/04}$	**.01**	.08	**.03**	**.02**	.25	.05	.00	**.03**	**.04**	.00	.00	.00	.20	.95	**.04**
$GA(500;30)_{Df/05}$	.00	.07	.05	**.03**	.99	.26	.00	**.01**	.08	.00	.00	.00	**.02**	.57	**.03**
$GA(500;30)_{Do}$	.32	**.01**	.31	.54	.16	.35	**.02**	**.00**	.23	.00	.00	.00	**.00**	**.04**	**.01**
$GA(500;30)_{Dt}$	.00	.16	.00	.00	**.01**	.00	.00	.95	.00	.00	.00	**.02**	.92	.52	.49
$MT-MRAuction$	.00	.58	.00	.00	.06	**.05**	.00	.72	.00	.00	.00	.16	.92	.44	.86
$MT-MRAuction_{Da}$	.00	.82	**.01**	.00	.09	.10	.00	.99	**.01**	.00	.00	.13	.75	.52	.95
$MT-MRAuction_{Df/04}$	**.01**	.78	**.01**	.00	.36	.06	.00	.28	**.01**	.00	.00	.00	.27	.86	.43
$MT-MRAuction_{Df/05}$	**.03**	**.00**	.64	.44	.42	.54	.00	**.00**	.90	.00	.00	.00	**.00**	.36	**.00**
$MT-MRAuction_{Do}$	-	**.00**	.16	.27	**.01**	**.01**	.25	**.00**	**.04**	.00	.00	**.01**	**.00**	**.01**	**.00**
$MT-MRAuction_{Dt}$	.00	-	.00	.00	**.03**	**.01**	.00	.15	.00	.00	.00	**.01**	.95	.61	.71
$PSO(500;100)$	.16	**.00**	-	1.0	**.03**	.24	.00	**.00**	.16	.00	.00	.00	**.00**	.45	**.00**
$PSO(500;100)_{Da}$	.27	**.00**	1.0	-	**.02**	.10	.00	**.00**	.05	.00	.00	.00	**.00**	.32	**.00**
$PSO(500;100)_{Df/04}$	**.01**	**.03**	**.03**	**.02**	-	.46	.00	**.00**	.05	.00	.00	.00	**.03**	.56	**.01**
$PSO(500;100)_{Df/05}$	**.01**	**.01**	.24	.10	.46	-	.00	**.00**	.53	.00	.00	.00	**.00**	.52	**.00**
$PSO(500;100)_{Do}$	.25	**.00**	.00	.00	.00	.00	-	.76	.00	.00	.00	**.03**	**.00**	**.01**	.25
$PSO(500;100)_{Dt}$	.00	.15	.00	.00	**.05**	.53	.00	-	.00	.00	.00	.00	.76	.56	.25
$Random(500)$	**.04**	**.00**	.16	.05	.05	.53	.00	.00	-	.00	1.0	.00	.00	.38	.00
$S-Auction$	.00	**.00**	.00	.00	.00	.00	.00	.00	1.0	-	.00	.00	.00	.00	.00
$S-Auction_{Da}$	.00	**.00**	.00	.00	.00	.00	.00	.00	.00	1.0	-	.00	.00	.00	.00
$S-Auction_{Df/04}$	.00	**.00**	.00	.00	.00	.00	.00	**.03**	.00	.00	.00	-	**.02**	.06	**.03**
$S-Auction_{Df/05}$	**.01**	.95	.45	.32	.56	.52	**.01**	.76	.38	.00	.00	.06	-	.66	.63
$S-Auction_{Do}$	.00	.61	.00	.00	.56	.52	**.01**	.56	.00	.00	.00	**.02**	.66	-	.44
$S-Auction_{Dt}$	.00	.71	.00	.00	.00	.76	.00	.25	.00	.00	.00	**.03**	.63	.44	-

Bold values indicate statistically significant differences between solution values ($p \leq 0.05$).
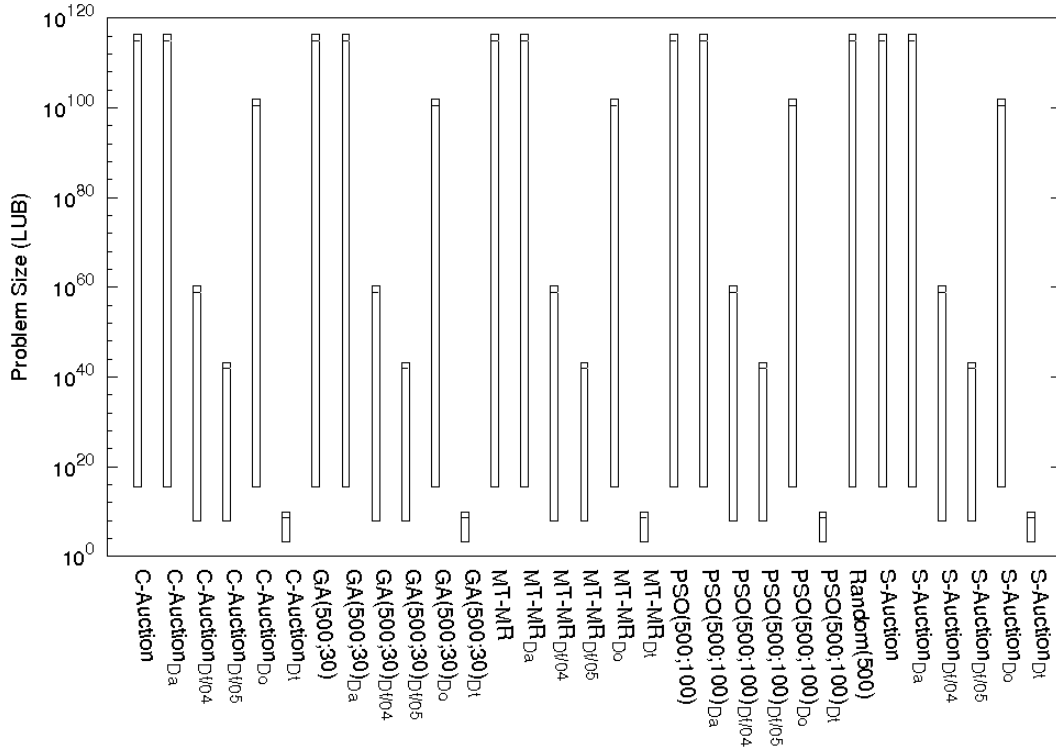
Figure 6.13: Average Solution Space Size Per Algorithm for ASAR.

isons of each of the algorithms is shown in Tables 6.7 and 6.8. These tables highlight in bold which solutions pairs are significantly statistically different using the WSR statistical test. The statistics confirm that the PSO and random approaches generally produce the least desirable results.

Although the effectiveness of the distributed approaches compares similarly with serial processing, the reduced search size of the distributed searches is more profound. Figure 6.13 shows the total size of the solution search space used by each of the solution approaches. The vertical bar indicates the range of solution space sizes from the smallest to the largest lower upper bound with a line in between indicating the average size. The non-distributed entries indicate the entire range of problems found in Table 6.4. Distributed entries indicate the range for the subproblems produced by the indicated decomposition method. Table 6.6 lists specific values for the decomposition methods as compared to the original problem solution space size.

The data clearly shows that distributing the problem using any decomposition method other than by agent substantially reduces the size of the solution space by several orders of magnitude over serial processing. There is a likely correlation between the success of the decomposition and the reduction in solution space size. The decomposition methods closer to an optimal decomposition reduces the size of the solution space the most. In this case, the task decomposition, which produced optimal decomposition, has the greatest reduction in solution space with an average reduction of 10^{106} (LUB) solutions. The other decomposition methods show, in sorted order, space reductions roughly equivalent to $0, 10^{14}, 10^{56}$, and 10^{73} for agent, operation, four-node and five-node decompositions.

Interestingly, task decomposition, the best utility-producing method, is also the best solution space reduction method. On average, it used less than $1.8 \times 10^{-13}\%$ (!) of the solution space to generate solutions. The other methods to not show such a significant size reduction. The worst subproblem-generating method, operation-based decomposition, only reduces the space by 50%. The five-node decomposition reduces the space more than four-node decomposition, but both are small fractions of the original space at 1.7×10^{-7} and 8.9×10^{-9}. However, the fixed-size decomposition methods appropriately generate smaller spaces for a larger number of processors as expected. Decomposition by agent only decreased the solution space in one specific case, otherwise that method searches 100% of the original solution space due to the increased relational constraints placed in the ASAR problem. But, overall, this implies that task decomposition provides the best search performance for solving an ASAR problem.

This is confirmed by measuring the average percent in utility error and solution space reduction for the test results. Figure 6.14 plots the average solution utility error for each algorithmic approach with a black bar. The lower the bar, the lower the difference percentage-wise from the best known utility solution value. From the data, it is clear that no one algorithm performs the best [122] with all algorithms showing an average error rate above 1%. However, the serial auction shows approaches

134
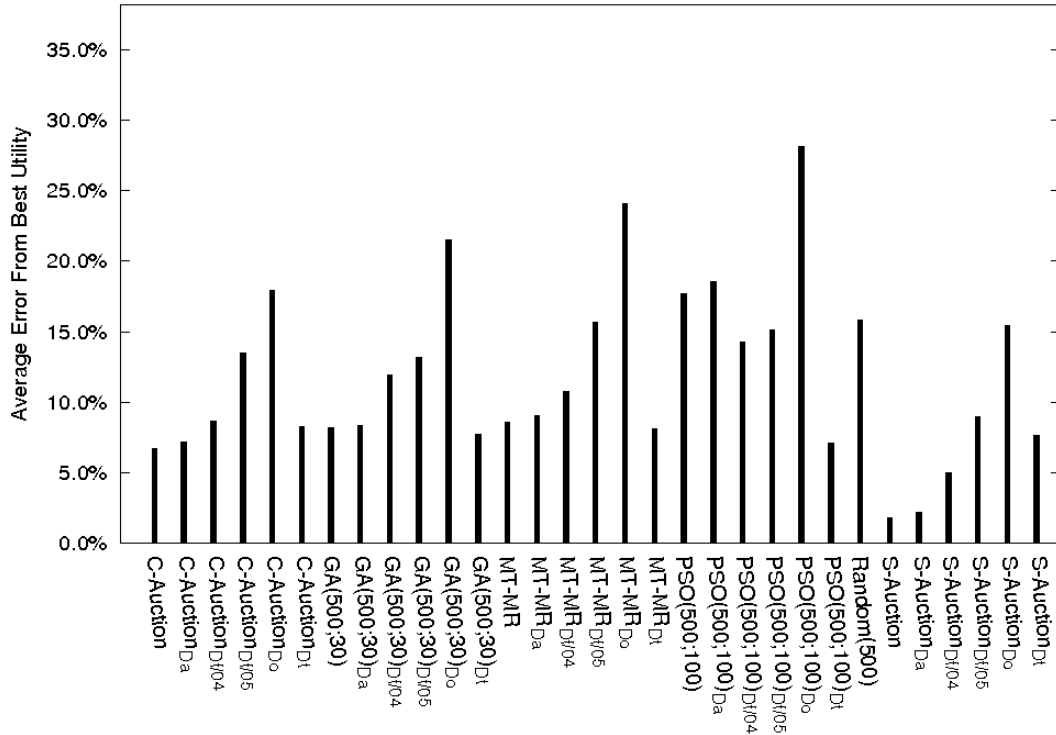
Figure 6.14: Average Utility Error Per Algorithm for ASAR.

with the least amount of error, averaging 6.83% from the best with a minimum error of 1.78% given by the serial solution approach. The concurrent auction, GA and MT-MR auction provide the next best average error rates in order. The PSO has an average error rate that is 0.97% higher than the smart random algorithm. On the other hand, the average error for task decomposition approaches, at 7.78%, is 0.81% lower than the average error for serial approaches at 8.59%. The next best decomposition method error exceeds 10%. Therefore, the task decomposition method for a distributed system provides less average error for the random ASAR problems used in this experiment.

Although there seems to be a correlation between the performance of the decomposition method and the reduction in solution space size, the data does not support any correlation between problem complexity and error margins. This suggests that the error in utility value of the distributed approach is independent of the complexity

in the problem even though decomposition is strongly related to problem complexity. This relationship is unexpected and requires further investigation to determine whether it is specific to the ASAR problem domain or if it generalizes over CMTS problems. The reduction in solution space is too constant (only six sample points on a line) to firmly conclude if there is a relationship between complexity and solution space reduction.

Thus, the distributed experiments demonstrate that the task decomposition method presents an approach that significantly increases efficiency by reducing the solution space search size while lowering the average utility error rate over the set of experimental algorithms for the ASAR problem. The effect is best noticed by the significant boost in performance given to the PSO algorithm, which moves from a last place serial method to fifth place of 31 distributed approaches. The task-based decomposition also boosted the performance of the GA and MT-MR auction approaches above their serial counterparts. However, the data continues to show that the serial auction method tends to perform the best on average in either a serial or distributed process, but with less confidence that they provide statistically significant solutions from all but operation-based decomposition using PSO and MT-MR. Though, the distributed forms of the evolutionary algorithms, GA and PSO, provide the highest possible solution quality for specific domain problems.

6.4 Behavioral Observations

Several interesting events occurred during the experiments. First, an overwhelming number of solutions generated throughout the ASAR problem set contained coalition assignments to tasks where not all members of the coalition could perform the task. Fortunately, this is not a design flaw, but rather the effect of using behavior-like methods for the agent and task value functions. The algorithms, based on the feedback of these methods, chose such solutions because they pre-staged some agents to a location close to a subsequent task in order to minimize total overall distance. Granted that there is a well-understood mathematical model behind the value of a

solution, pre-staging an agent could be considered a type of emergent behavior, such as those revealed in multi-robot behavior systems. Note that a pre-staged agent may adversely affect the assignment value of the coalition since the presence of additional abilities tends to lower the effect of performing a task.

Another type of behavior observed in the alarms and ASAR problem is coalition splitting and joining actions similar to animal flocking. Again, this behavior is encouraged by the underlying agent and task value functions, but becomes prevalent in solutions based on optimization of an objective function which is transformed to a single value. It appears that even without specific coalition-level balancing methods, the value functions of the agent abilities are sufficient to drive coalition-level effects.

6.5   Summary

Three distinct experiments are conducted to test the effectiveness and efficiency of multiagent task assignment algorithms modified to solve CMTS problems. In the first test, four auction-based methods implement MRTA taxonomy classes for a simple gathering domain problem. As expected, the MT-MR concurrent auctioning method outperforms the other taxonomy auction approaches for the cooperative gathering problem, offering solutions at least 31% more effective than any other approach. In these experiments, the concurrent auction always found the best solution. Although the auction approaches are assignment-complete searches, they are not guaranteed to find the optimal solution because they lack the benefits of scheduling found in other CMTS-ready algorithms defined in Chapter IV. The gathering problem domain used in these experiments does not often enough force the auction methods to form solutions with significant depth because the problems are not complex—each problem has complexity $\chi = 0$.

The second experiment involved applying tree-based searches, A* and beam search, to the alarm handling problem domain where multiagent solutions are required. This experiment tests the effectiveness of the tree-based search algorithm in reaching optimal values and the efficiency of those algorithms when using fixed

node size distributed processing. The A* algorithm provides optimal solutions for all domain instances with the beam search coming to within roughly 5% of the optimal solution. As compared to random selection, the tree-searches offer solutions averaging 50% more effective. The distributed approach, however, generates solutions that come within 20% but require less than 0.7% of the number of solutions to generate a result. This shows that the effectiveness of the distributed approach for fixed-sized distribution is not remarkably effective, but is highly efficient.

Secondary tests add two auction methods for a distributed-only series of trials using all four problem space decomposition methods. The results show that the A* algorithm produces better quality solutions than auctioning, which is competitive with the beam search. Task decomposition produced an optimal set of subproblems, showed the least error from serial processing than the other decoposition methods, and required the smallest search space. The decomposition methods averaged a reduction of 10^{26} solutions of a search space of size magnitude 10^{41}; task decomposition managed with a reduction of 10^{39} while producing solutions within 37% of the serial solution. However, like the gathering problems, the alarm problems do not have any relational constraints in order to use large-resource-using methods to solve small domain problems. Therefore, a better test of CMTS-ready algorithms requires a much more complex domain that includes constrained problems.

Thus, the third experiment shows the effectiveness of low-resource-using approximation algorithms, specifically, serial and concurrent auction with the MT-MR variant, the GA, and the PSO, with the complex ASAR problem domain. The problem instances are larger than the other domains and have varied complexity. Results show that the serial auction method produces the best results for both serial and distributed processing. However, task-based decomposition tends to improve the solution quality on average by just under one percent while using less than $2 \times 10^{-13}\%$ of the solution space. The average error for task-based decomposition for the complex problems stayed within 7.78% of the serial solution. This empirically shows that task

decomposition is a highly viable approach to increasing the effectiveness and efficiency of solving larger CMTS problems.

VII. Conclusions

This research presents a two-part unified framework that provides a novel approach to unifying the representation of and solving multiagent task assignment domain problems. The constrained multiagent task scheduling (CMTS) problem descriptor models complex multiagent task scheduling problems involving multi-capable agents concurrently performing tasks that can require multiple agents. The model is founded in multiagent task allocation and blends in concepts from resource constrained project scheduling, distributed constraint satisfaction and coalition formation.

7.1 The Unified Framework

The CMTS descriptor combines key elements of the multi-robot task allocation (MRTA) taxonomy (Section 2.1.1) with project scheduling (Section 2.1.2), distributed constraint satisfaction (Section 2.1.3) and coalition formation (Section 2.1.4), to produce a unified problem representation for multiagent task assignment problems. The expressiveness of the descriptor is shown by representing several classical and modern problems from combinatorial optimization in the same notation as defined in Section 3.1. The problems include foundational forms of the optimal assignment problem, flow and job shop scheduling, multiple traveling salesman, vehicle routing, and multi-object tracking, each defined in Section 3.2. Elements of these problems are combined to form the autonomous search and recovery (ASAR) problem as a demonstration of the level of complexity the CMTS problem expresses.

Solutions to the CMTS problem are generated by modifying existing algorithm bases and multiagent approaches to utilize the CMTS descriptor as prescribed in Chapter IV. Branch and bound techniques provide methods to generate optimal solutions but require a significant amount of resources to generate solutions. Alternately, approximation algorithms based on auctioning, policy learning and evolutionary approaches provide effective solutions without requiring significant resources. Furthermore, domain uncertainty is incorporated in the solution development process through partially observable Markov decision process models as shown in Section 4.9.

Algorithms modified to use the CMTS definition are able to solve the broad range of simple, unconstrained one-to-one agent-to task problems to highly complex, relationally constrained many-to-many problems as well as problems from the constraint satisfaction and resource scheduling domains.

To solve increasingly larger problems, distributed algorithms decompose large problems (Section 5.2) and recompose solutions (Section 5.3) from a set of subsolutions. The distribution uses multiple instances of the modified CMTS algorithms with one of four types of problem decompositions to generate approximated solutions. The distributed system also provides methods to generate optimal solutions based on solution-space decomposition rather than problem-space decomposition (Section 5.5).

7.2   Future Work

The algorithms presented in this framework modify different types of solution techniques used to solve multiagent task assignment to use the CMTS problem representation. Significant research, discussed in Chapter II, has formulated and demonstrated improvements to the search characteristics of the underlying algorithms in a variety of problem domains. That work can now apply to CMTS algorithms. The CMTS-ready implementations are provided for researchers to build upon, particularly for implementing the improved versions of these basic templates found in current literature, or by generating new algorithms by creating hybrids of these.

Although many algorithms generate competitive solutions, the solution space for a typical CMTS problem can be very complex based on the image in Figure 3.6. However, given the nature of some of the search techniques, is it possible to enumerate the coalitions and task sequences in such a way to minimize the number of disjoint, invalid solution subspaces? This kind of enumeration can be captured in algorithmic operations such as crossover or particle velocity in order to take advantage of the search capabilities in the overlying algorithms. Similarly, are there problem-specific enumerations such that the surface of the value plane is continuous almost

141

everywhere? Such enumerations increase the probability of creating a valid solution which can lead to more effective and efficient searching.

The results generated by the CMTS-ready algorithms, despite the solution enumeration, demonstrate that distributed problem solving with task-based decomposition is generally the best approach for solving ASAR problems. But the design of the problem is possibly influencing these results. Additional problem domains that use all of the components of the CMTS problem need to be developed in order to sufficiently generalize the performance characteristics of the distributed process. The difference in problems domains should be clearly delineated in order to precisely capture how the decomposition methods are related to relational constraints if such a relationship exists.

Another direction for future work is to incorporate conditional relational constraints into the CMTS definition. For instance, the binding set, influenced by physical world models, prohibits abilities from concurrently working on separate tasks. But, if those tasks were "close enough," then the binding should be relaxed to allow for agents to concurrently perform both such as two grippers picking up blocks that are centimeters apart. A similar idea applies to inhibitor sets where maybe abilities only inhibit each other within a certain radius, such as mobile antennas.

Finally, the CMTS problem descriptor does not model assignment failure. This dynamic event is only solvable by continuously solving updated problem instances relevant to a dynamic environment. Most real time robot systems handle this on-the-fly using the equivalent to value functions, but whether there is an underlying analytical model for handling situations like preemption or intentional coalition failure are not specifically addressed by this work.

7.3   Summary

In conclusion, this unified framework for multiagent task assignment is an innovative advancement in the multiagent research field. The CMTS problem descriptor

is the first problem model to compositely represent a broad range of multiagent task assignment problems from a number of disciplines in a single description. The descriptor provides a unified approach to representing variable agent-to-task ratios with inter-agent and inter-task relational constraints. The solution representation provides a well-understood structure that single-handedly shows the task allocation and scheduling results for a CMTS problem. The algorithms presented in this work generate optimal and highly effective approximated solutions to simple and challenging problem domains while simultaneously solving the coalition formation and scheduling aspects of the problem. Thus, the CMTS-ready algorithms fulfill a critically missing requirement in the multiagent task assignment field by solving a wide variety of task assignment problems without requiring modification and in a single stage. Furthermore, a distributed process with an appropriate problem decomposition technique applied to solve the domains used in these experiments empirically shows significant reduction in solution search size without impacting solution quality.

Experiments using simple and complex CMTS problem domains show that multiagent approaches to solving CMTS problems can outperform single-agent approaches. The multiagent approach performs than better single-agent approaches in cooperative and competitive environments, regardless of whether tasks require multiple agents or time-extended planning. For complex problems, the serial auction algorithm provides the best average utility value for individual and distributed processing. However, the task decomposition method for a distributed system often improves solution quality. Task decomposition is also the most effective method for reducing the solution space size for distributed search algorithms based on the experimental results summarized in Section 6.5. Yet, no single approximation algorithm provided the best answer across the span of all problem instances as predicted by the No Free Lunch theorems [122], which state that there is no best algorithm for optimization problems.

Therefore, the unified framework, consisting of unified problem descriptor and problem-independent algorithmic solvers, offers a viable approach to researching mul-

143

tiagent task assignment through a universal interface. The techniques illustrated by this research demonstrate the basic approach to representing and solving constrained assignment problems while incorporating basic agent behaviors in a multiagent environment. This research opens the door for hybridized algorithm development, distributed solution development with high scalability and real-time scheduling in a dynamic environment using techniques coming from several disciplines.

144

Appendix A. Problem Specific Experiment Results

This appendix provides individual results graphs for each problem. The top-most figure shows the solution utility values generated by each algorithm with the horizontal axis showing the algorithms rank-ordered with the best performing algorithm to the left. The line across the chart represents the average solution value over all of the algorithms for the one problem. The middle figure shows the percent error in solution value each algorithm produces from the best value by a black bar. The lower the bar, the lower the difference percentage-wise from the best known utility solution value for the problem. Algorithms with no bar found the best solution. The bottom figure illustrates the total size of the solution space used to generate solutions relative to the original problem size. This primarily highlights how much solution space the decomposition methods are using when searching for solutions.
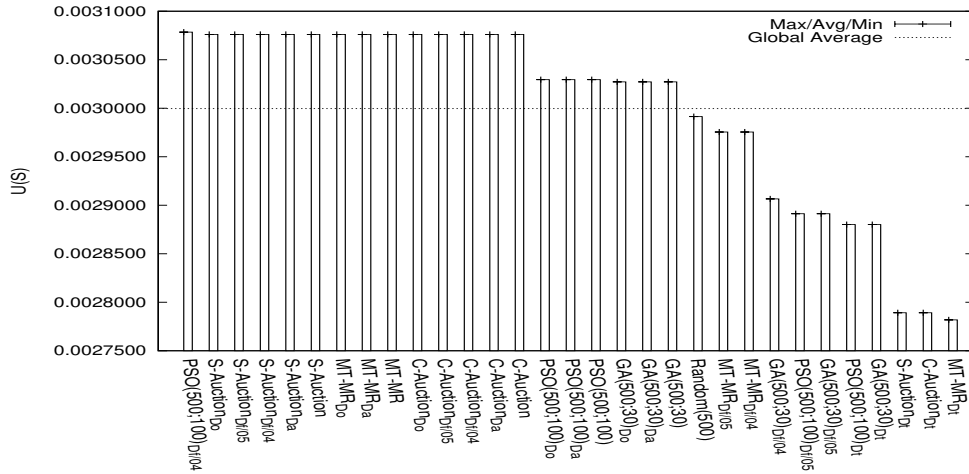
(a) Sorted Utilities.
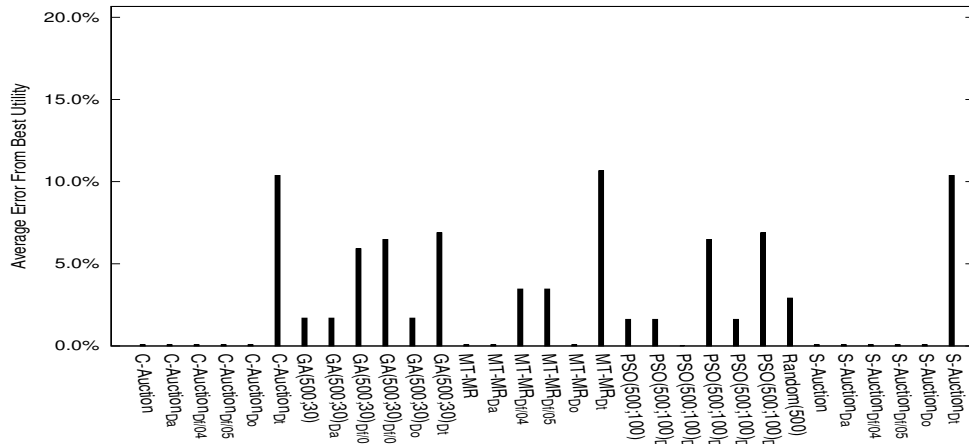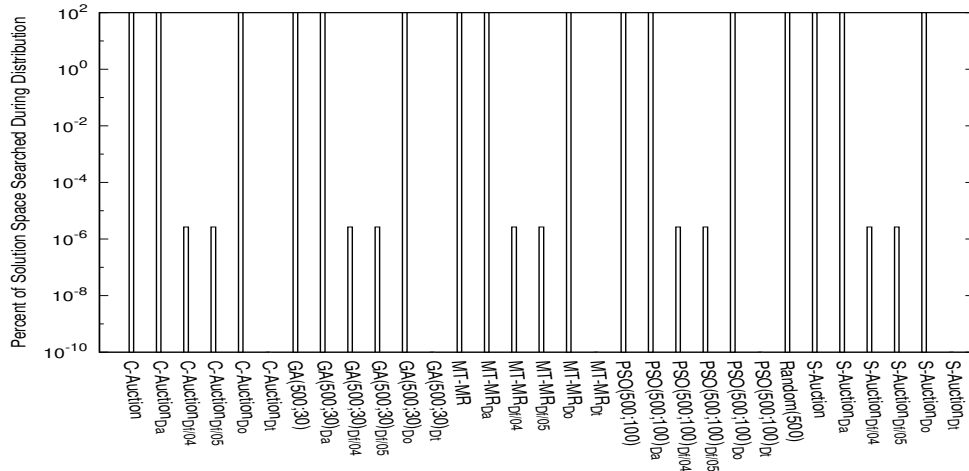
(b) Utility Error.

(c) Size Reduction.

Figure A.1: 3×6 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.2: 3×7 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.3: 3×8 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.4: 3×9 ASAR Problem Results.

149

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.5: 3×10 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.6: 3×11 ASAR Problem Results.

151

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.7: 3×12 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.8: 3×13 ASAR Problem Results.

153
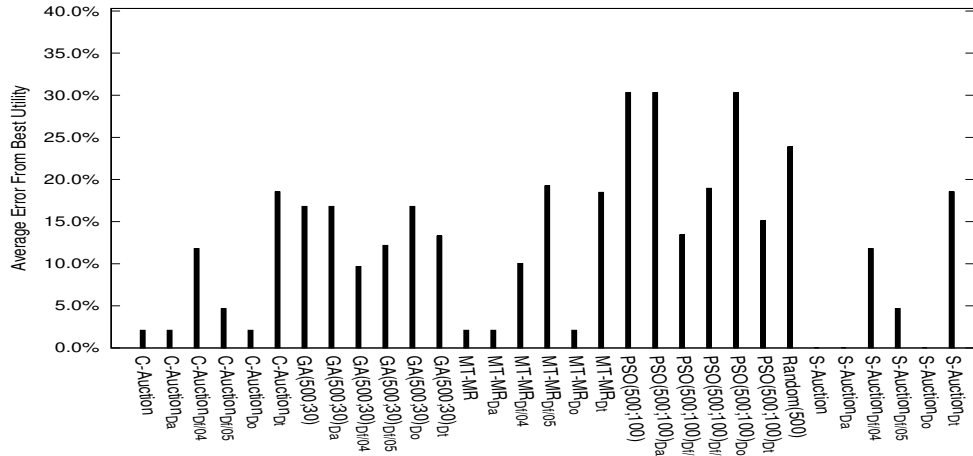
(a) Sorted Utilities.
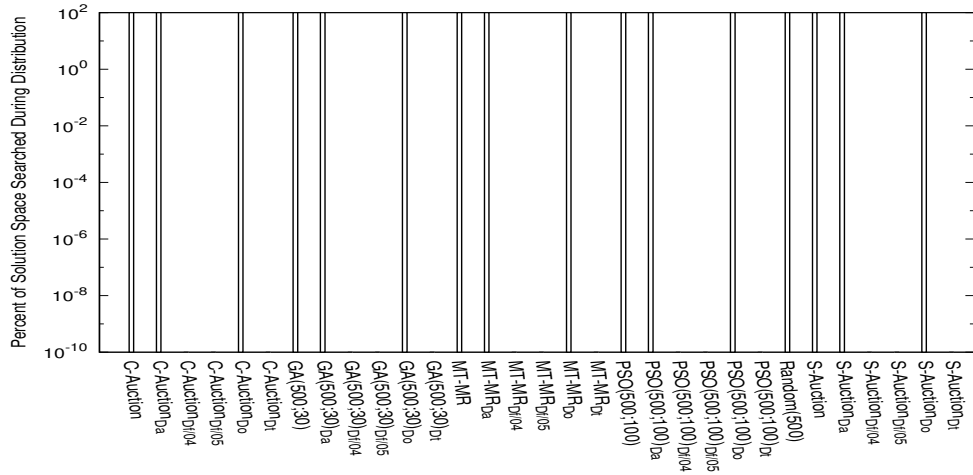
(b) Utility Error.

(c) Size Reduction.

Figure A.9: 3×14 ASAR Problem Results.

154

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.10: 3×15 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.11: 4×6 ASAR Problem Results.
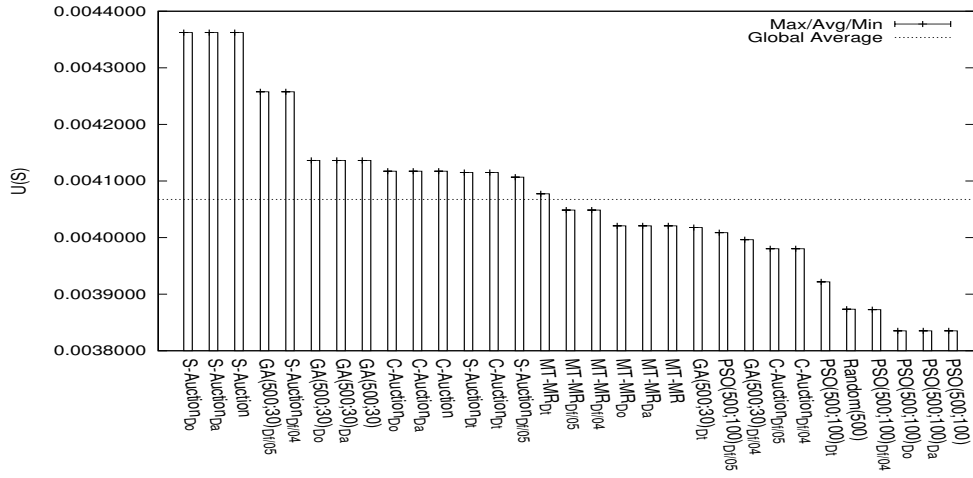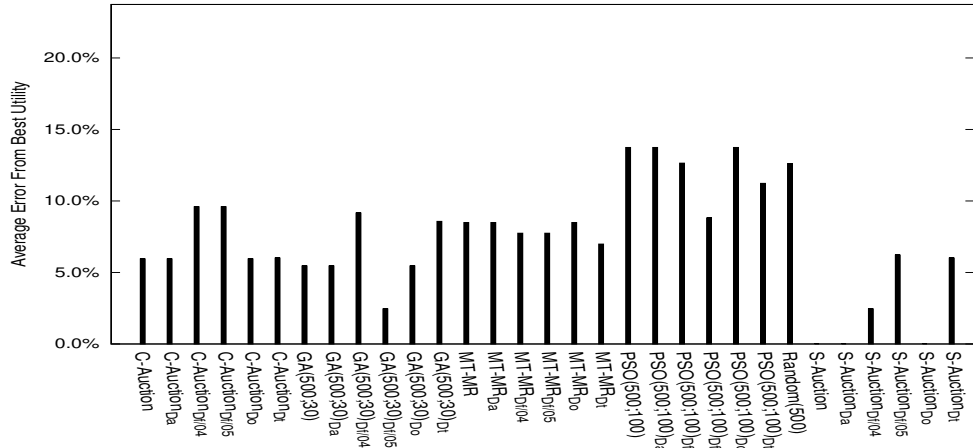
(a) Sorted Utilities.
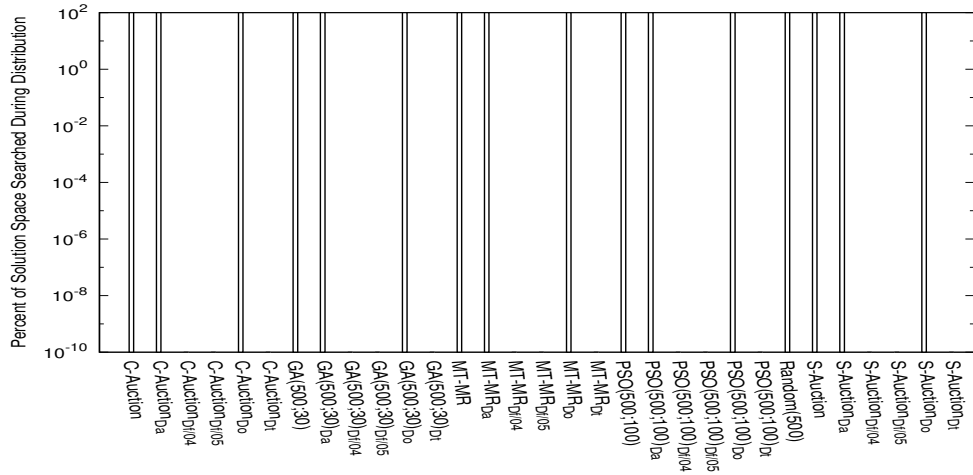
(b) Utility Error.

(c) Size Reduction.

Figure A.12: 4×7 ASAR Problem Results.
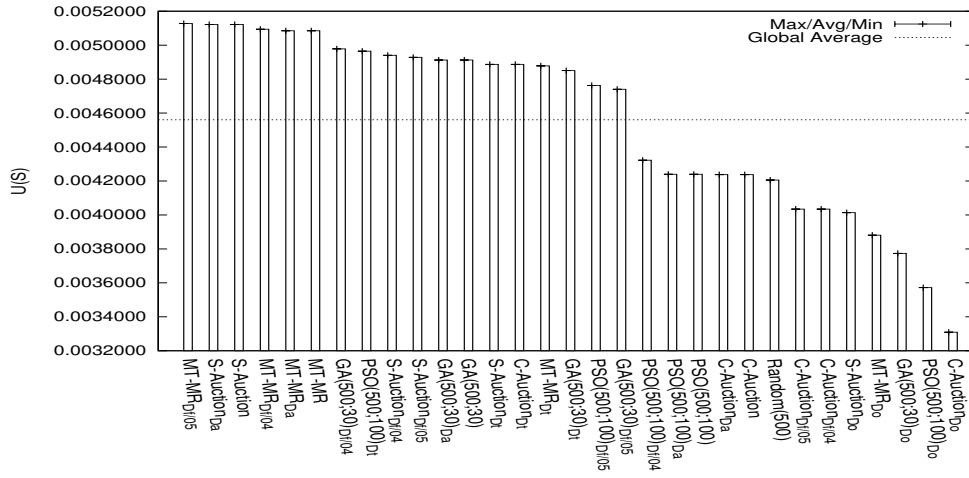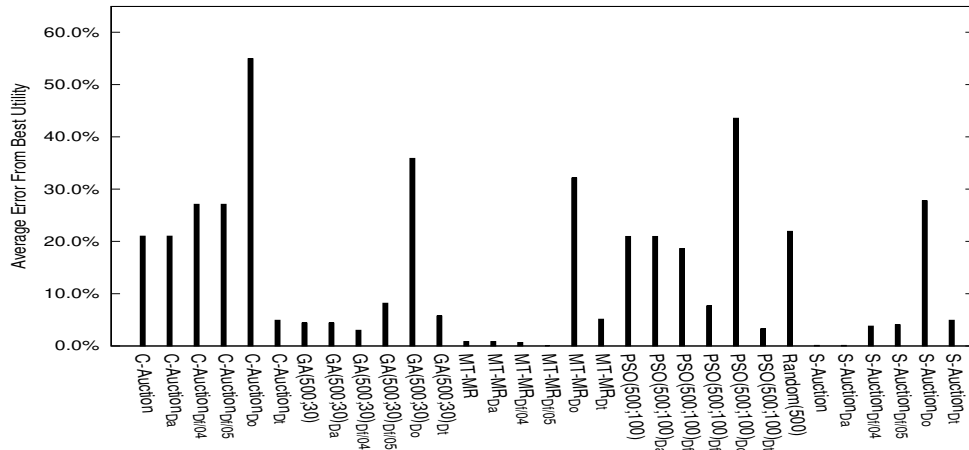
(a) Sorted Utilities.
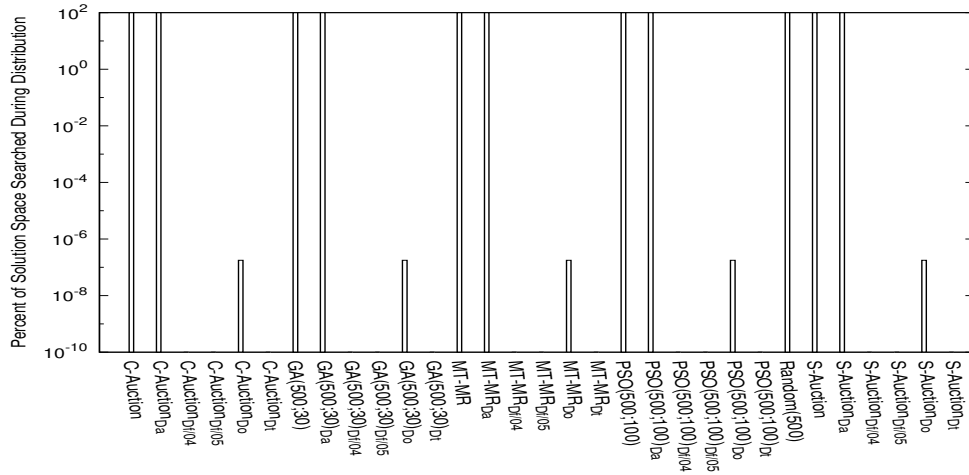
(b) Utility Error.

(c) Size Reduction.

Figure A.13: 4×8 ASAR Problem Results.
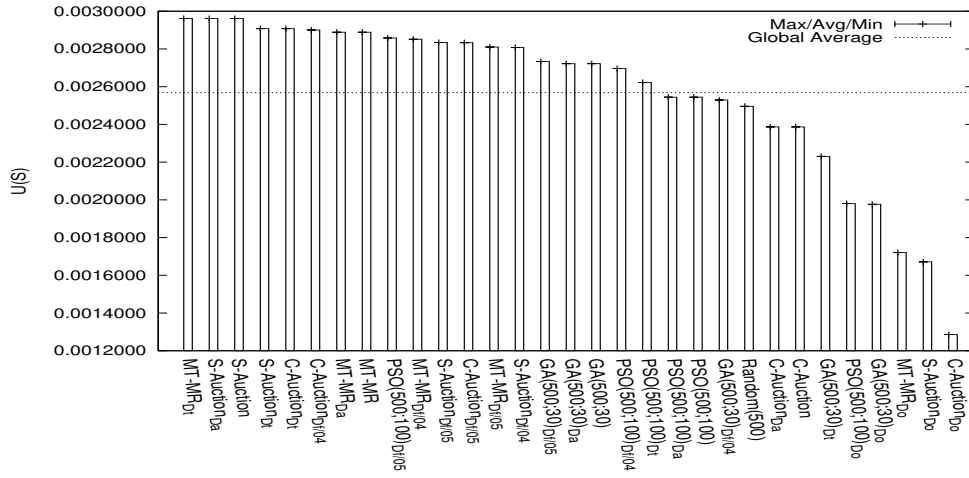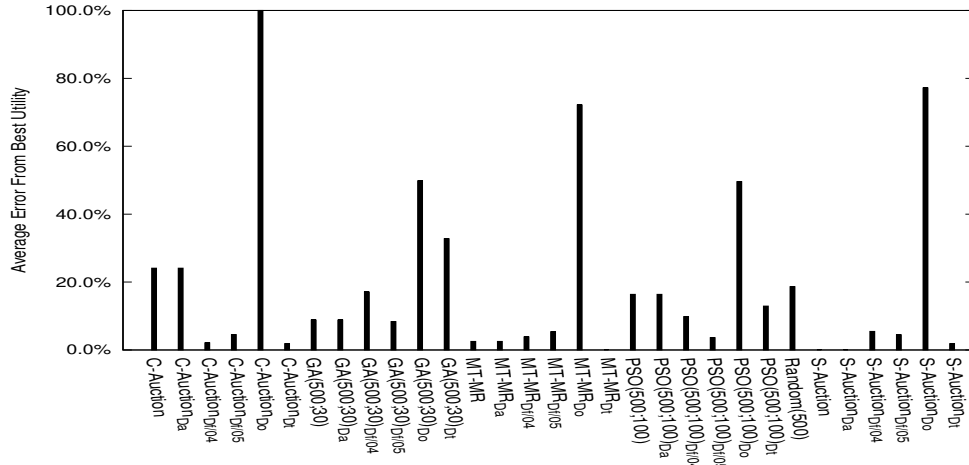
(a) Sorted Utilities.
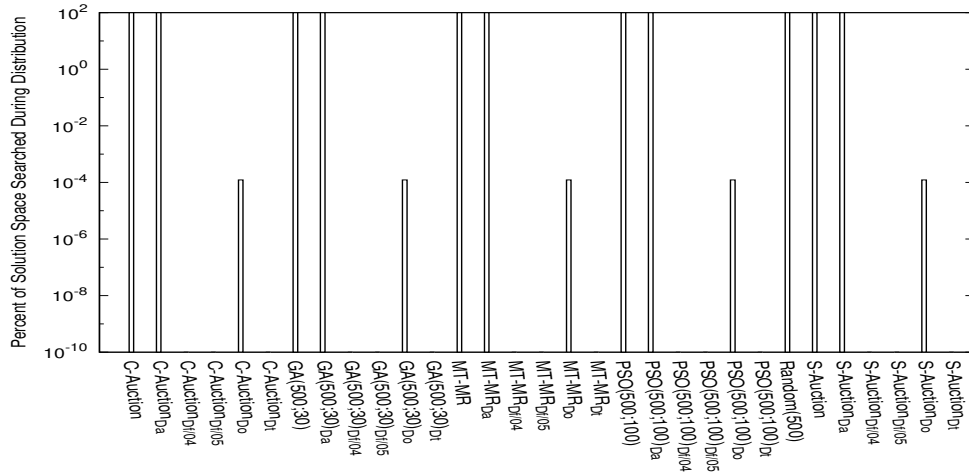
(b) Utility Error.

(c) Size Reduction.

Figure A.14: 4×9 ASAR Problem Results.

159
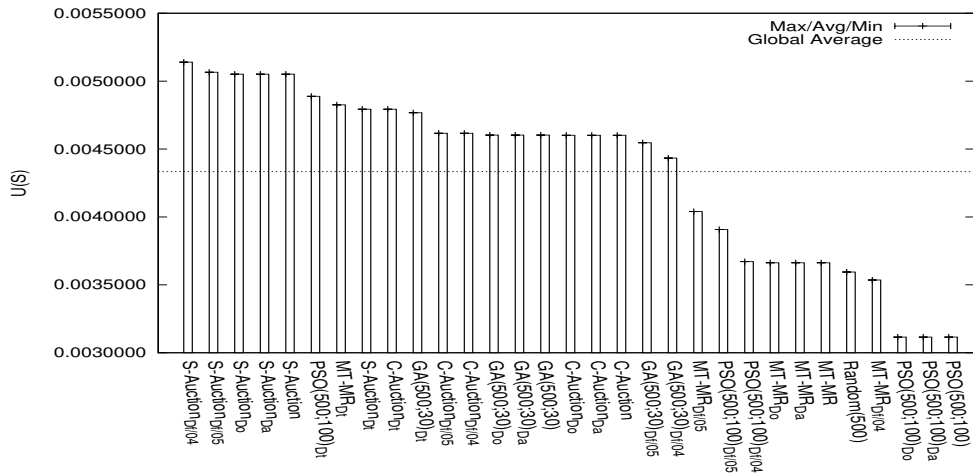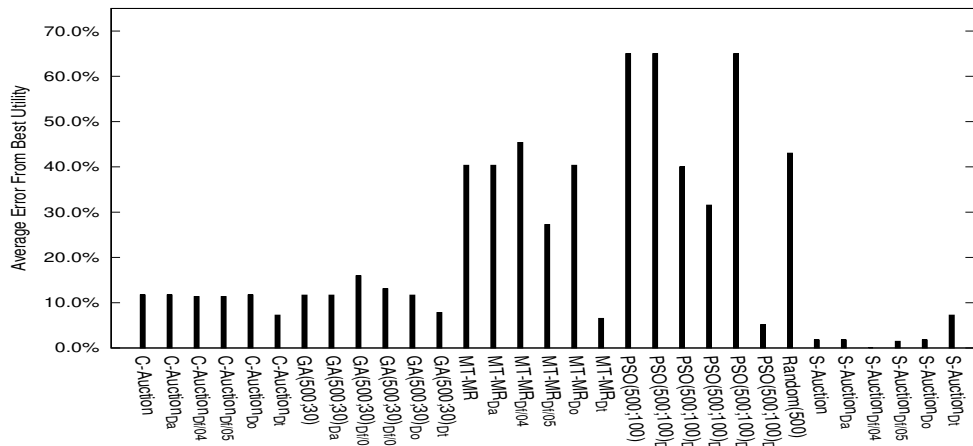
(a) Sorted Utilities.
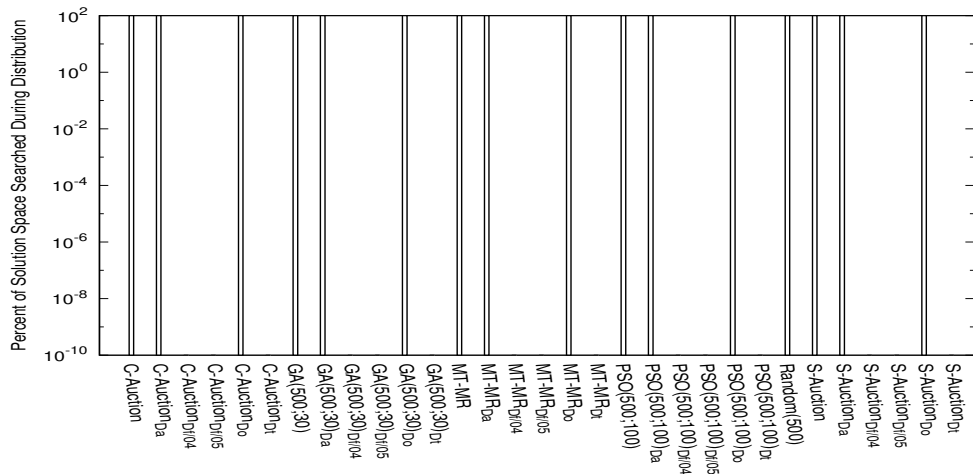
(b) Utility Error.

(c) Size Reduction.

Figure A.15: 4×10 ASAR Problem Results.
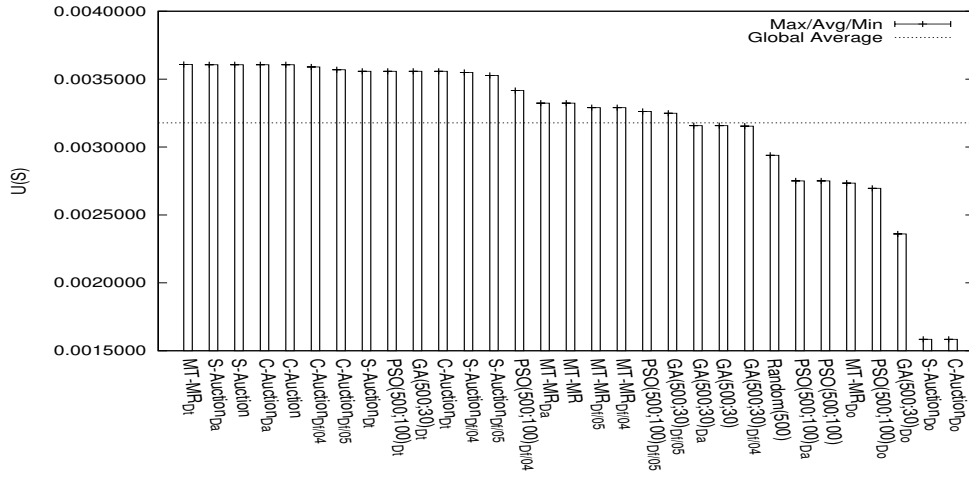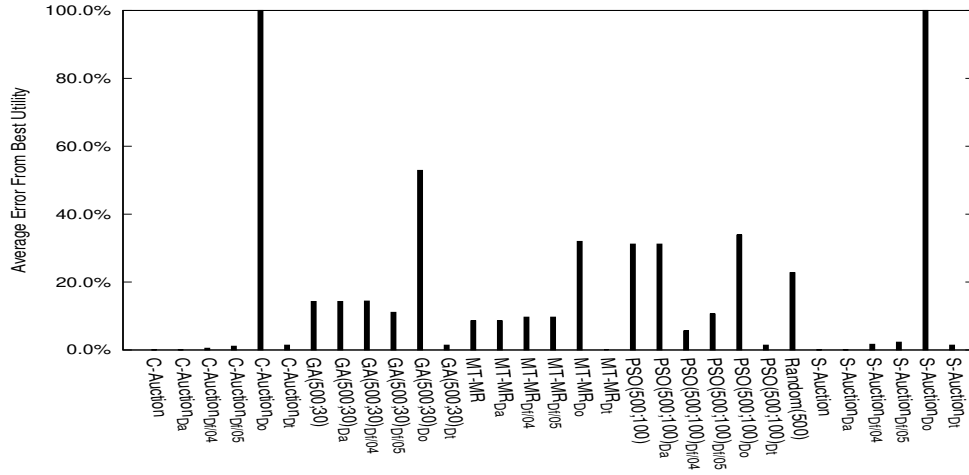
(a) Sorted Utilities.
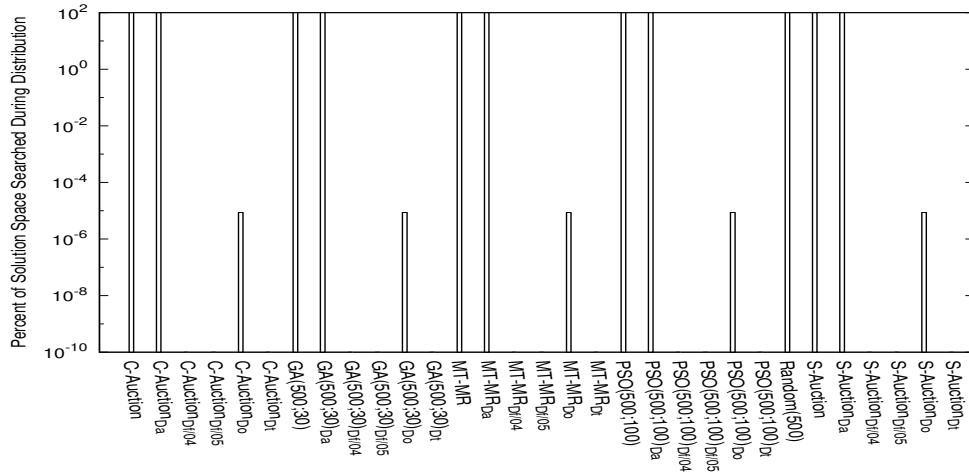
(b) Utility Error.

(c) Size Reduction.

Figure A.16: 4×11 ASAR Problem Results.
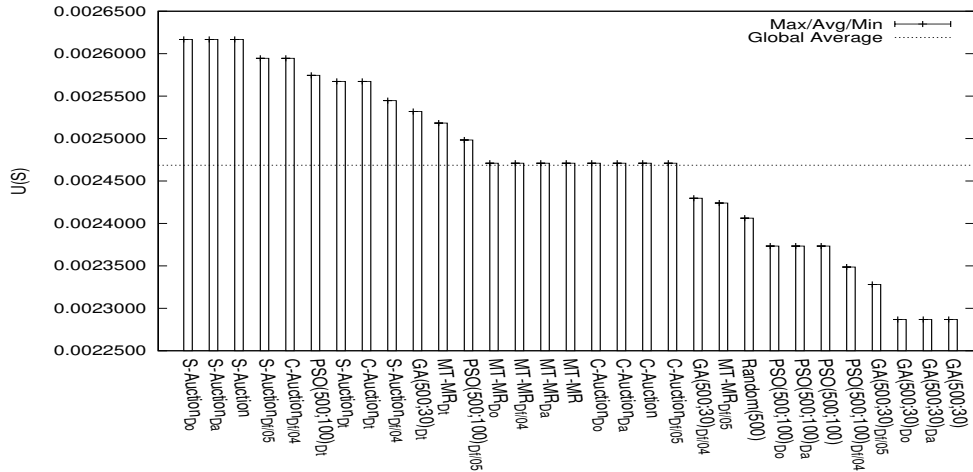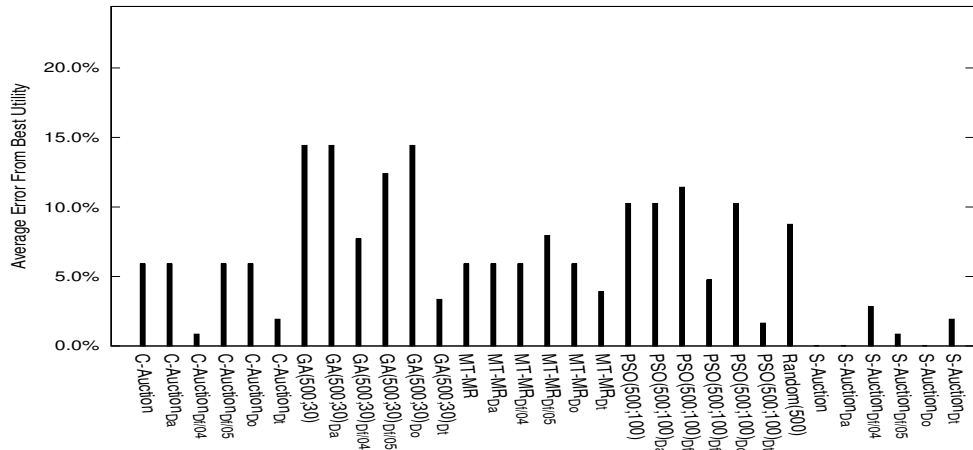
(a) Sorted Utilities.
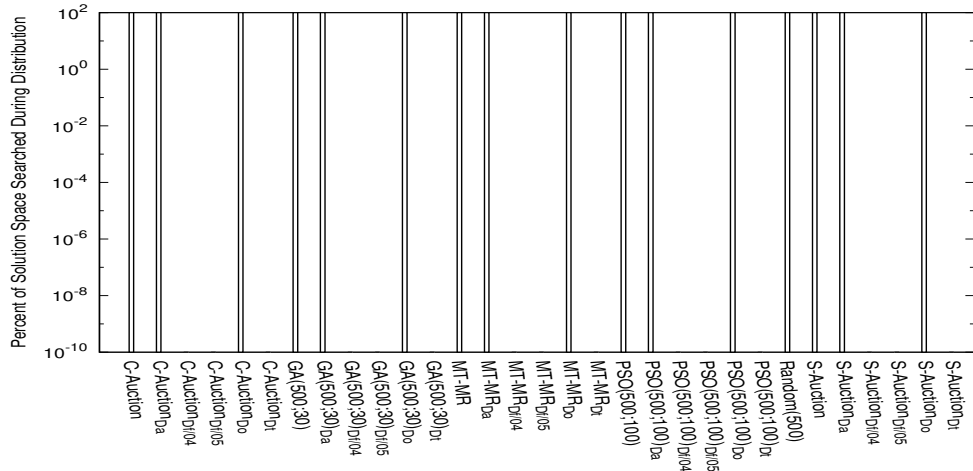
(b) Utility Error.

(c) Size Reduction.

Figure A.17: 4×12 ASAR Problem Results.

162
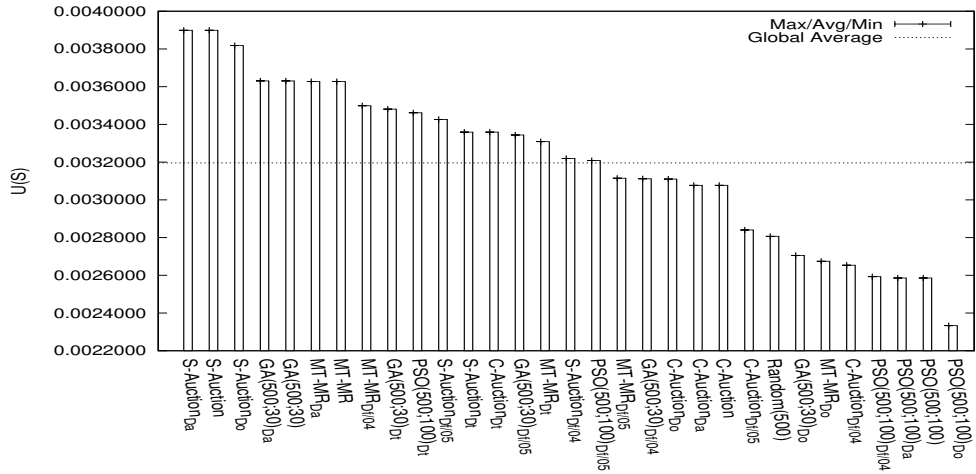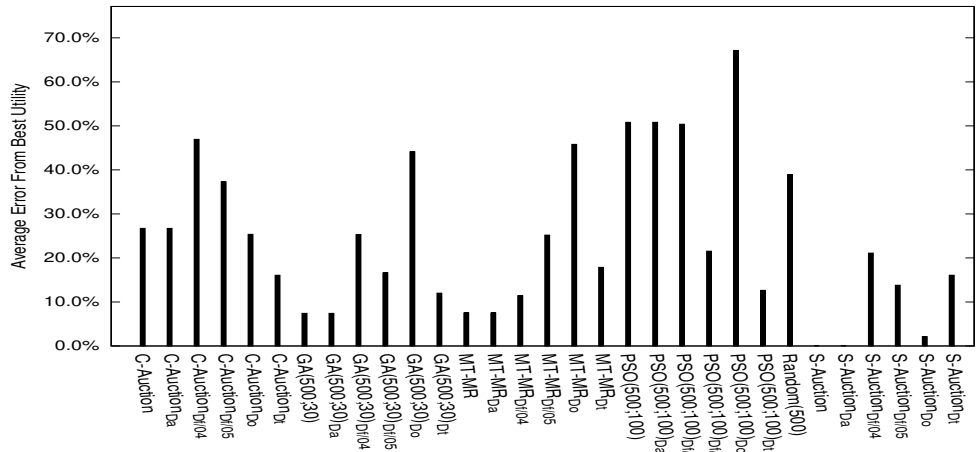
(a) Sorted Utilities.
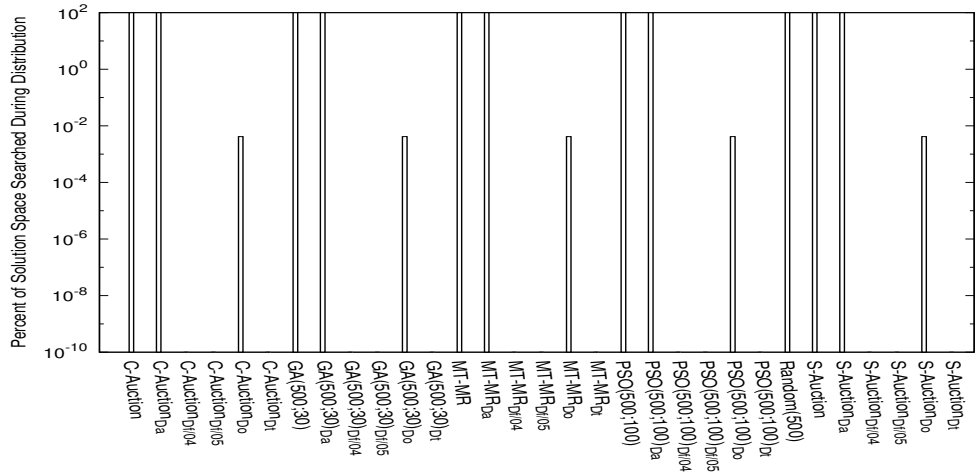
(b) Utility Error.

(c) Size Reduction.

Figure A.18: 4×13 ASAR Problem Results.

163
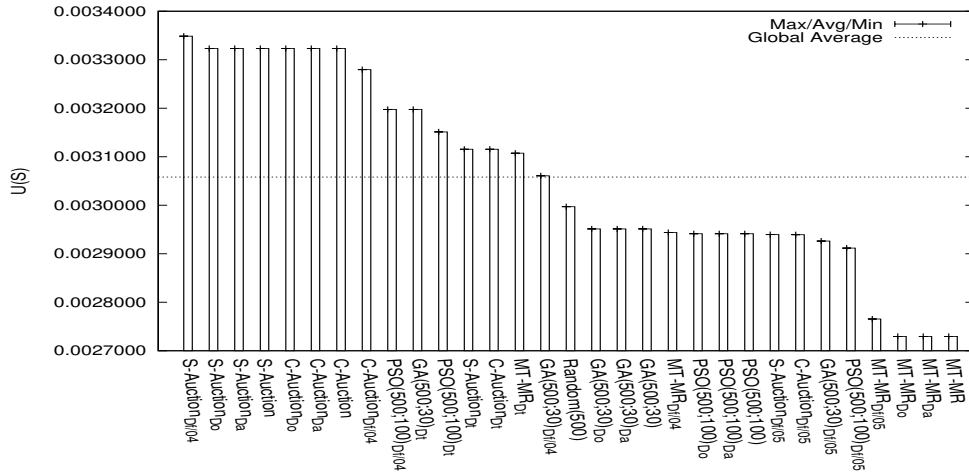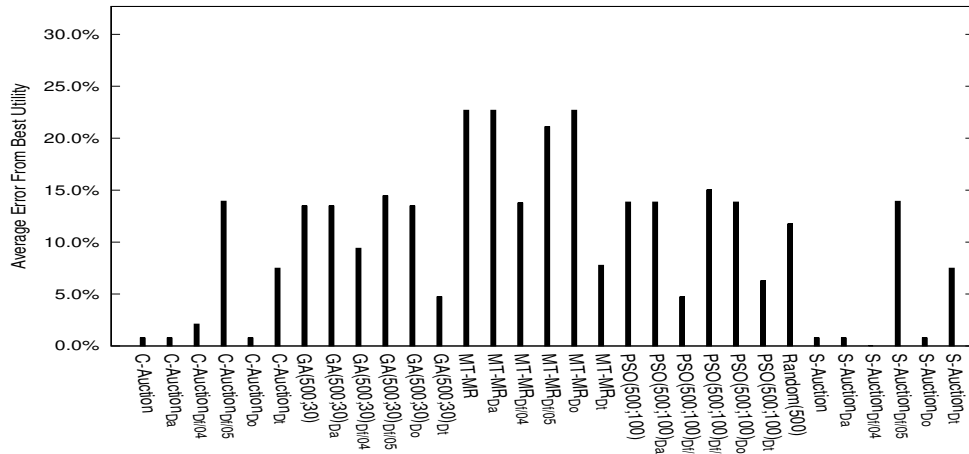
(a) Sorted Utilities.
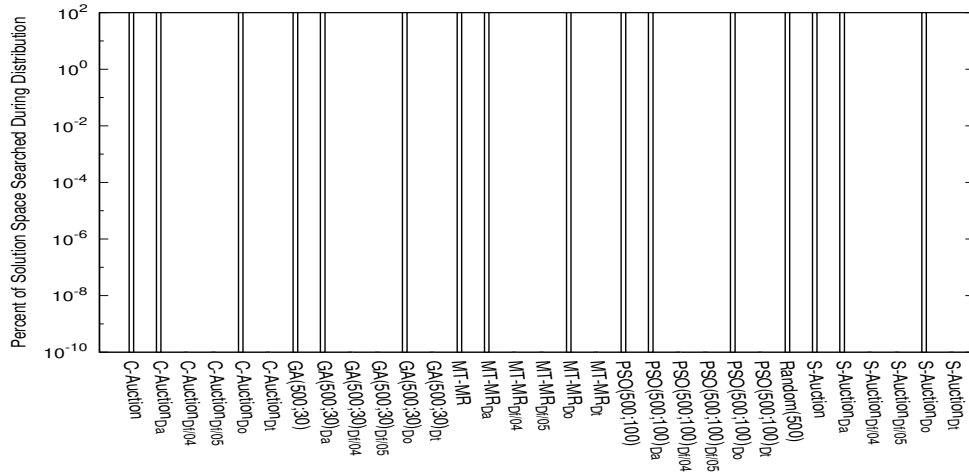
(b) Utility Error.

(c) Size Reduction.

Figure A.19: 4×14 ASAR Problem Results.
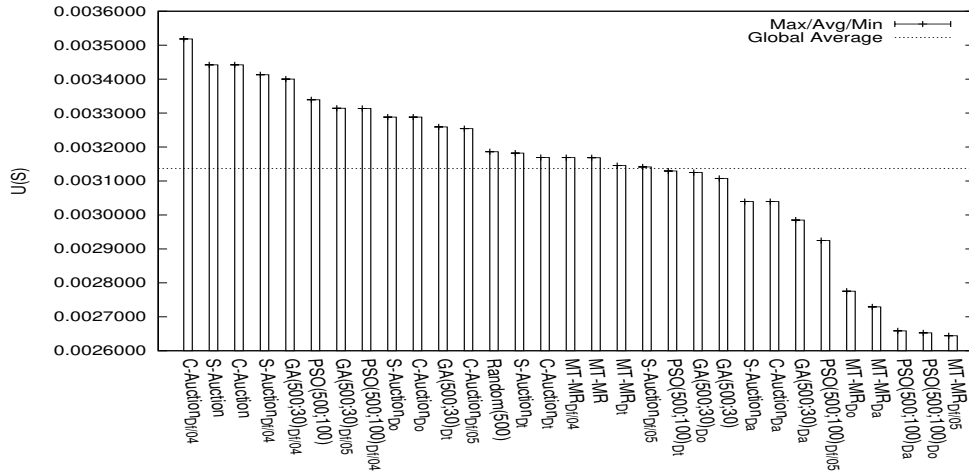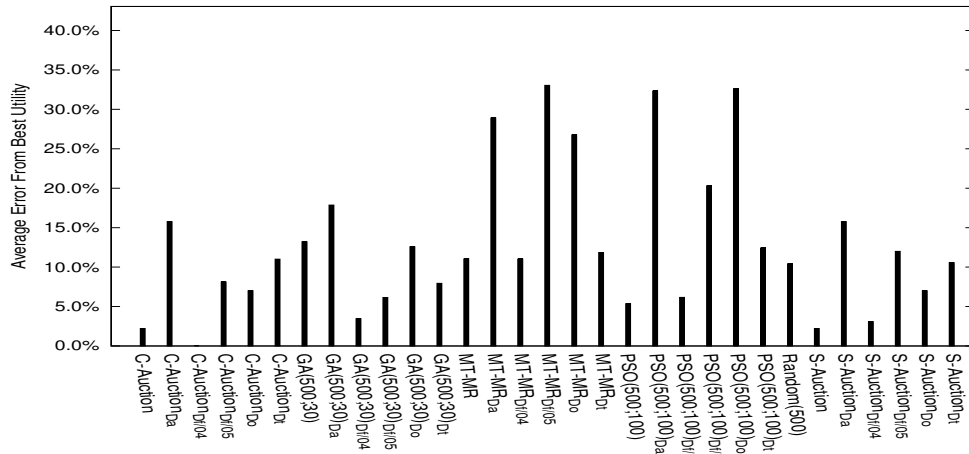
(a) Sorted Utilities.
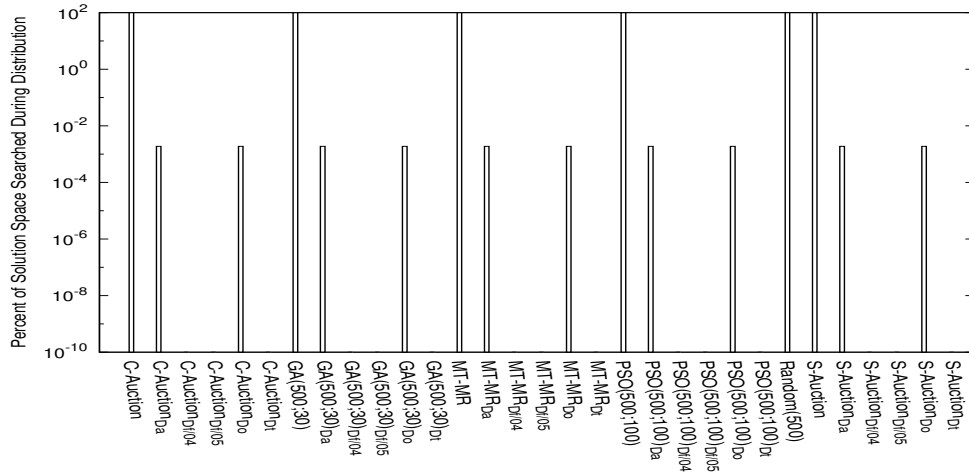
(b) Utility Error.

(c) Size Reduction.

Figure A.20: 4×15 ASAR Problem Results.

165
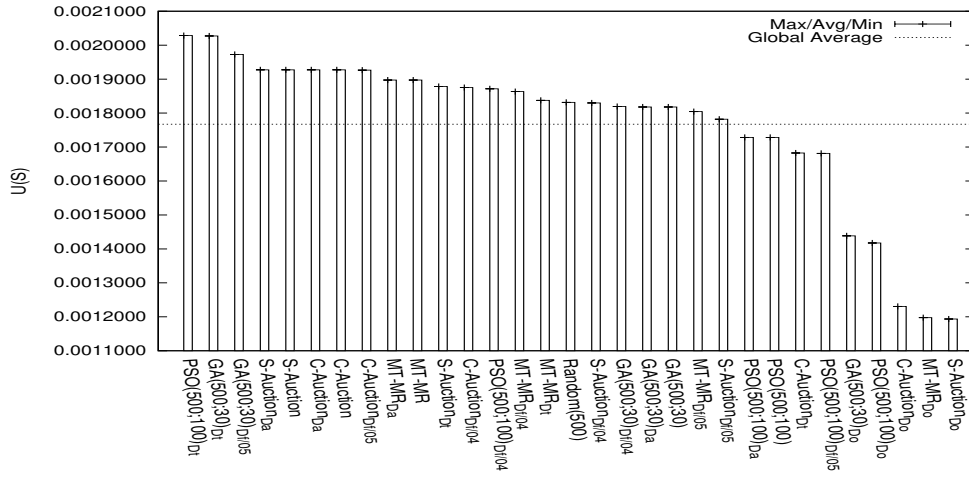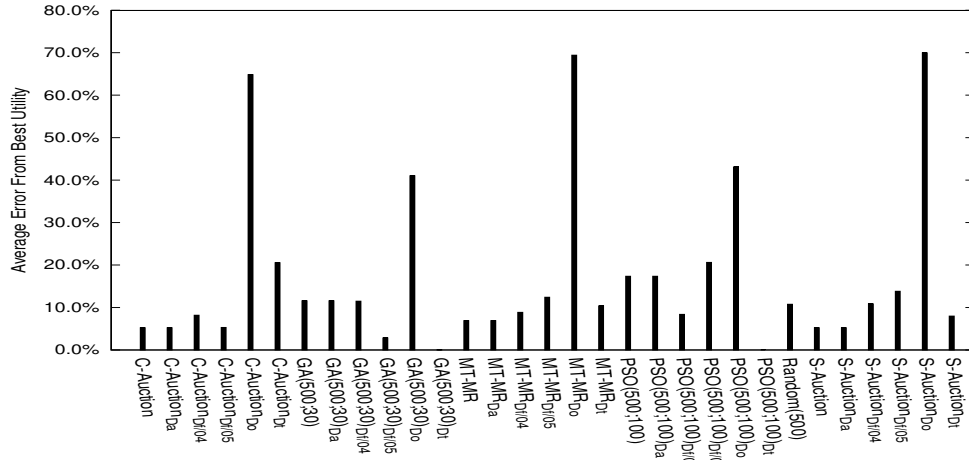
(a) Sorted Utilities.
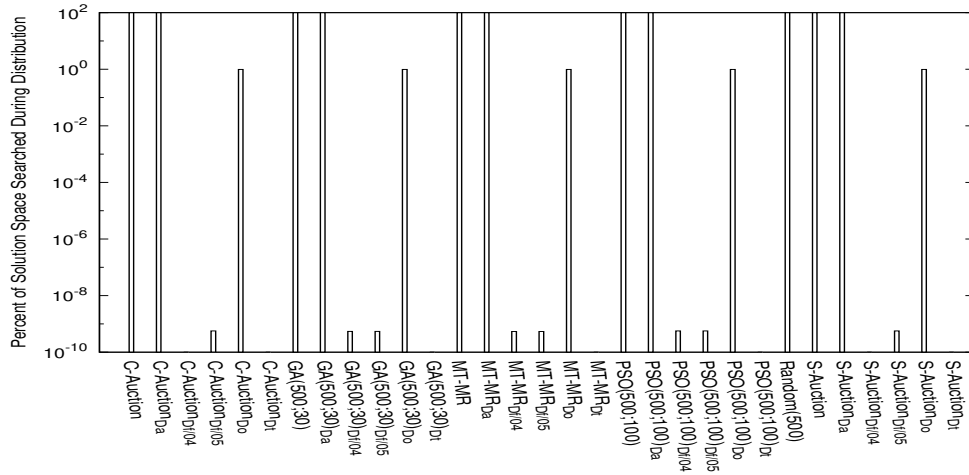
(b) Utility Error.

(c) Size Reduction.

Figure A.21: 5×6 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.22: 5×7 ASAR Problem Results.
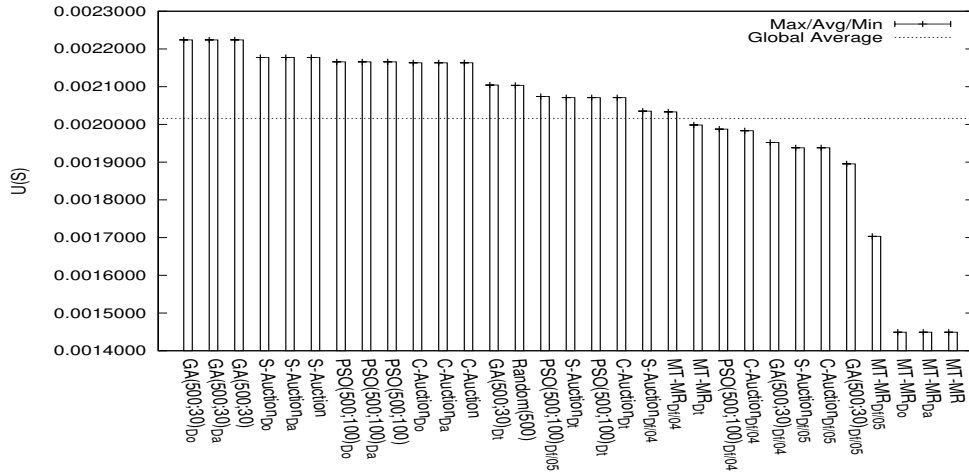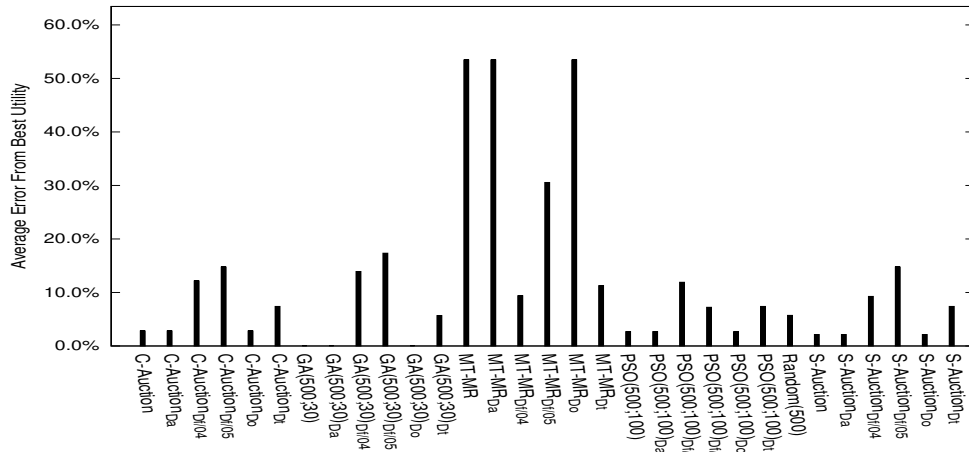
(a) Sorted Utilities.
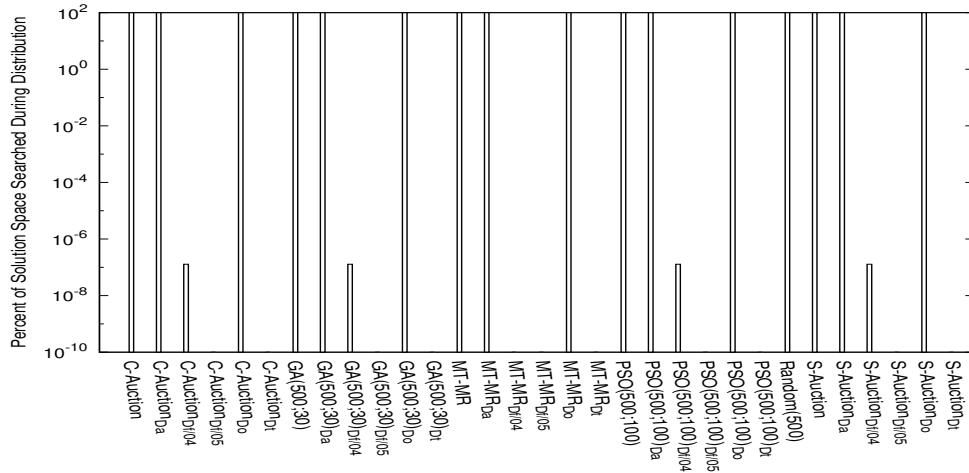
(b) Utility Error.

(c) Size Reduction.

Figure A.23: 5×8 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.24: 5×9 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.

(c) Size Reduction.

Figure A.25: 5×10 ASAR Problem Results.
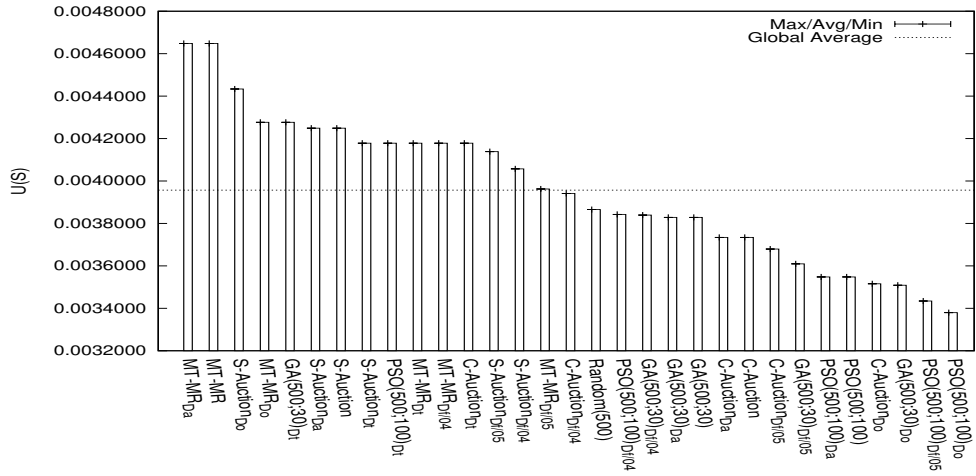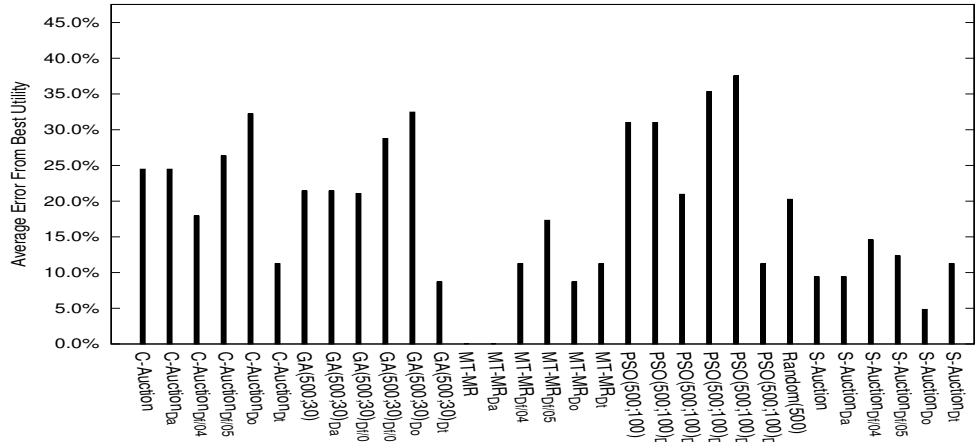
(a) Sorted Utilities.
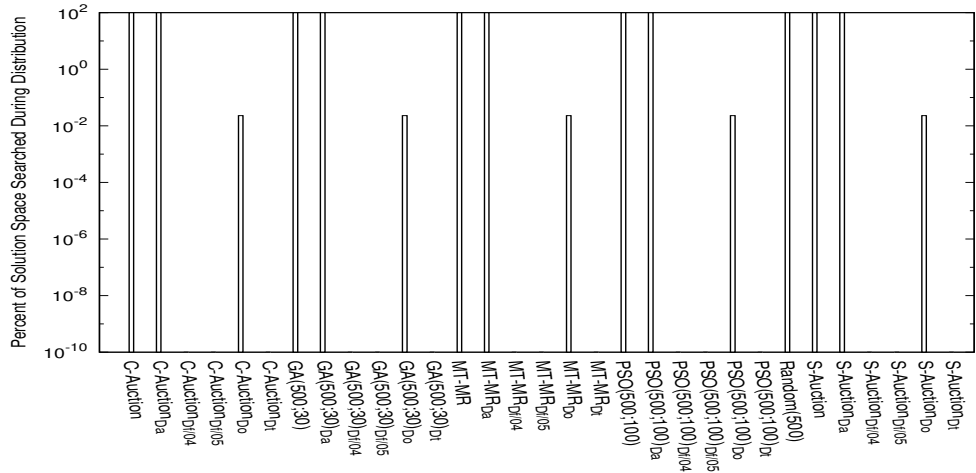
(b) Utility Error.

(c) Size Reduction.

Figure A.26: 5×11 ASAR Problem Results.

171
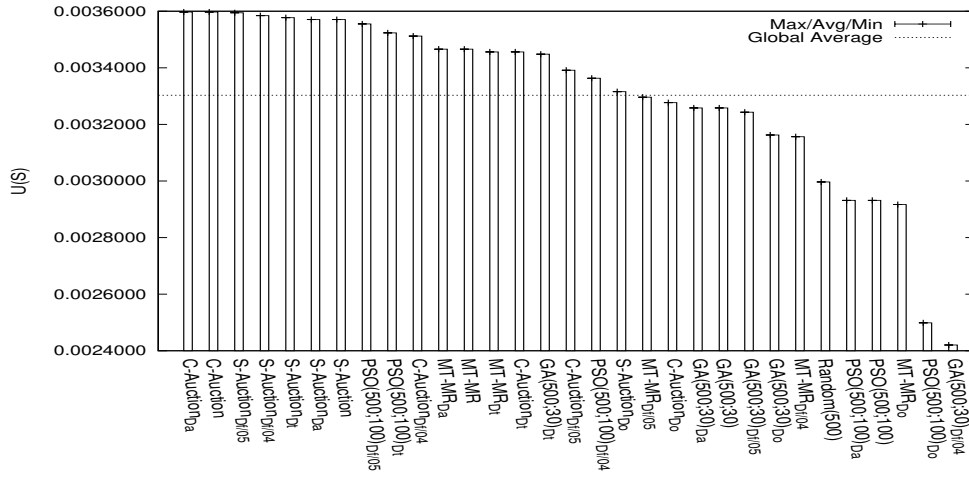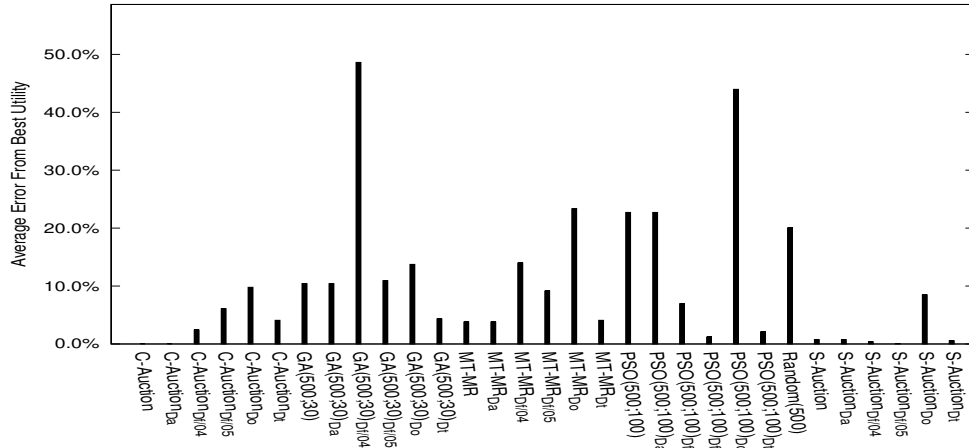
(a) Sorted Utilities.
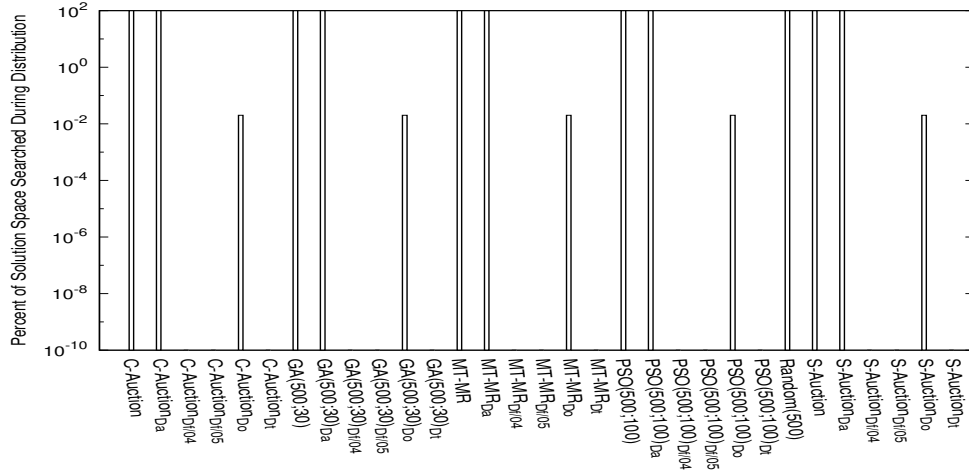
(b) Utility Error.

(c) Size Reduction.

Figure A.27: 5×12 ASAR Problem Results.
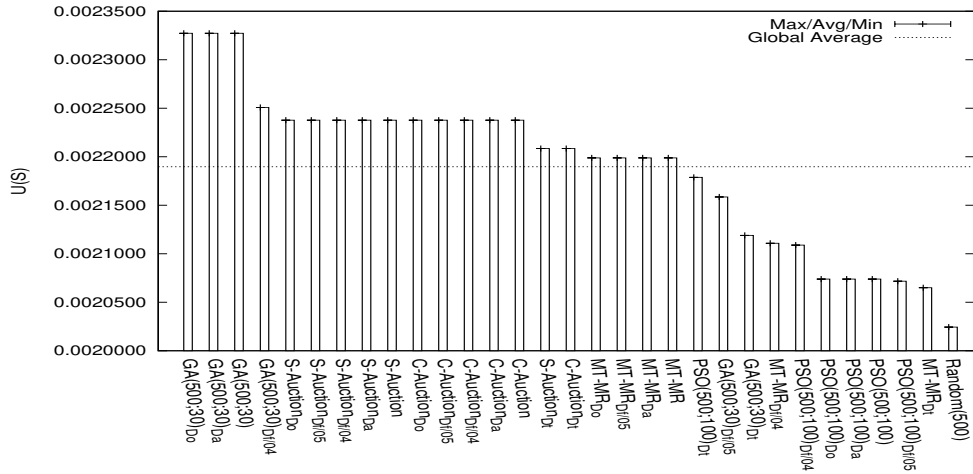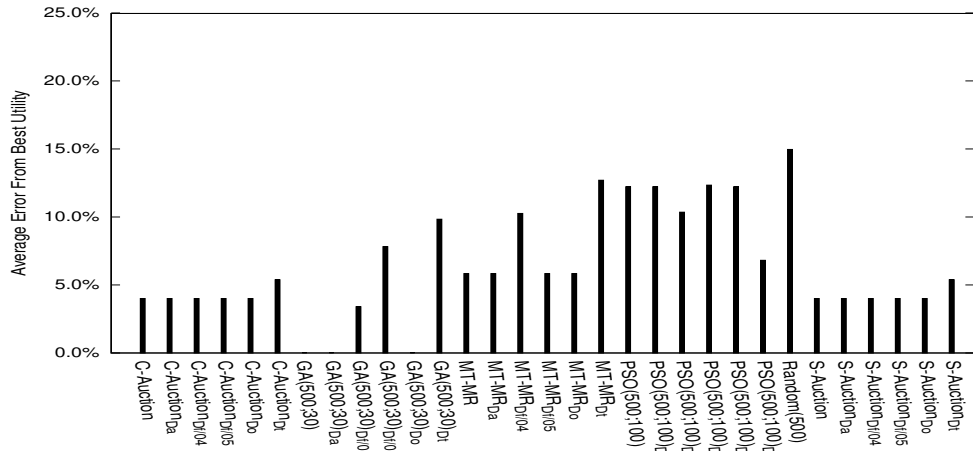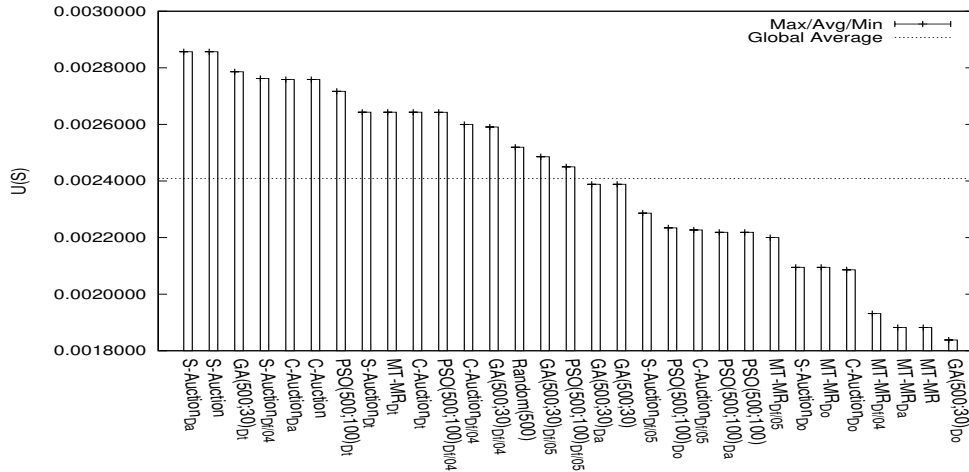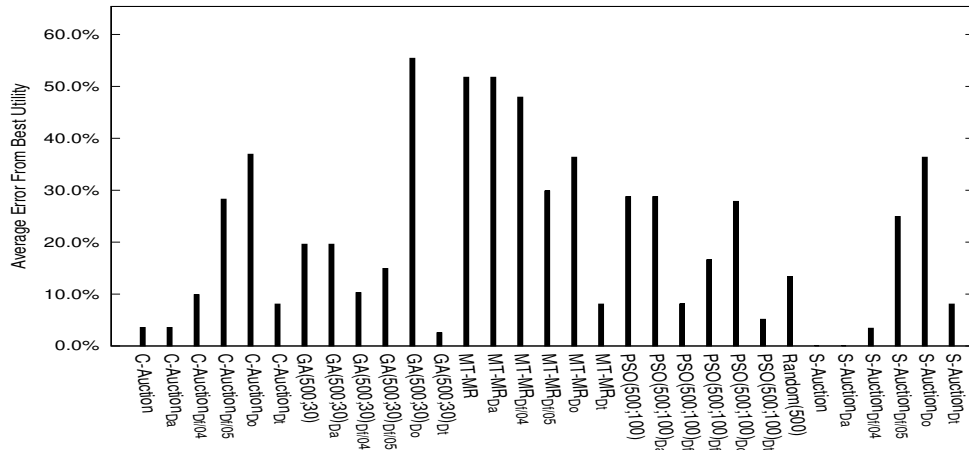
(a) Sorted Utilities.
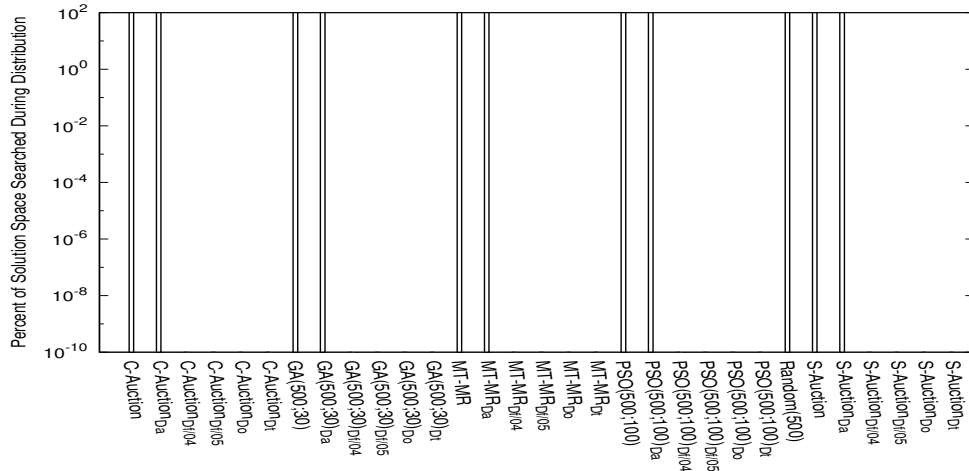
(b) Utility Error.

(c) Size Reduction.

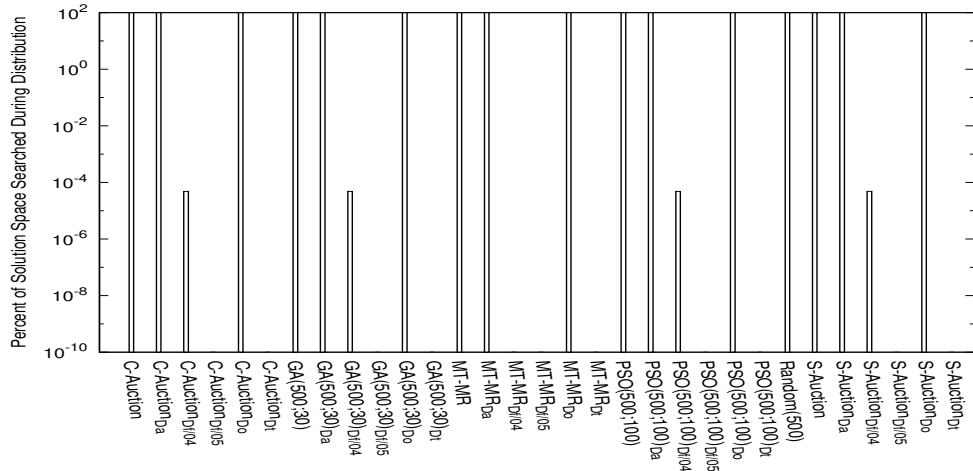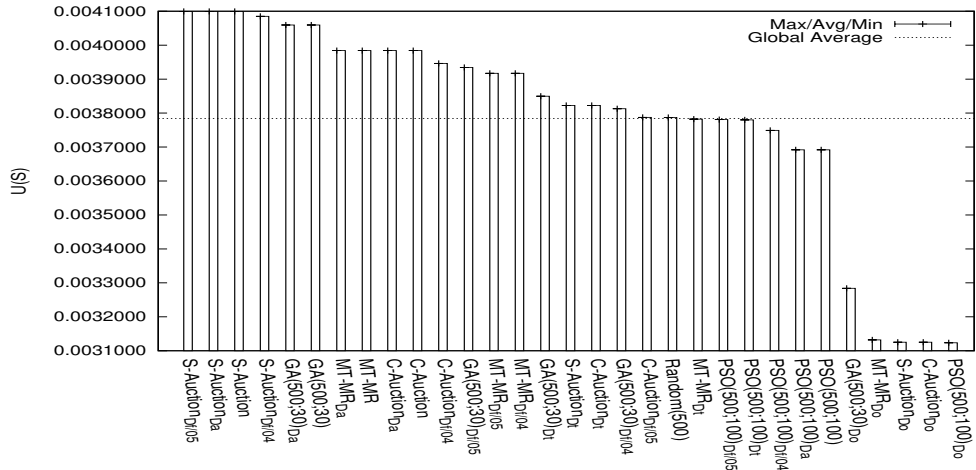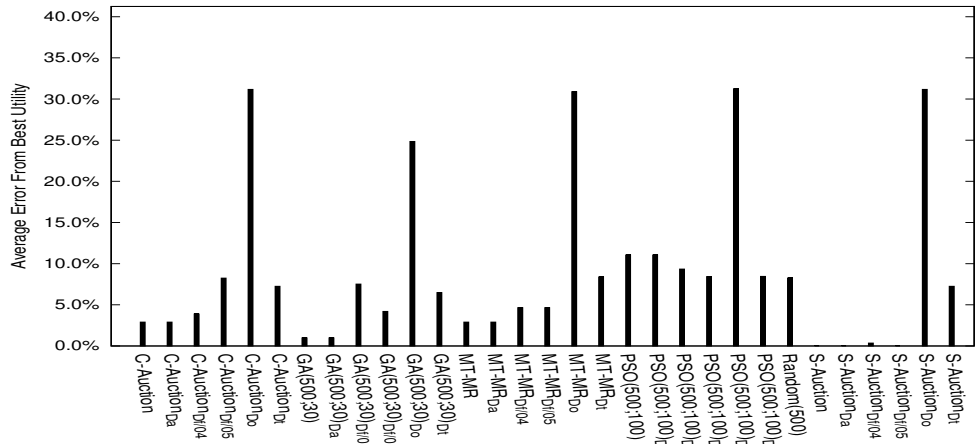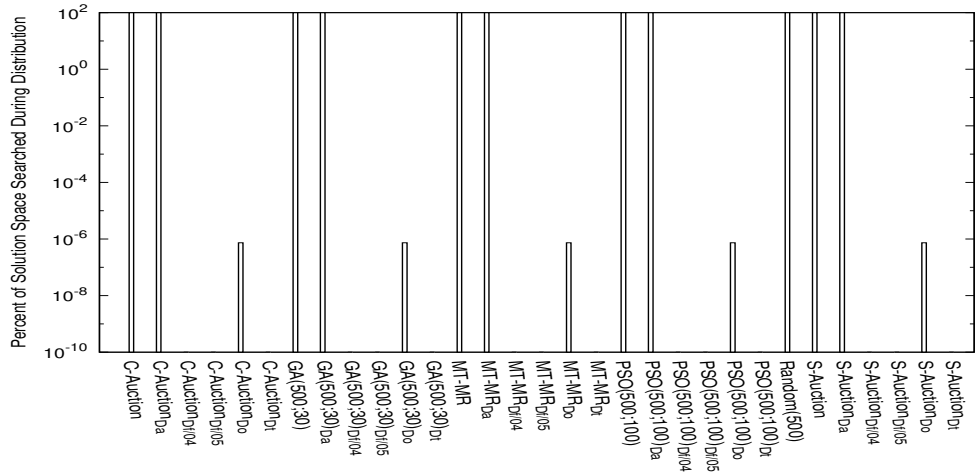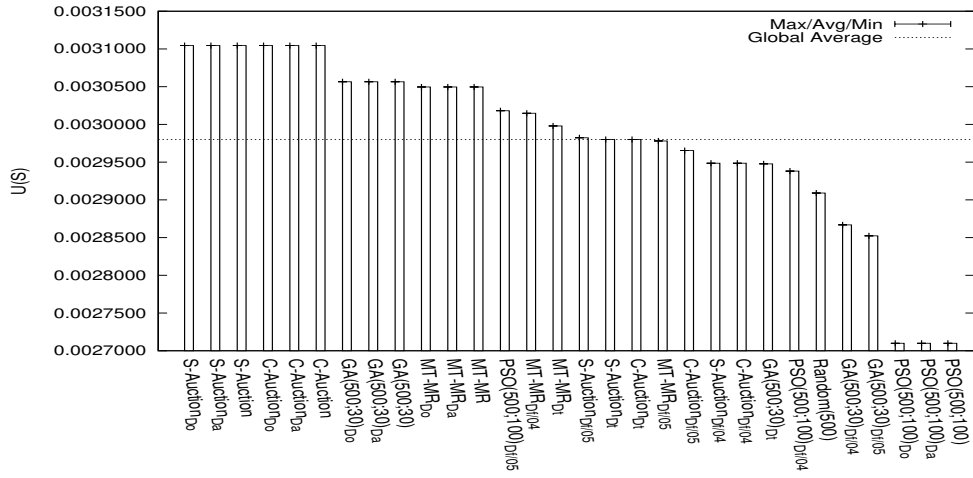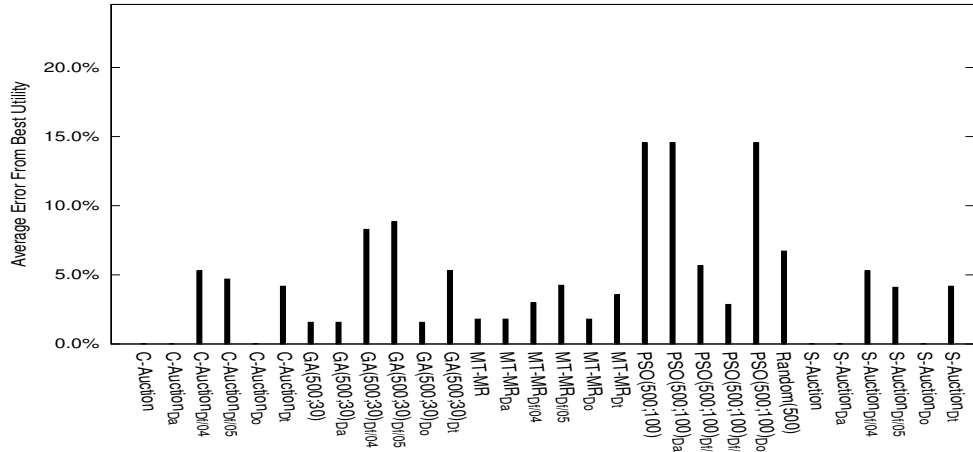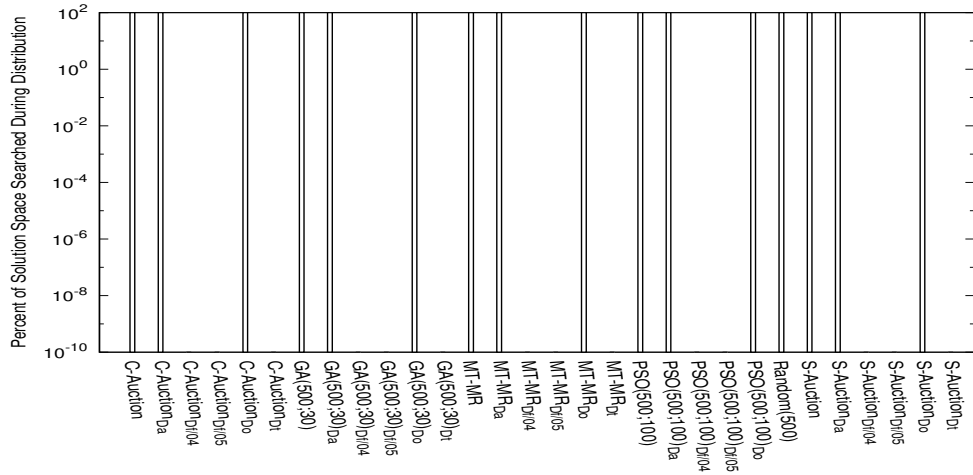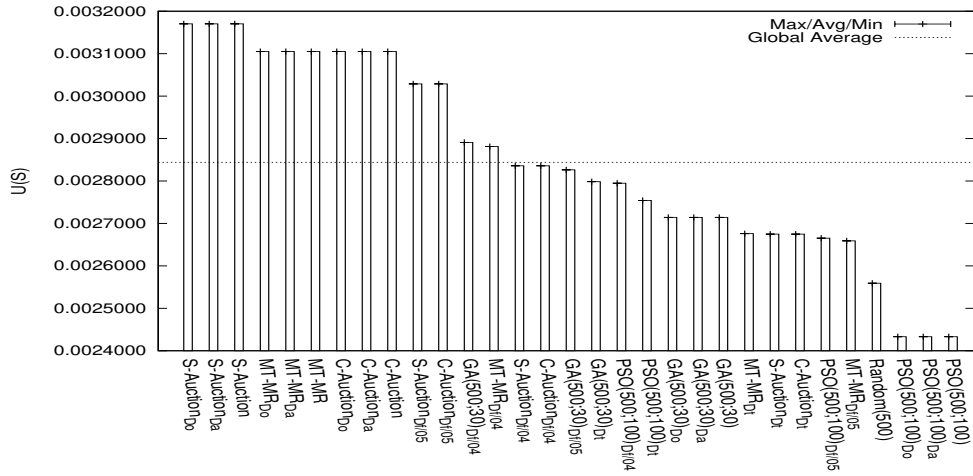Figure A.28: 5×13 ASAR Problem Results.

(a) Sorted Utilities.

(b) Utility Error.
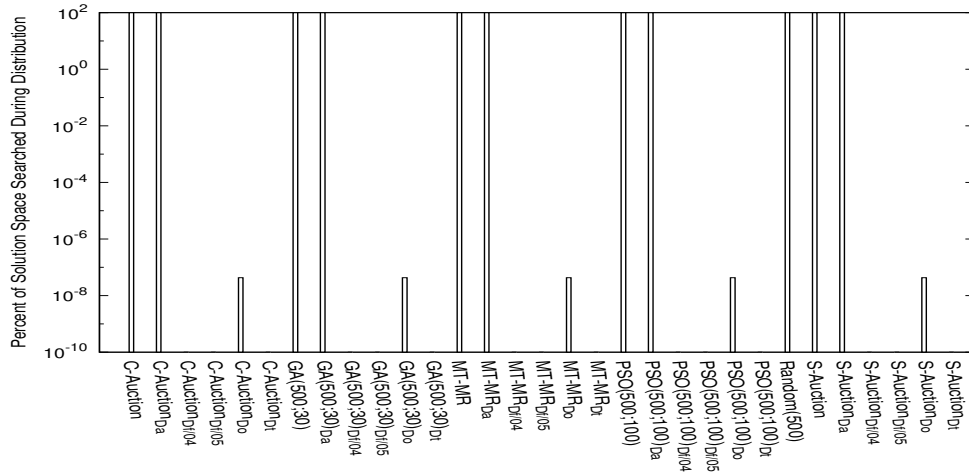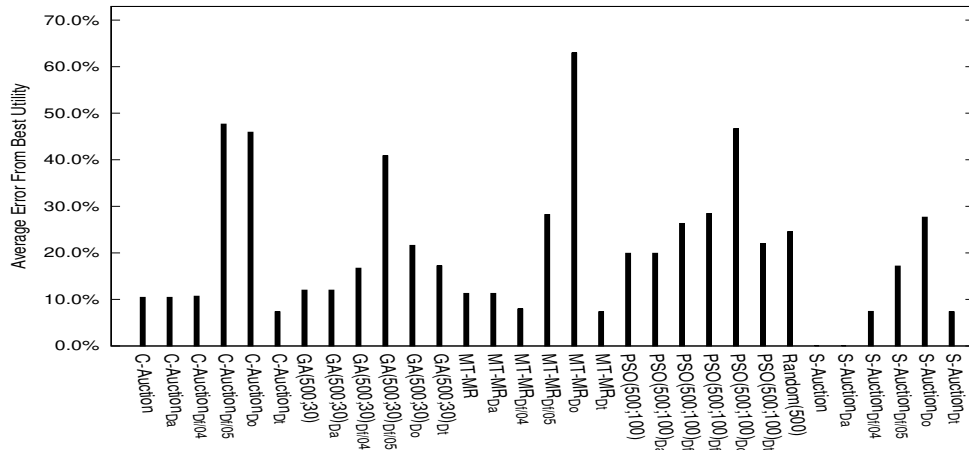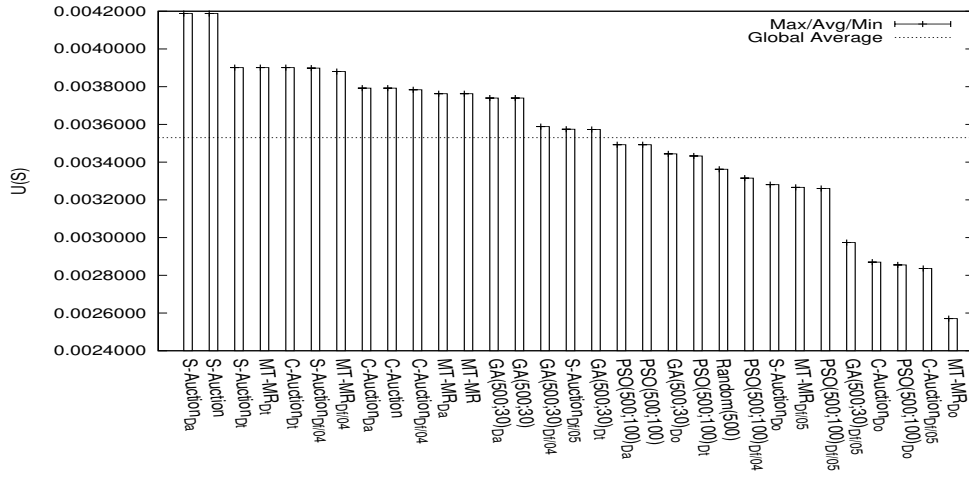
(c) Size Reduction.

Figure A.29: 5×14 ASAR Problem Results.

174
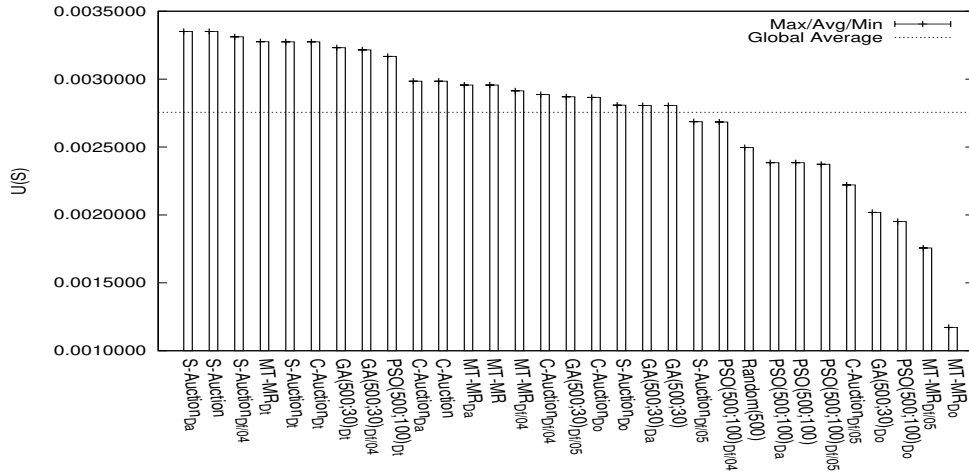
(a) Sorted Utilities.
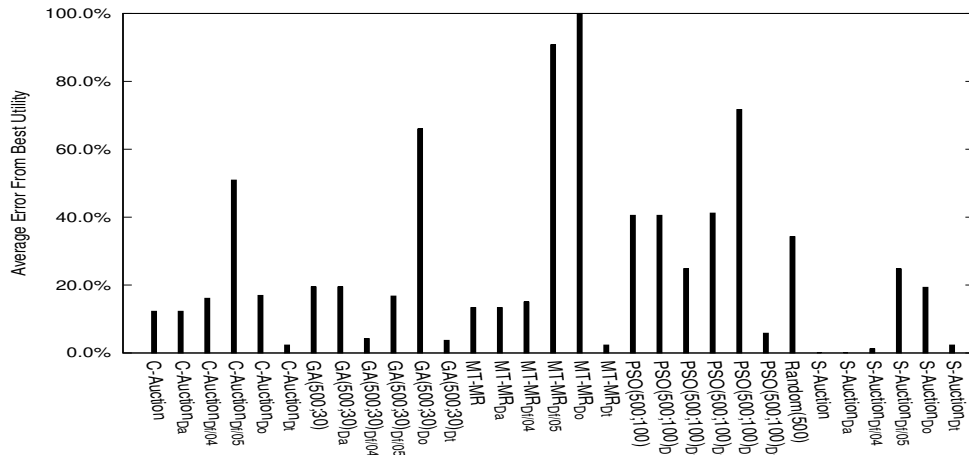
(b) Utility Error.
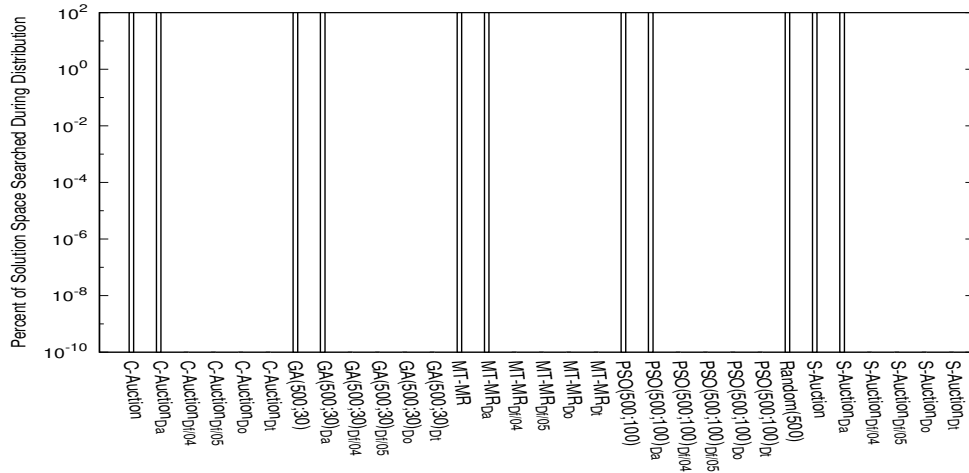
(c) Size Reduction.

Figure A.30: 5×15 ASAR Problem Results.

175

Appendix B. ASAR Problem Repository

This appendix outlines the methods used to generate random autonomous search and recovery (ASAR) problem data for experimental testing of constrained multiagent task scheduling (CMTS) algorithms. A list of sample instances used in the experiments for this research follows this discussion. Characteristics of these problems is identified in Table B.1 below.

ASAR problems are generated based on the number of agents and tasks that are desired. Each agent is given a random subset of abilities from the domain operations with the appropriate constraints placed—disable and transport abilities inhibit each other within an agent, as do observe and monitor. The problem is checked to make sure that every ability is able to perform a task. For example, if only one agent has monitor and observe, then no recover tasks can be performed because the only two effective abilities are inhibiting each other. In this case, other agents are randomly given copies. Thus, the total number of abilities varies for and is no less than any agent size.

Two types of tasks are generated for the ASAR problem: secure and recover. Each secure task is generated with some random number of gunmen and a 2D location. The only requirements are to have a coalition capable of performing disable and monitor operations. There are no prerequisites. Recover tasks complement a secure task by having a single hostages placed at the same location. The secure task used for the location becomes the prerequisite for the recover task. A recovery point is identified for a coalition to observe and transport the hostage. The coalition must also support monitoring of the initial recover location.

For every secure task, there could be several recover tasks. The problem instances used for this experiment ensures that every secure task has at least one complementary recover task, but that the total number of tasks does not exceed the desired problem size. This creates a network of tasks potentially requiring multiple agents to visit and move items while observing the location.

176

Thus, the random generator emulates randomized multiple traveling salesman and vehicle routing problem data with constraints for monitoring events similar to cooperative target tracking (see Sections 3.4.3 and 3.4.4 on how to represent these problems using CMTS). Actual operations research repository data [53,68] can replace the random data if warranted.

Table B.1: ASAR Repository Problem Characteristics.

M×Y	X	Size (LUB)	χ	M×Y	X	Size (LUB)	χ
2×6	4	$6.87 \cdot 10^{10}$	0.1577	4×15	9	$5.45 \cdot 10^{94}$	0.2063
2×10	8	$1.32 \cdot 10^{46}$	0.5939	4×20	7	$3.01 \cdot 10^{142}$	0.3990
2×15	5	$4.48 \cdot 10^{71}$	0.4318	5×6	11	$3.77 \cdot 10^{26}$	0.2749
2×20	8	$2.08 \cdot 10^{150}$	0.7362	5×7	7	$1.72 \cdot 10^{23}$	0.2213
3×6	6	$3.21 \cdot 10^{15}$	0.2347	5×8	9	$8.68 \cdot 10^{34}$	0.4206
3×7	9	$2.65 \cdot 10^{28}$	0.5696	5×9	14	$2.07 \cdot 10^{56}$	0.5625
3×8	9	$8.68 \cdot 10^{34}$	0.5838	5×10	11	$2.15 \cdot 10^{56}$	0.3530
3×9	7	$1.16 \cdot 10^{34}$	0.3293	5×11	11	$5.96 \cdot 10^{63}$	0.3673
3×10	8	$1.32 \cdot 10^{46}$	0.3037	5×12	15	$1.00 \cdot 10^{91}$	0.3748
3×11	9	$3.13 \cdot 10^{57}$	0.4633	5×13	10	$3.23 \cdot 10^{80}$	0.2359
3×12	8	$2.13 \cdot 10^{62}$	0.4588	5×14	11	$3.47 \cdot 10^{96}$	0.3050
3×13	10	$3.06 \cdot 10^{80}$	0.4980	5×15	13	$2.29 \cdot 10^{116}$	0.4358
3×14	9	$2.57 \cdot 10^{86}$	0.3738	5×20	16	$1.38 \cdot 10^{204}$	0.6037
3×15	6	$8.00 \cdot 10^{78}$	0.2770	6×6	16	$3.58 \cdot 10^{36}$	0.2018
3×20	8	$2.08 \cdot 10^{150}$	0.5913	6×10	17	$2.80 \cdot 10^{75}$	0.4176
4×6	10	$3.71 \cdot 10^{24}$	0.3220	6×15	14	$8.15 \cdot 10^{121}$	0.3521
4×7	13	$9.97 \cdot 10^{37}$	0.4517	6×20	14	$2.08 \cdot 10^{188}$	0.3955
4×8	11	$3.70 \cdot 10^{40}$	0.2643	7×6	14	$3.47 \cdot 10^{32}$	0.1759
4×9	12	$4.74 \cdot 10^{50}$	0.5455	7×10	18	$4.32 \cdot 10^{78}$	0.2903
4×10	15	$7.79 \cdot 10^{69}$	0.5157	7×15	21	$5.49 \cdot 10^{153}$	0.5006
4×11	12	$2.54 \cdot 10^{68}$	0.5684	7×20	14	$1.46 \cdot 10^{191}$	0.3556
4×12	10	$3.44 \cdot 10^{68}$	0.4243	8×6	14	$4.53 \cdot 10^{32}$	0.1759
4×13	11	$3.30 \cdot 10^{85}$	0.3402	8×10	16	$2.98 \cdot 10^{72}$	0.1679
4×14	9	$9.72 \cdot 10^{85}$	0.3738	8×15	19	$1.22 \cdot 10^{144}$	0.3976

	Observe	Disable	Transport	Monitor	Position
Agent 1			√	√	(0,-20)
Agent 2	√	√			⋮

	# gunmen	# hostages	
Site 1	5	1	(16,32)
Site 2	1	1	(9,33)
Site 3	4	1	(26,30)

Table B.2: 2×6 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2	√	√	√	√	⋮

	# gunmen	# hostages	
Site 1	3	1	(12,14)
Site 2	2	1	(13,50)
Site 3	3	1	(42,26)
Site 4	5	3	(29,27)

Table B.3: 2×10 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	√		√		(0,-20)
Agent 2		√	√	√	⋮

	# gunmen	# hostages	
Site 1	2	1	(13,50)
Site 2	5	6	(17,30)
Site 3	2	5	(12,18)

Table B.4: 2×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2	√	√	√	√	⋮

	# gunmen	# hostages	
Site 1	3	6	(-38,44)
Site 2	4	7	(26,20)
Site 3	5	4	(10,24)

Table B.5: 2×20 ASAR Problem Instance.

178

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√		√	(0,-20)
Agent 2	√				⋮
Agent 3		√	√		⋮

	# gunmen	# hostages	
Site 1	5	1	(-13,22)
Site 2	4	1	(-8,42)
Site 3	3	1	(-31,18)

Table B.6: 3×6 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3	√				⋮

	# gunmen	# hostages	
Site 1	5	1	(-9,1)
Site 2	3	2	(-9,9)
Site 3	2	1	(-16,14)

Table B.7: 3×7 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√		(0,-20)
Agent 2			√	√	⋮
Agent 3	√	√	√	√	⋮

	# gunmen	# hostages	
Site 1	2	2	(45,26)
Site 2	5	2	(-3,43)
Site 3	4	1	(48,38)

Table B.8: 3×8 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2		√			⋮
Agent 3		√		√	⋮
	# gunmen		# hostages		
Site 1	3		1		(12,34)
Site 2	1		1		(-35,18)
Site 3	3		1		(29,34)
Site 4	2		2		(47,13)

Table B.9: 3×9 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1		√	√		(0,-20)
Agent 2	√			√	⋮
Agent 3	√	√	√	√	⋮
	# gunmen		# hostages		
Site 1	4		1		(-22,1)
Site 2	1		3		(-18,11)
Site 3	5		1		(34,16)
Site 4	5		1		(-12,11)

Table B.10: 3×10 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2		√	√	√	⋮
Agent 3	√		√		⋮
	# gunmen		# hostages		
Site 1	2		5		(-15,24)
Site 2	4		1		(45,9)
Site 3	4		2		(17,19)

Table B.11: 3×11 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1			√		(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3	√		√	√	⋮

	# gunmen		# hostages		
Site 1	1		1		(-2,15)
Site 2	4		1		(30,50)
Site 3	4		5		(10,3)
Site 4	2		1		(-17,6)

Table B.12: 3×12 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2	√		√	√	⋮
Agent 3	√	√	√		⋮

	# gunmen		# hostages		
Site 1	5		2		(24,18)
Site 2	1		1		(21,7)
Site 3	1		5		(33,33)
Site 4	4		1		(14,15)

Table B.13: 3×13 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1			√	√	(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3	√		√	√	⋮

	# gunmen		# hostages		
Site 1	4		1		(-29,11)
Site 2	5		1		(34,16)
Site 3	3		1		(-40,31)
Site 4	1		1		(39,5)
Site 5	2		1		(0,25)
Site 6	2		1		(-5,23)
Site 7	5		1		(20,7)

Table B.14: 3×14 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓		✓		(0,-20)
Agent 2		✓	✓		⋮
Agent 3			✓	✓	⋮
	# gunmen		# hostages		
Site 1	4		1		(39,35)
Site 2	4		3		(-34,0)
Site 3	5		3		(47,12)
Site 4	3		1		(-6,15)
Site 5	2		2		(47,25)

Table B.15: 3×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓	✓		✓	(0,-20)
Agent 2		✓	✓		⋮
Agent 3	✓		✓	✓	⋮
	# gunmen		# hostages		
Site 1	3		10		(2,24)
Site 2	3		6		(28,14)
Site 3	5		1		(-19,19)

Table B.16: 3×20 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓	✓	✓	✓	(0,-20)
Agent 2			✓		⋮
Agent 3	✓		✓		⋮
Agent 4	✓		✓	✓	⋮
	# gunmen		# hostages		
Site 1	4		1		(-49,47)
Site 2	3		1		(-45,24)
Site 3	5		1		(-4,19)

Table B.17: 4×6 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓		✓	✓	(0,-20)
Agent 2		✓	✓		⋮
Agent 3	✓	✓	✓	✓	⋮
Agent 4	✓	✓	✓	✓	⋮
	# gunmen		# hostages		
Site 1	1		1		(11,3)
Site 2	3		2		(-35,25)
Site 3	1		1		(-20,37)

Table B.18: 4 × 7 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1		✓	✓	✓	(0,-20)
Agent 2	✓	✓		✓	⋮
Agent 3		✓	✓		⋮
Agent 4	✓	✓	✓		⋮
	# gunmen		# hostages		
Site 1	3		1		(-12,5)
Site 2	5		1		(39,7)
Site 3	2		1		(-21,23)
Site 4	5		1		(-8,11)

Table B.19: 4 × 8 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓	✓			(0,-20)
Agent 2		✓	✓	✓	⋮
Agent 3	✓	✓	✓		⋮
Agent 4	✓	✓	✓	✓	⋮
	# gunmen		# hostages		
Site 1	4		1		(-33,48)
Site 2	2		2		(-9,21)
Site 3	2		3		(47,28)

Table B.20: 4 × 9 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3	√	√	√	√	⋮
Agent 4	√		√	√	⋮

	# gunmen	# hostages	
Site 1	3	1	(1,19)
Site 2	4	1	(14,10)
Site 3	5	1	(7,10)
Site 4	4	1	(-24,34)
Site 5	3	1	(-24,18)

Table B.21: 4×10 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√	√	(0,-20)
Agent 2	√	√			⋮
Agent 3	√	√	√	√	⋮
Agent 4	√		√		⋮

	# gunmen	# hostages	
Site 1	4	2	(-10,1)
Site 2	5	2	(27,14)
Site 3	2	4	(-18,39)

Table B.22: 4×11 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√	√		(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3		√	√		⋮
Agent 4		√			⋮

	# gunmen	# hostages	
Site 1	1	1	(11,24)
Site 2	3	2	(27,1)
Site 3	5	6	(-39,48)

Table B.23: 4×12 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1		√	√	√	(0,-20)
Agent 2				√	⋮
Agent 3	√	√	√		⋮
Agent 4	√	√	√	√	⋮
	# gunmen		# hostages		
Site 1	3		2		(-35,16)
Site 2	2		1		(36,31)
Site 3	2		1		(-46,40)
Site 4	2		1		(42,26)
Site 5	4		1		(-48,37)
Site 6	2		1		(9,1)

Table B.24: 4 × 13 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√		√	(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3	√				⋮
Agent 4			√		⋮
	# gunmen		# hostages		
Site 1	2		1		(9,44)
Site 2	3		1		(-1,38)
Site 3	5		5		(11,27)
Site 4	4		1		(46,15)
Site 5	2		1		(15,11)

Table B.25: 4 × 14 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓	✓		✓	(0,-20)
Agent 2	✓	✓	✓		⋮
Agent 3		✓		✓	⋮
Agent 4		✓			⋮
	# gunmen		# hostages		
Site 1	1		1		(-44,46)
Site 2	5		1		(-50,14)
Site 3	1		2		(-50,9)
Site 4	3		1		(-31,45)
Site 5	2		1		(-46,33)
Site 6	1		1		(39,23)
Site 7	5		1		(13,8)

Table B.26: 4×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1		✓	✓		(0,-20)
Agent 2	✓				⋮
Agent 3		✓	✓		⋮
Agent 4	✓			✓	⋮
	# gunmen		# hostages		
Site 1	5		2		(34,40)
Site 2	2		3		(39,31)
Site 3	1		1		(2,6)
Site 4	2		3		(-44,34)
Site 5	4		1		(-28,20)
Site 6	5		1		(-50,20)
Site 7	4		2		(44,32)

Table B.27: 4×20 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√		√	√	(0,-20)
Agent 2	√		√		⋮
Agent 3			√		⋮
Agent 4				√	⋮
Agent 5	√	√	√	√	⋮
	# gunmen		# hostages		
Site 1	5		1		(-22,33)
Site 2	5		1		(-44,9)
Site 3	4		1		(-38,25)

Table B.28: 5×6 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1		√			(0,-20)
Agent 2			√		⋮
Agent 3		√	√	√	⋮
Agent 4	√				⋮
Agent 5				√	⋮
	# gunmen		# hostages		
Site 1	1		1		(-7,5)
Site 2	3		2		(-3,24)
Site 3	3		1		(47,12)

Table B.29: 5×7 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√			(0,-20)
Agent 2			√		⋮
Agent 3			√	√	⋮
Agent 4		√			⋮
Agent 5	√		√	√	⋮
	# gunmen		# hostages		
Site 1	4		1		(-12,22)
Site 2	5		2		(34,32)
Site 3	5		2		(-27,27)

Table B.30: 5×8 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√	√			(0,-20)
Agent 2	√	√	√	√	⋮
Agent 3	√	√	√	√	⋮
Agent 4	√	√	√		⋮
Agent 5		√			⋮
	# gunmen		# hostages		
Site 1	3		2		(-45,35)
Site 2	4		3		(18,32)
Site 3	4		1		(9,28)

Table B.31: 5 × 9 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	√				(0,-20)
Agent 2		√		√	⋮
Agent 3	√	√	√	√	⋮
Agent 4	√		√	√	⋮
Agent 5			√		⋮
	# gunmen		# hostages		
Site 1	1		3		(46,23)
Site 2	2		2		(-22,40)
Site 3	3		2		(-20,7)

Table B.32: 5 × 10 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1			√		(0,-20)
Agent 2		√		√	⋮
Agent 3	√	√	√		⋮
Agent 4			√	√	⋮
Agent 5		√	√	√	⋮
	# gunmen		# hostages		
Site 1	1		3		(-9,1)
Site 2	2		1		(47,30)
Site 3	1		4		(6,13)

Table B.33: 5 × 11 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓	✓	✓	✓	(0,-20)
Agent 2		✓			⋮
Agent 3	✓	✓	✓	✓	⋮
Agent 4		✓	✓		⋮
Agent 5	✓	✓	✓	✓	⋮

	# gunmen	# hostages	
Site 1	4	1	(-6,43)
Site 2	2	1	(-43,41)
Site 3	2	1	(-16,37)
Site 4	4	1	(-9,11)
Site 5	1	1	(-24,26)
Site 6	4	1	(46,15)

Table B.34: 5×12 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓			✓	(0,-20)
Agent 2	✓		✓	✓	⋮
Agent 3		✓			⋮
Agent 4	✓	✓	✓		⋮
Agent 5				✓	⋮

	# gunmen	# hostages	
Site 1	5	1	(7,0)
Site 2	2	3	(-33,37)
Site 3	3	2	(-14,40)
Site 4	3	1	(50,48)
Site 5	4	1	(5,11)

Table B.35: 5×13 ASAR Problem Instance.

189

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓				(0,-20)
Agent 2	✓	✓	✓	✓	⋮
Agent 3			✓		⋮
Agent 4		✓			⋮
Agent 5	✓	✓	✓	✓	⋮

	# gunmen	# hostages	
Site 1	5	1	(-28,21)
Site 2	1	1	(-36,5)
Site 3	3	1	(27,8)
Site 4	3	1	(9,20)
Site 5	3	1	(-26,17)
Site 6	2	1	(-45,30)
Site 7	3	1	(36,42)

Table B.36: 5×14 ASAR Problem Instance.

	Transport	Disable	Monitor	Observe	Position
Agent 1	✓	✓	✓	✓	(0,-20)
Agent 2	✓				⋮
Agent 3	✓	✓			⋮
Agent 4	✓	✓	✓	✓	⋮
Agent 5	✓		✓		⋮

	# gunmen	# hostages	
Site 1	5	3	(27,41)
Site 2	3	1	(17,40)
Site 3	3	3	(-27,16)
Site 4	1	2	(-38,3)
Site 5	4	1	(22,37)

Table B.37: 5×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓			✓	(0,-20)
Agent 2	✓	✓	✓	✓	⋮
Agent 3	✓	✓	✓	✓	⋮
Agent 4		✓		✓	⋮
Agent 5	✓	✓	✓	✓	⋮
	# gunmen		# hostages		
Site 1	1		10		(-43,9)
Site 2	5		3		(25,4)
Site 3	1		1		(-6,16)
Site 4	4		2		(-7,28)

Table B.38: 5×20 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓	✓		✓	(0,-20)
Agent 2			✓	✓	⋮
Agent 3			✓	✓	⋮
Agent 4	✓		✓	✓	⋮
Agent 5		✓	✓	✓	⋮
Agent 6	✓	✓		✓	⋮
	# gunmen		# hostages		
Site 1	1		1		(-43,30)
Site 2	2		1		(-44,23)
Site 3	4		1		(-9,5)

Table B.39: 6×6 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1		✓			(0,-20)
Agent 2	✓	✓	✓		⋮
Agent 3	✓	✓	✓	✓	⋮
Agent 4	✓		✓	✓	⋮
Agent 5		✓		✓	⋮
Agent 6	✓	✓	✓	✓	⋮
	# gunmen		# hostages		
Site 1	3		2		(-3,50)
Site 2	5		4		(21,0)
Site 3	3		1		(44,16)

Table B.40: 6×10 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓		✓	✓	(0,-20)
Agent 2		✓			⋮
Agent 3		✓	✓		⋮
Agent 4	✓	✓	✓	✓	⋮
Agent 5	✓	✓	✓		⋮
Agent 6				✓	⋮
	# gunmen		# hostages		
Site 1	4		1		(0,36)
Site 2	4		1		(-17,48)
Site 3	3		1		(-27,26)
Site 4	1		3		(-38,3)
Site 5	2		2		(-36,3)
Site 6	5		1		(-44,45)

Table B.41: 6×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1			✓		(0,-20)
Agent 2			✓	✓	⋮
Agent 3	✓	✓	✓	✓	⋮
Agent 4		✓	✓	✓	⋮
Agent 5		✓	✓	✓	⋮
Agent 6				✓	⋮
	# gunmen		# hostages		
Site 1	4		1		(-9,40)
Site 2	4		7		(-22,25)
Site 3	1		6		(42,23)
Site 4	4		2		(-36,6)

Table B.42: 6×20 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓				(0,-20)
Agent 2		✓		✓	⋮
Agent 3		✓			⋮
Agent 4		✓	✓	✓	⋮
Agent 5		✓		✓	⋮
Agent 6	✓	✓	✓	✓	⋮
Agent 7				✓	⋮
	# gunmen		# hostages		
Site 1	2		1		(39,25)
Site 2	1		1		(-50,38)
Site 3	2		1		(20,8)

Table B.43: 7×6 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓	✓		✓	(0,-20)
Agent 2	✓				⋮
Agent 3		✓		✓	⋮
Agent 4			✓	✓	⋮
Agent 5	✓		✓	✓	⋮
Agent 6	✓		✓	✓	⋮
Agent 7	✓	✓	✓	✓	⋮
	# gunmen		# hostages		
Site 1	3		4		(23,30)
Site 2	5		1		(29,32)
Site 3	3		2		(47,9)

Table B.44: 7×10 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓	✓		✓	(0,-20)
Agent 2		✓			⋮
Agent 3	✓	✓	✓		⋮
Agent 4	✓	✓	✓	✓	⋮
Agent 5	✓	✓	✓	✓	⋮
Agent 6	✓		✓	✓	⋮
Agent 7	✓	✓	✓		⋮
	# gunmen		# hostages		
Site 1	3		3		(-23,13)
Site 2	4		4		(5,24)
Site 3	5		5		(-32,23)

Table B.45: 7×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓		✓	✓	(0,-20)
Agent 2	✓			✓	⋮
Agent 3				✓	⋮
Agent 4		✓		✓	⋮
Agent 5	✓		✓	✓	⋮
Agent 6	✓		✓		⋮
Agent 7	✓				⋮
	# gunmen		# hostages		
Site 1	5		1		(26,16)
Site 2	2		2		(33,15)
Site 3	1		14		(-15,27)

Table B.46: 7×20 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	✓			✓	(0,-20)
Agent 2	✓		✓	✓	⋮
Agent 3		✓			⋮
Agent 4		✓			⋮
Agent 5		✓		✓	⋮
Agent 6	✓	✓	✓		⋮
Agent 7		✓			⋮
Agent 8				✓	⋮
	# gunmen		# hostages		
Site 1	5		1		(-38,11)
Site 2	5		1		(38,31)
Site 3	4		1		(-18,12)

Table B.47: 8×6 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	√		√		(0,-20)
Agent 2		√			⋮
Agent 3			√		⋮
Agent 4	√		√		⋮
Agent 5	√		√	√	⋮
Agent 6	√	√		√	⋮
Agent 7		√	√	√	⋮
Agent 8	√				⋮
	# gunmen		# hostages		
Site 1	1		1		(2,35)
Site 2	1		2		(-9,23)
Site 3	2		1		(35,16)
Site 4	4		2		(-16,21)

Table B.48: 8×10 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1	√			√	(0,-20)
Agent 2	√	√	√		⋮
Agent 3	√	√			⋮
Agent 4	√				⋮
Agent 5	√	√	√	√	⋮
Agent 6	√	√	√	√	⋮
Agent 7	√				⋮
Agent 8	√		√		⋮
	# gunmen		# hostages		
Site 1	3		9		(9,10)
Site 2	3		2		(20,35)
Site 3	5		1		(3,24)

Table B.49: 8×15 ASAR Problem Instance.

	Observe	Disable	Transport	Monitor	Position
Agent 1			√		(0,-20)
Agent 2	√	√	√		⋮
Agent 3	√		√	√	⋮
Agent 4	√	√	√	√	⋮
Agent 5		√	√		⋮
Agent 6	√	√	√	√	⋮
Agent 7	√	√		√	⋮
Agent 8			√		⋮

	# gunmen	# hostages	
Site 1	3	7	(-13,41)
Site 2	5	4	(-13,23)
Site 3	2	6	(48,6)

Table B.50: 8×20 ASAR Problem Instance.

Bibliography

1. Ahmadabadi, Majid Nili and Masoud Asadpour. "Expertness based cooperative Q-learning". *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(1):66–76, 2000.

2. Ahmadabadi, Majid Nili, Masoud Asadpour, Seyyed H. Khodaabakhsh, and Eiji Nakano. "Expertness Measuring in Cooperative Learning". *IEEE/RSJ International Conference On Intelligent Robots and Systems (IROS 2000)*, volume 3, 2261–2267. IEEE, 2000.

3. Ambroszkiewicz, Stanislaw, Krzyszof Cetnarowicz, and Wojciech Turek. "Multi-robot Management Framework Based on the Agent Dual-Space Control Paradigm". *AAAI RIDIS Workshop 2007*. 2007.

4. Apt, K. R. and T. Radzik. "Stable partitions in coalitional games". *ArXiv Computer Science e-prints*, May 2006.

5. Arai, Sachiyo, Katia Sycara, and Terry R. Payne. "Multi-Agent Reinforcement Learning for Planning and Scheduling Multiple Goals". *Fourth International Conference on Multi-Agent Systems*, 00:0359, 2000.

6. Bäck, Thomas, David B. Fogel, and Zbigniew Michalewicz. *Evolutionary Computation*, volume 1 - 2. Institute of Physics Publishing, 2000.

7. Bagchi, Tapan P. *Multi-objective Scheduling By Genetic Algorithms*. Kluwer, 1999.

8. Banerjee, Bikramjit and Jing Peng. "Adaptive policy gradient in multiagent learning". *2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 686–692. ACM Press, 2003.

9. Beasley, J. E. "OR-Library: distributing test problems by electronic mail". *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

10. Bernstein, D. S., E. Hansen, and S. Zilberstein. "Bounded policy iteration for decentralized POMDPs". *In Proceedings of the Nineteenth International Joint Conf. on Artificial Intelligence*. 2005.

11. Bonabeau, Eric, Marco Dorigo, and T. Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

12. Borzello, Eric and Laurence D. Merkle. "Multi-agent Cooperation Using Ant Algorithm with Variable Pheromone Placement". *IEEE Congress on Evolutionary Computation*, volume 2, 1232–1237. IEEE, 2005.

13. Botelho, S. and R. Alami. "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement". *IEEE International Conference on Robotics and Automation*, 2:1234–1239, 1999.

14. Botelho, S. and R. Alami. "A Multi-Robot Cooperative Task Achievement System". *IEEE International Conference on Robotics and Automation*, 2716–2721. San Francisco, USA, 2000.

15. Bowling, Michael H. *Multiagent Learning in the Presence of Agents with Limitations*. Ph.D. thesis, Carnegie Mellon University, 2003.

16. Bowling, Michael H. and Manuela M. Veloso. "Multiagent learning using a variable learning rate". *Artificial Intelligence*, 136(2):215–250, 2002.

17. Bradley, Jay and Gillian Hayes. "Adapting Reinforcement Learning for Computer Games: Using Group Utility Functions". *IEEE Symposium on Computational Intelligence and Games*, 133–140, 2005.

18. Burkard, R.E. and F. Rendl. "QAPLIB–A Quadratic Assignment Problem Library". *European Journal of Operational Research*, 55:115–119, 1991.

19. "Cambridge Advance Learner's Dictionary". Http://dictionary.cambridge.org/.

20. Chalkiadakis, Georgios and Craig Boutilier. "Bayesian Reinforcement Learning for Coalition Formation Under Uncertainty". *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 03:1090–1097, 2004.

21. Clarke, G. and J. W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Operations Research*, 12(4):568–581, 1964.

22. Claus, Caroline and Craig Boutilier. "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems". *AAAI-97 Workshop on Multiagent Learning*, 746–752. 1998.

23. Connelly, D. T. "General Purpose Simulated Annealing". *Operations Research*, 34(3):495–505, 1992.

24. Couśin, Kevin and Gilbert L. Peterson. "Cooperative Reinforcement Learning Using An Expert-Measuring Weighted Strategy With WoLF". *IASTED Conference on Artificial Intelligence and Soft Computing*, 165–170. 2005.

25. Couśin, Kevin and Gilbert L. Peterson. "A Distributed Approach To Solving Constrained Multiagent Task Scheduling Problems". *Proc. of the 2007 AAAI Fall Symposium on Regarding the "Intelligence" in Distributed Intelligent Systems*, 42–48. November 2007.

26. Couśin, Kevin, Gilbert L. Peterson, Christopher B. Mayer, and Gary B. Lamont. "WoLF Ant", 2006. Submit to IEEE Transactions on Evolutionary Computation.

27. Dang, Viet Dung and Nick R. Jennings. "Generating coalition structures with finite bound from the optimal guarantees". *Proceedings of the Third International Conference on Autonomous Agents and Multi-Agent Systems*, 564–571, 2004.

28. Dang, Viet Dung and Nick R. Jennings. "Coalition structure generation in task-based settings". *Proc 17th European Conference on AI*, 2006.

29. Dorigo, Marco and Thomas Stützle. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, chapter The ant colony optimization metaheuristic: Algorithms, applications, and advances, 251–285. Kluwer Academic Publishers, 2002.

30. Dorigo, Marco and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004.

31. Dudek, Gregory, Michael R. Jenkin, Evangelos Milios, and David Wilkes. *A Taxonomy for Multi-Agent Robotics*, volume 3, 375–397. Kluwer Academic Publishers, December 1996.

32. Duvivier, D., Ph. Preux, and E-G. Talbi. "Stochastic Algorithms for Optimization and Application to Job Shop Scheduling", 1995.

33. Engelbrecht, Andries P. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2006.

34. Eshgh, Sahar Mastour and Majid Nili Ahmadabadi. "An Extension of Weighted Strategy Sharing in Cooperative Q-Learning for Specialized Agents". *9th International Conference on Neural Information Processing (ICONIP'02)*, volume 1, 106–110. IEEE, 2002.

35. Farinelli, Alessandro, Luca Iocchi, and Daniele Nardi. "Multirobot systems: A classification focused on coordination". *IEEE Trasactions on Systems, Man and Cybernetics*, 34(5):2015–2028, October 2004.

36. Gage, Aaron. *Multi-Robot Task Allocation Using Affect*. Ph.D. thesis, University of South Florida, 2004.

37. Gage, Aaron, Robin Murphy, Kimon Valavanis, and Matt Long. "Affective Task Allocation for Distributed Multi-Robot Teams". *Nineteenth National Conference on Artificial Intelligence*, 98–999. 2004.

38. Gale, D. and L. S. Shapley. "College Admissions and the Stability of Marriage". *American Mathematical Monthly*, 69:9–14, 1962.

39. Gale, David. *The Theory of Linear Economic Models*. McGraw Hill, 1960.

40. Galstyan, Aram, Tad Hogg, and Kristina Lerman. "Modeling and Mathematical Analysis of Swarms of Microscopic Robots". *IEEE Swarm Intelligence Symposium*, 2005.

41. Gardiol, Natalia H. and Leslie Pack Kaelbling. "Envelope-based Planning in Relational MDPs". Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf (editors), *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

42. Gerkey, Brian P. *On Multi-robot Task Allocation*. Ph.D. thesis, University of Southern California, 2003.

43. Gerkey, Brian P. and Maja J. Matarić. "MURDOCH: Publish/Subscribe Task Allocation for Heterogeneous Agents". *Autonomous Agents 2000*, 203–204. Barcelona, Spain, June 3 - 7 2000.

44. Gerkey, Brian P. and Maja J. Matarić. "A framework for studying multirobot task allocation". A. C. Schultz et al. (editor), *Proceedings of the 2nd International Naval Research Laboratory Workshop on Multi-Robot Systems*. NRL, Washington D.C., March 17 - 19 2003.

45. Gerkey, Brian P. and Maja J. Matarić. "A formal analysis and taxonomy of task allocation in multi-robot systems". *International Journal of Robotics Research*, 23(9):939–954, 2004.

46. Gerkey, Brian P. and Maja J. Matarić. "Sold!: Auction methods for multi-robot coordination". *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, 18(5):758–768, October 2004.

47. Glover, Fred and M. Laguna. "Tabu Search". C. Reeves (editor), *Modern Heuristic Techniques for Combinatorial Problems*, 70–141. Blackwell Scientific Publishing, Oxford, England, 1993.

48. Gmytrasiewicz, Piotr J. and Prashant Doshi. "Framework for Sequential Planning in Multi-Agent Settings". *Journal of AI Research*, 2005.

49. Goldberg, Dani and Maja J. Matarić. *Design and Evaluation of Robust Behavior-Based Controllers for Distributed Multi-Robot Collection Tasks*, 315–244. AK Peters, Ltd., 2001.

50. Gong, Tao and Andrew L. Tusonć. "Particle Swarm Optimization For Quadratic Assignment Problems". *International Journal of Computational Intelligence Research*, 2, 2006.

51. Hartmann, S. and Rainer Kolisch. "Experimental evaluation of state-of-the-art heuristics for resource constrained project scheduling". *European Journal of Operation Research*, 127(2):394–407, 2000.

52. Hayes, A.T. and P. Dormiani-Tabatabaei. "Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots". *IEEE Conference on Robotics and Automation*, 4:3900–3905, 2002.

53. University of Heidelberg, Department of Computer Science. "TSPLIB", 2006. Available on-line at: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95.

54. Hertz, Alain and Marino Widmer. "An Improved Tabu Search Approach for Solving the Job Shop Scheduling Problem with Tooling Constraints." *Discrete Applied Mathematics*, 65(1-3):319–345, 1996.

55. Hoey, Jesse, Robert St-Aubin, Alan Hu, and Craig Boutilier. "SPUDD: Stochastic planning using decision diagrams". *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288. 1999.

56. Hollander, M. and D. A. Wolfe. *Nonparametric Statistical Methods*. Wiley, 2nd edition, 1999.

57. Hoos, Holger H. and Thomas Stützle. *Stochastic Local Search: Foundations and Application*. Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann/Elsevier, 2004.

58. Jäger, M. and B. Nebel. "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots". *Int. Conf. on Intelligent Robots and Systems (IROS)*. Maui, Hawaii, 2001.

59. Jose B. Cruz, Jr, Genshe Chen, Dongxu Li, and Xu Wang. "Particle Swarm Optimization for Resource Allocation in UAV Cooperative Control". *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 1–11, 2004.

60. Kaelbling, Leslie P., Michael L. Littman, and Anthony R. Cassandra. "Planning and Acting in Partially Observable Domains". *Artificial Intelligence*, 101:99 – 134, 1998.

61. Kennedy, J. and R. C. Eberhart. "Particle Swarm Optimization". *IEEE International Conference on Neural Networks*, 1942–1948, 1995.

62. Kitano, H., S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S.ć Shimada. "RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomout Agents Research". *IEEE Systems, Man, and Cybernetics*, 6:739–743, 1999.

63. Klusch, Matthias and Andreas Gerber. "Dynamic coalition formation among rational agents". *IEEE Intelligent Systems*, 17(3):42–47, 2002.

64. Koes, Mary, Illah Nourbakhsh, and Katia Sycara. "Communication Efficiency in Multi-agent Systems". *Proceedings of ICRA 2004*. May 2004.

65. Kok, Jelle R., Pieter Jan't Hoen, Bram Bakker, and Nikos A. Vlassis. "Utile Coordination: Learning Interdependencies Among Cooperative Agents". *CIG*. IEEE, 2005.

66. Kolisch, Rainer and S. Hartmann. *Heuristic algorithms for solving the resource-constrained project scheduling problem - Classification and computational analysis*, 147–178. Kluwer, 1999.

67. Kolisch, Rainer and S. Hartmann. "Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update". *European Journal of Operation Research*, 174(1):23–27, 2005.

68. Kolisch, Rainer, C. Schwindt, and Arno Sprecher. *Benchmark instances for project scheduling problems*, 197–212. Kluwer, 1999.

69. Kolisch, Rainer and Arno Sprecher. "PSPLIB - A Project Scheduling Problem Library". *European Journal of Operation Research*, 96:205–216, 1996.

70. Kolling, Andreas and Stefano Carpin. "Multirobot Cooperation for Surveillance of Multiple Moving Targets - A New Behavioral Approach". *IEEE International Conference on Robotics and Automation*, 1311–1316, 2006.

71. Kraus, S., O. Shehory, and G. Taase. "Coalition formation with uncertain heterogeneous information". *Proceedings of AAMAS*, 1–8. 2003.

72. Krishna, K Madhava, Henry Hexmoor, and Shravan Soganić. "A T-Step Ahead Optimal Target Detection Algorithm for a Multi Sensor Surveillance System". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1840–1845, 2005.

73. Kuhn, Harold W. ""The Hungarian Method for the assignment problem"". *Naval Research Logistic Quarterly*, 2:83–97, 1955.

74. Kumar, Vipin. "Algorithms for Constraint-Satisfaction Problems: A Survey". *AI Magazine*, 13(1):32–44, 1992.

75. Lau, Hoong Chuin and Lei Zhang. "Task Allocation via Multi-Agent Coalition Formation: Taxonomy, Algorithms and Complexity". *15th IEEE International Conference on Tools with Artificial Intelligence*, 346, 2003.

76. LaValle, Steven M. *Planning Algorithms*. Cambridge University Press, 2006.

77. Lawrence, S. "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)", 1984.

78. Lecoutre, Christophe. "Benchmarks 2.0. XML representation of CSP instances", 2007. Online. http://www.cril.univ-artois.fr/ lecoutre/research/benchmarks/benchmarks.html.

79. Lerman, Kristina, Chris Jones, Aram Galstyan, and Maja J. Matarić. "Analysis of Dynamic Task Allocation in Multi-Robot Systems". *International Journal of Robotics Research*, 25(3):225–241, 2006.

80. Li, Cuihong and Katia Sycara. "Algorithms for Combinatorial Coalition Formation and Payoff Division in an Electronic Marketplace". *First International Joint Conference on Autonomous Agents and Multiagent Systems(AAMAS)*, 120–127. 2001.

81. Li, Cuihong and Katia Sycara. *A stable and efficient scheme for task allocation via agent coalition formation*, 193–211. World Scientific, 2004.

82. Littman, M. L. "Markov games as a framework for multi-agent reinforcement learning". *11th International Conference on Machine Learning*, 157–163. Morgan Kaufmann, 1994.

83. Liu, C. L. and James W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment". *Journal of the ACM*, 20(1):46–61, 1973.

84. Luke, Sean, Keith Sullivan, Gabriel Catalin Balan, and Liviu Panait. *Tunably Decentralized Algorithms for Cooperative Target Observation*. Technical report,

Department of Computer Science, George Mason University, 4400 University Drive MS 4A5, Fairfax, VA 22030-4444 USA, 2004.

85. Mailler, Roger and Victor Lesser. "Asynchronous Partial Overlay: A New Algorithm for Solving Distributed Constraint Satisfaction Problems". *Journal of Artificial Intelligence Research*, 25:529–576, April 2006.

86. Maniezzo, V., A. Colorni, and Marco Dorigo. *The Ant System applied to the Quadratic Assignment Problem*. Technical report, Université Libre de Bruxelles, 1994.

87. Merkle, Daniel, Martin Middendorf, and Hartmut Schmeck. "Ant Colony Optimization for Resource-Constrained Project Scheduling". Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer (editors), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 893–900. Morgan Kaufmann, Las Vegas, Nevada, USA, 10-12 2000.

88. "Marriam-Webster's Online Dictionary". Http://www.merriam-webster.com/dictionary/framework.

89. Michalewicz, Zbigniew and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2nd edition, 2004.

90. Mitrovic-Minic, Snezana and Ramesh Krishnamurti. "The multiple traveling salesman problem with time windows: vehicle bounds based on precedence graphs". *Operations Research Letters*, 34(1):111–120, 2006.

91. Miyata, Natsuki, Jun Ota, Tamio Arai, and Hajime Asama. "Cooperative Transport by Multiple Mobile Robots in Unknown Static Environments Associated With Real-Time Task Assignment". *IEEE Trans. on Robotics and Automation*, 18(5):769–780, 2002.

92. Modi, Pragnesh Jay, Hyuckchul Jung, Milind Tambe, Wei-Min Shen, and Shriniwas Kulkarni. "Dynamic Distributed Resource Allocation: A Distributed Constraint Satisfaction Approach". John-Jules Meyer and Milind Tambe (editors), *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages*, 181–193. 2001.

93. Modi, Pragnesh Jay, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. "Solving Distributed Constraint Optimization Problems Optimally, Efficiently and Asynchronously". *Proceedings of AAMAS*. 2003.

94. Murphy, Robin Roberson, Christine Laetitia Lisetti, Russell Tardif, Liam Irish, and Aaron Gage. "Emotion-Based Control of Cooperating Heterogeneous Mobile Robots". *IEEE Trans. on Robotics and Automation*, 18(5):744–757, 2002.

95. Nair, Ranjit, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. "Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs". *Proceedings of AAAI*. 2005.

96. Parker, Lynne E. "ALLIANCE: An architecture for fault-tolerant multi-robot cooperation". *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

97. Parker, Lynne E. "Cooperative robotics for multi-target observation". *Intelligent automation and soft computing*, 5:5–19, 1999.

98. Parr, Ronald and Stuart Russell. "Reinforcement Learning with Hierarchies of Machines". Michael I. Jordan, Michael J. Kearns, and Sara A. Solla (editors), *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.

99. Pineau, Joelle, Geoffrey Gordon, and Sebastian Thrun. "Point-based value iteration: An anytime algorithm for POMDPs". *International Joint Conference on Artificial Intelligence (IJCAI)*, 1025–1032. August 2003.

100. "Princeton WordNet Search". Http://wordnet.princeton.edu/perl/webwn.

101. Pugh, Jim and Alcherioć Martinoli. "MultiRobot Learning with Particle Swarm Optimization". *International Conference on Autonomous Agents and Multiagent Systems*, 441–448, 2006.

102. Rainer Kolischa, Sonke Hartmannbć. "Experimental investigation of heuristics for resource-constrained project scheduling: An update". *European Journal of Operational Research*, 174:23–37, 2006.

103. Russell, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

104. Salman, Ayed, Imtiaz Ahmad, and Sabah Al-Madani. "Particle swarm optimization for task assignment problem". *Microprocessors and Microsystems*, 26(8):363–371, 2002.

105. Sandholm, Tuomas W. and Victor R. Lesser. "Coalitions Among Computationally Bounded Agents". *Artificial Intelligence*, 94(1-2):99–137, 1997.

106. Sannon, C. E. "Programming a Computer for Playing Chess". *Philosophical Magazine*, 41:256–275, 1950.

107. Shehory, Onn and Sarit Kraus. "Methods for Task Allocation via Agent Coalition Formation". *Artificial Intelligence*, 101(1–2):165–200, 1998.

108. Sigdel, K, K.L.M. Bertels, B Pourebrahimi, S. Vassiliadis, and L.S Shuai. "A framework for Adaptive Matchmaking in Distributed Computing". *proceeding of GRID Workshop Cracow-04*. December 2004.

109. Sinnen, Oliver, Leonel Augusto Sousa, and Frode Eika Sandnes. "Toward a Realistic Task Scheduling Model". *IEEE Transactions on Parallel and Distributed Systems*, 17(3):263–275, 2006.

110. Sondik, Edward J. *The Optimal Control of Partially Observable Markov Processes.* Ph.D. thesis, Stanford University, 1971.

111. Stone, Peter and Richard S. Sutton. "Scaling Reinforcement Learning toward RoboCup Soccer". *Proceedings of the Eighteenth International Conference on Machine Learning*, 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

112. Sutton, Richard S. and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

113. Sycara, K., J. A. Giampapa, B.K. Langley, and M. Paolucci. *The RETSINA MAS, a Case Study*, volume LNCS 2603, 232–250. Springer-Verlag, July 2003.

114. Taillard, E. "Benchmarks for basic scheduling problems". *European Journal of Operations Research*, 64:278–285, 1993.

115. Torbjørn S. Dahl, Maja J Matarić and Gaurav S. Sukhatme. *Complex Engineering Systems*, chapter A machine learning method for improving task allocation in distributed multi-robot transportation. Perseus Books, 2004.

116. Toth, P. and D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.

117. Verma, Deepak and Rajesh Rao. *Graphical Models for Planning and Imitation in Uncertain Environments*. Technical Report 2005-02-01, Department of CSE, University of Washington, 2005.

118. Vidal, René, Omid Shakernia, H. Jim Kim, Hyunchul Shim, and Shankar Sastry. "Multi-Agent Probabilistic Pursuit-Evasion Games with Unmanned Ground and Aerial Vehicles". *IEEE Trans. on Robotics and Automation*, 18(5):662–669, 2002.

119. Watkins, Christopher J.C.H. and Peter Dayan. "Q-learning". *Machine Learning*, 8:279–292, 1992.

120. Wegner, Peter. "Why interaction is more powerful than algorithms". *Commun. ACM*, 40(5):80–91, 1997.

121. Werger, Barry B. and Maja J. Matarić. *Broadcast of local eligibility for multitarget observation*, volume 4, 347–356. New York: Springer-Verlag, 2000.

122. Wolpert, David H. and William G. Macready. "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

123. Wu, Ke, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. "A Simple Model to Generate Hard Satisfiable Instances". *Proc. of 19th International Joint Conference on Artificial Intelligence*, 337–342, 2005.

124. Wu, Ke, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. "Random Constraint Satisfaction: Easy Generation of Hard (Satisfiable) Instances". *Artificial Intelligence*, 171:514–534, 2007.

125. Yakoo, Makoto. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-agent Systems*. Springer Series on Agent Technology, 1st edition, 2001.

126. Yang, Erfu and Dongbing Gu. "Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey". *IEEE Symposium on Computational Intelligence and Games*, 292–299, 2005.

127. Yin, Peng-Yeng, Shiuh-Sheng Yu, Pei-Pei Wang, and Yi-Te Wangć. "A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems". *Computer Standards and Interfaces*, 28:441450, 2005.

128. Yin, Peng-Yeng, Shiuh-Sheng Yu, Pei-Pei Wang, and Yi-Te Wangć. "Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization". *Journal of Systems and Software*, 80(5):724–735, 2007.

129. Yokoo, Makoto, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms". *Knowledge and Data Engineering*, 10(5):673–685, 1998.

130. Zlot, Robert Michael and Anthony (Tony) Stentz. "Complex Task Allocation For Multiple Robots". *Proceedings of the International Conference on Robotics and Automation*. IEEE, April 2005.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
18–12–2007	Doctoral Dissertation	Sept 2004 — Dec 2007

4. TITLE AND SUBTITLE

A Unified Framework For Solving
Multiagent Task Assignment Problems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Cousín, Kevin, Major, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/DCS/ENG/08-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

1) AFRL/SN, Attn: Mr. Jacob Campbell, 2241 Avionicx Circle,
Wright-Patterson Air Force Base, OH 45433, DSN 785-6127 x4154
(jacob.campbell@wpafb.af.mil)
2) DAGSI, Attn: Dr. Elizabeth Downie, 3155 Research Blvd., Ste. 205,
Kettering, Ohio 45420; Comm 937-781-4000 (edownie@dagsi.org)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT This research presents a unified approach to representing and solving the multiagent task assignment problem for complex problem domains using ideas central to multiagent task allocation, project scheduling, constraint satisfaction, and coalition formation, forming the basis of the constrained multiagent task scheduling (CMTS) problem. The CMTS descriptor represents a wide range of classical and modern problems, such as job shop scheduling, the traveling salesman problem, vehicle routing, and cooperative multi-object tracking. Problems using the CMTS representation are solvable by a suite of algorithms ranging from simple random scheduling to state-of-the-art biologically inspired approaches incorporating evolutionary algorithms, dynamic coalition formation, auctioning, and behavior-based robotics to highlight different solution generation strategies. The framework includes a distributed process to show how to scale adapted algorithms to solve increasingly larger domain problems. This approach introduces several methods for problem decomposition and recomposition without significantly compromising solution quality. Decomposition techniques show methods to reduce the search space by several orders of magnitude allowing for improved search efficiency.

15. SUBJECT TERMS

artificial intelligence, multiagent systems, distributed data processing, learning machines

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson
U	U	U	UU	232	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255–6565, ext 4281 (gpeterson@afit.edu)

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18