Decentralized Planning for Autonomous Agents Cooperating in Complex Missions

by

Andrew Whitten

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Aeronautics and Astronautics
August 13, 2010

Certified by .....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Professor Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
SEP 2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Decentralized Planning for Autonomous Agents Cooperating in Complex Missions	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Massachusetts Institute of Technology,Cambridge,MA,02139**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Planning for autonomous vehicles remains an important and challenging research topic. This thesis focuses on decentralized planning for autonomous agents performing complex missions. These types of missions often involve a set of tasks, each representing a component of the mission. Task planning algorithms may be used as part of the mission planner to assign agents to tasks; however, the decentralized task assignment problem becomes increasingly difficult when there exists coupling in the task set. Coupling may be in the form of assignment relationships, where the value of a task is condition on whether or not another task has been assigned, or temporal relationships where the value of a task is conditioned on when it is performed relative to other tasks. In this work, task coupling is treated as a constraint, and a task planning framework is introduced which is specifically designed to ensure that all coupled constraints are satisfied by the assignment. The new algorithm is developed from a baseline Consensus-Based Bundle Algorithm (CBBA) and is called Coupled- Constraint CBBA, or CCBBA. The new algorithm is compared to the baseline in a complex mission simulation and is found to outperform the baseline by up to a factor of 3 with respect to assignment score. A separate extension to CBBA is also developed for satisfying refuel constraints in the task assignment process, and the technique is verified through numerical simulation. Lastly, this thesis examines the autonomous search problem where a fleet of sensorequipped agents are deployed to find objects of interest in an environment. A local search strategy developed for the Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance (OPS-USERS) program is described which is a receding horizon optimization technique. A global search strategy is also described which extends the planning horizon of the local search strategy by incorporating larger amounts of information into the planning. The two strategies are compared both with and without an artificially simulated human operator, and the global search strategy is shown to outperform the local search strategy in terms of number of targets found.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**125**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Decentralized Planning for Autonomous Agents Cooperating in Complex Missions

by

Andrew Whitten

Submitted to the Department of Aeronautics and Astronautics
on August 13, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Planning for autonomous vehicles remains an important and challenging research topic. This thesis focuses on decentralized planning for autonomous agents performing complex missions. These types of missions often involve a set of tasks, each representing a component of the mission. Task planning algorithms may be used as part of the mission planner to assign agents to tasks; however, the decentralized task assignment problem becomes increasingly difficult when there exists coupling in the task set. Coupling may be in the form of assignment relationships, where the value of a task is condition on whether or not another task has been assigned, or temporal relationships where the value of a task is conditioned on when it is performed relative to other tasks. In this work, task coupling is treated as a constraint, and a task planning framework is introduced which is specifically designed to ensure that all coupled constraints are satisfied by the assignment. The new algorithm is developed from a baseline *Consensus-Based Bundle Algorithm* (CBBA) and is called Coupled-Constraint CBBA, or CCBBA. The new algorithm is compared to the baseline in a complex mission simulation and is found to outperform the baseline by up to a factor of 3 with respect to assignment score. A separate extension to CBBA is also developed for satisfying refuel constraints in the task assignment process, and the technique is verified through numerical simulation.

Lastly, this thesis examines the autonomous search problem where a fleet of sensor-equipped agents are deployed to find objects of interest in an environment. A local search strategy developed for the Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance (OPS-USERS) program is described, which is a receding horizon optimization technique. A global search strategy is also described which extends the planning horizon of the local search strategy by incorporating larger amounts of information into the planning. The two strategies are compared both with and without an artificially simulated human operator, and the global search strategy is shown to outperform the local search strategy in terms of number of targets found.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank my advisor, Professor Jonathan How, for the opportunity to come to MIT and contribute to the technologies being developed by the Aerospace Controls Laboratory. I would also like to thank the members of the ACL for helping me in numerous way, and making my time here enjoyable: Alborz Geramifard, Albert Wu, Justin Teo, Sameera Ponda, Vishnu Desaraju, Georges Aoude, Buddy Michini, Dan Levine, Josh Redding, Aditya Undurti, Frank Fan, Luke Johnson, Brandon Luders, Kenneth Lee, Frant Sobolic, Cameron Fraser, Karl Kulling, Brett Bethke, Luca Bertuccelli, Sergio Cafarelli, Han-Lim Choi, and Kathryn Fischer. I also want to thank Olivier Toupet for helping me in my research during both OPS-USERS, and Sensor Fusion.

I cannot thank my family enough for their support and encouragement: my mom Eileen; my dad Daniel; and my sister Emily. A special thanks goes to my wife, Dana, for her patience, understanding, and faith. I could not have done this without you.

This work was funded in part by: ONR # N00014-08-C-0707, AFOSR # FA9550-08-1-0086, and AFRL # FA8750-10-C-0107.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

$\boldsymbol{\nu}_i$	List of number of iterations in constraint violation for agent i
$\boldsymbol{\tau}_i$	Times: list of execution times corresponding to the tasks in plan $\mathbf{p}_i$
$\Delta(A, B)$	Time required to travel from task A to task B
$\mathbf{b}_i$	Bundle: list of tasks agent i has selected, descending order of score
$\mathbf{p}_i$	Path: list of tasks agent i has selected, in order they are to be executed
$\mathbf{s}_i$	Times stamps: list last communication time between agent i and all other agents
$\mathbf{tz}_i$	List of arrival times for all tasks that agent i believes to have a current winner
$\mathbf{w}_i^{\text{any}}$	Permission to bid any
$\mathbf{w}_i^{\text{solo}}$	Permission to bid solo
$\mathbf{y}_i$	Winning Bids: list of winning bids according to agent i
$\mathbf{z}_i$	Winning Agents: list of winning agents according to agent i
$\mathcal{A}$	Set of all activities
$\mathcal{A}_j$	Set of all tasks in activity j
$\mathcal{D}_j$	Dependency matrix for activity j
$\mathcal{I}$	Set of all agent indices

$\mathcal{J}$	Set of all task indices
$\mathcal{T}_j$	Temporal constraint matrix for activity j
$\oplus_n$	Insertion operator
$\setminus$	Set difference operator
τ_j^{earliest}	Minimum allowed start time for task j, constraints considered
τ_j^{end}	Maximum allowed start time for task j
τ_j^{latest}	Maximum allowed start time for task j, constraints considered
τ_j^{start}	Minimum allowed start time for task j
$\tau_{ij}^{\text{allowed}}$	Set of times agent i can begin task j and still have time to perform all tasks in their plan
τ_{ij}^{opt}	Visit time that gives greatest score for agent i performing task j
c_{ij}	Marginal score for agent i interested in task j
d_j	Duration of task j
l_b	Current length of bundle, equivalently current length of path
L_t	Maximum bundle length
N_t	Number of tasks
N_u	Number of agents
$N_{\text{req}}(j_q)$	Number of dependency constraints associated with task j_q
o_{j_q}	Timeout parameter for task j_q
S_j	Score function for task j
$S_{path}(\mathbf{p}_i)$	Score for path $\mathbf{p}_i$
$\text{canBid}_i(j_q)$	Boolean, true if agent i is allowed to bid on task j_q

Chapter 1

Introduction

Missions involving multiple autonomous agents require planning and control on many levels. Mission planners of varying complexity have been developed to take in high level mission objectives and measurement data from the agents to decide on actions for each of the agents in the network [1–4]. Typically, mission planners have many layers, each dedicated to an aspect of the decision making process. This thesis investigates two of the planning problems associated with autonomous agents cooperating in complex missions: 1) the task planning problem with coupled constraints, and 2) the autonomous search problem in a partially unknown environment.

The Task Planning Problem with Coupled Constraints To simplify the planning problem, missions are often broken down into tasks, where each task represents some aspect of the mission that needs to be performed [5]. Task planning decisions are made regarding which agent should carry out which task, and when each task should be performed. Decisions are made such that the mission is accomplished with the lowest cost, or equivalently the largest reward.

In simple mission scenarios, each task may be assigned independently, as long as two agents are not assigned to the same task. However, in a complex mission setting, coupling exists between certain tasks in the task set. The importance of a task may depend on whether or not another task is assigned, and the value a task adds to a mission may be a function of when it is performed in relation to other tasks. This

thesis investigates the task planning problem with coupled constraints.

The Autonomous Search Problem in a Partially Unknown Environment Mission objectives often include locating objects of interest in the environment. The object's initial positions are typically unknown, but there may exist an a priori probability distribution that is known. The objects may be mobile, and so the possible trajectories of the objects must be considered. The autonomous agents in a search problem are equipped with some type of sensor with a finite field of view, and some limited range. An object may be observed if it is contained within the sensor field of view. The first time an object is observed, it is said to be *found.* The goal of the autonomous fleet is to find as many objects as possible, and each object should be found as soon as possible.

1.1 Previous Work

Complex missions involving a fleet of autonomous agents pose a difficult planning problem. The architecture of the mission planner used in such a scenario may be either centralized, or decentralized. In a purely centralized planning architecture [6–9], all decisions are made by a single computation source, and each agent in the network receives its plan from this centralized node. In a decentralized architecture, multiple nodes in the planning network contribute to the decision-making process. Each architecture posses both advantages and disadvantages, and many modern systems are not purely one or the other, but use elements of each.

One advantage of a centralized architecture is that all computation can be performed from a ground control station, enabling lighter and less expensive vehicles. Furthermore, techniques have been developed to specify a general set of constraints, allowing very complex mission scenario. However, this type of architecture requires a global information set, which may be difficult or impossible to obtain in real time. As a result, centralized architectures often rely on a high bandwidth communication infrastructure.

Ideally, the communication link between all elements of the system (command station, autonomous vehicles, manned vehicles, etc.) is high bandwidth, low latency, low cost, and highly reliable. However, even the most modern communication infrastructures do not posses all of these characteristics. If the inter-agent communication mechanism has a more favorable combination of these characteristics compared to agent-to-base communication, then a decentralized planning architecture offers performance and robustness advantages. In particular, response times to changes in situational awareness can be significantly faster via decentralized control than those achieved under a purely centralized planner. In addition, decentralized planning schemes are well-suited for situations where the information needed for decision making is local with respect to the network diameter. This is particularly noticeable in the task assignment consensus problem where agents near each other will require the most communication to resolve task assignment conflicts, whereas agents that are spatially separated are less likely to choose the same tasks. Decentralized algorithms with strong inter-agent communication should support this scenario more efficiently than centralized approaches. Therefore, the focus of this thesis is on decentralized methods for planning in complex missions. However, this work also attempts build an architecture that possesses advantages commonly found in centralized methods, most importantly, the ability to handle coupled constraints.

1.1.1 Previous Work in Task Planning

Centralized Techniques for Satisfying General Coupled Constraints The work described in [10] provides a framework to solve the task assignment problem while enforcing timing constraints. The task assignment problem can be formulated as a Mixed Integer Linear Program (MILP) and solved exactly by a commercially available numerical solver. Alternatively, a suboptimal solution can be found using Tabu search. The contributions in [11] provide a method for encoding detailed mission specific constraints into a MILP, using the language of Linear Temporal Logic (LTL). This work enables the user to rapidly, and intuitively encode the correct temporal and dependency constraints.

In [12], three types of timing constraints are considered: simultaneous arrival where agents must begin a task at the same time, tight sequencing where the difference in arrival times between agents is specified exactly, and loose sequencing where the difference in arrival times between agents is constrained to fall within a specified range. The solution technique focuses on the cooperative path planning problem in the context of multiple UAV task planning and scheduling.

These centralized methods are very power because of the constraint specification generality. The characteristics of these methods are desirable for the planning systems for complex missions, but since the techniques are centralized, they may be unsuitable for certain communication infrastructures.

Decentralized Techniques and Market Based Methods One approach to decentralized planning is to instantiate multiple instances of the same planner on each autonomous agent [13]. Each agent plans for the entire fleet, and executes only the plan it generated for itself. If all agents begin with the same information set, or Situational Awareness (SA), then they will generate identical plans, and this is referred to as *implicit coordination.* However, in real-world applications, the SA may not be completely consistent across the fleet. Consensus protocols can be used to share information such that the agent network approaches a consistent SA [14–21]. However, in a dynamic environment, consensus schemes can be slow to converge, introducing latency, or never converge at all. The work in [22] attempts to add robustness to the task assignment process via implicit coordination by communicating plan information as well as parameter information.

Other methods for solving the task assignment problem are *market-based approaches* [23, 24], which are tractable for real-time applications. Many successful market-based approaches use an auction mechanism [25, 26]. Auctions may be run via an auctioneer [27–29], where each agent computes the reward associated with a task based on their own SA, and uses that as their *bid.* Each agent communicates their bid to the auctioneer, and the auctioneer determines the winner for each task. These types of methods guarantee conflict free solutions since the auctioneer only selects

one agent as the winner. They are decentralized in the sense that the computation is distributed, but they do rely on a centralized auctioneer.

Some auction-based protocols do not need to designate a single agent as the auctioneer, but utilize a different protocol where the winner is decided based on a set of self-consistent rules [30]. One such method is the Consensus-Based Bundle Algorithm (CBBA) [31, 32], which is a polynomial-time, market-based approach that uses consensus to reach agreement on the plan as opposed to the mission parameters. Several extensions to CBBA has been made recently including predictive planning [33] where the task duration in uncertain, time windows [34] where each task is time-sensitive, asynchronous communication [35] which enables realistic decentralized implementation, and cooperative assignment [36] which enforces some limited coupling relationships in the task set.

Techniques for Enforcing Priority Levels The task planning problem with *priority* has drawn some attention in the cooperative control community [37–40]. The mission scenario involves groups of tasks which posses a distinct priority. The constraints specify that lower priority tasks must not be assigned unless their corresponding higher priority task is executed first. This type of framework is commonly applied to the cooperative search, track, and engage mission. Each target in the environment must be first tracked to confirm they are hostile, then engaged, and finally observed to provide a battle damage assessment (BDA). These tasks must be executed in this order, and cannot be assigned if the preceding task is not assigned.

Solutions to the single assignment problem with priority constraints are given in [37, 38]. The single assignment problem is characterized by each agent planning at most one task in advanced, whereas in the multiple assignment problem, a list of tasks is planned for each agent. Multiple assignment is a more complex planning problem than single assignment, but offers a significant performance increase [32]. The primary objective in this thesis is to develop a solution strategy for the multiple-assignment task planning problem with coupled constraints.

1.1.2 Previous Work in Search

One standard approach to the UAV search problem is to discretize the world into cells, and utilize a *cognitive map* to guide the search [38, 41–43]. The cognitive map contains information pertaining to the value of observing each cell. Quantities represented in the map can include the probability that a target is present in a cell at a given time step, the current level of uncertainty in each cell, or the time-since-last-visit for each cell. Path planning decisions are made based on the cognitive maps, and often a coordination strategy is employed to increase efficiency of the search. Several authors have demonstrated the benefits of coordinating the search effort to avoid redundant efforts.

Cognitive maps may be implemented according to various architectures. Maps can be centralized [38] and continuously accessed by agents in the network. Conversely, they may be distributed, where each agent carries their own version of the map, which reflects what that agent believes about the world. The advantage of a shared centralized map is that all agents have the same information set. Since situational awareness pertaining to information in the world is consistent across the fleet, implicit coordination is possible: Agents can infer the plan for the fleet as a whole, and execute their respective portion of that plan. The disadvantage is that large amounts of communication are required to update and access the central map. An alternative, is the distributed map approach where each agent carries their own version of the map. Since communication is prone to drop outs and latency, these maps likely differ in a quantitative sense, but contain similar qualitative information. Inconsistent situational awareness is inherent in the distributed map approach, so coordination must be planned explicitly.

Ideally, the planning system generates trajectories that maximize the collective information gained over the course of the mission, and minimize the expected time until each of the targets are found. However, constructing full trajectories that span the entire mission is often intractable for real-world applications. Therefore, the planner only generates trajectories over a small planning horizon, which can be done

in real-time.

The practice of optimizing over a short planning horizon is a tractable method for searching an environment. However, this strategy alone is prone to local minima because of its inherent near-sightedness. For instance, the area around the refuel base may be heavily traveled if all agents refuel from the same base. The uncertainty around the base will then be very low for most of the mission. An agent that has recently refueled, will plan a locally optimal trajectory, but that trajectory may not lead the agent into the most interesting portions of the map. The best place to search (an area yet to be explored, for example) may be beyond the agent's planning horizon, thus ignored. To overcome these issues the second objective of this thesis is to develop a method for effectively extending the planning horizon of an autonomous team searching for targets in an environment.

1.2 Summary of Contributions

This thesis presents a decentralized method for solving the task assignment problem with coupled constraints. The method builds on an existing task assignment algorithm called the Consensus Based Bundle Algorithm (CBBA). The new framework is able to handle both assignment constraints, which specify which combinations of tasks are permissible, and temporal constraints, which specify the permissible relationships between the start times for certain tasks. The performance of the new method is compared to the baseline algorithm in a numerical simulation and the results indicate the the new method achieves a performance increase of up to a factor of three.

This thesis also presents a decentralized method for autonomous searching. The search algorithm presented builds off of an existing algorithm and effectively extends its planning horizon. Mission simulations confirm that the search algorithm presented out performs the original algorithm in maximizing the number of targets found, and minimizing the amount of time required to find each target.

1.3 Thesis Overview

The remainder of this thesis is organized as follows: Chapter 2 provides a description of the baseline task assignment algorithm. It first explains the information structures specific to the method, then describes the task assignment process. Chapter 3 describes an extension to the baseline task assignment algorithm which is capable of handling coupled constraints. Chapter 3.10 presents numerical results for the algorithmic extension described in Chapter 3. Chapter 4 discusses an algorithmic extension for handling refuel constraints in the task assignment process. Numerical results for the extension are also presented in this chapter. Chapter 5 introduces several strategies for solving the autonomous search problem, including a newly developed algorithm. The strategies are compared in several mission scenarios, and the results are presented with commentary in this chapter as well.

Chapter 2

Background on Baseline Task Planning Algorithm

The Consensus-Based Bundle Algorithm (CBBA) is a decentralized, polynomial-time, market-based task planning protocol. CBBA consists of two phases which alternate until the assignment converges. In the first phase: *task selection*, each agent sequentially adds tasks to its plan, attempting to maximize the total score for its plan. In the second phase: *conflict resolution*, plan information is exchanged between neighbors, and tasks go to the highest bidder. CBBA is guaranteed to reach a conflict free assignment.

2.1 Information Structures

During the task assignment process, each agent i stores several information structures that are described below:

Bundle: The bundle, $\mathbf{b}_i \triangleq \{b_{i1}, \ldots, b_{i(l_b)}\}$, whose elements are $b_{in} \in \mathcal{J}$ for $n = 1, \ldots, l_b$, contains the indices of the tasks agent i is currently winning. The length of the bundle, denoted l_b, is restricted to be no longer than some maximum L_t. Therefore, the parameter L_t sets the planning horizon for the system. The bundle is ordered chronologically with respect to when the tasks were chosen: task b_{in} was

chosen before task $b_{i(n+1)}$. If agent i is not currently winning any tasks, $\mathbf{b}_i = \emptyset$, and $l_b = 0$.

Path: The path, $\mathbf{p}_i \triangleq \{p_{i1}, \ldots, p_{i(l_b)}\}$, whose elements are $p_{in} \in \mathcal{J}$ for $n = 1, \ldots, l_b$, also contains the set of tasks agent i is currently winning. However, the path is ordered according to when agent i plans to carry out each task: p_{in} is scheduled to begin before $p_{i(n+1)}$. The path is the same length as the bundle, and thus is also not permitted to be longer than L_t.

Times: The vector of times, $\boldsymbol{\tau}_i \triangleq \{\tau_{i1}, \ldots, \tau_{i(l_b)}\}$, whose elements are $\tau_{in} \in \mathbb{R}_+$ for $n = 1, \ldots, l_b$, contains the set of planned arrival times corresponding to each of the tasks in the path, $\mathbf{p}_i$. Since the path is sorted according to arrival times, the elements in the time vector monotonically increase. The vector of times is the same length as the path by definition.

Winning Agents: The winning agents list, $\mathbf{z}_i \triangleq \{z_{i1}, \ldots, z_{iN_t}\}$, whose elements are $z_{in} \in \mathcal{I} \cup \{\emptyset\}$ for $n = 1, \ldots, N_t$, stores information pertaining to which agent is currently winning each task. Specifically, the value in element z_{in} is the index of the agent who is currently winning task n according to agent i. If agent i believes that there is no winner for task n, $z_{in} = \emptyset$.

Winning Bids: The winning bids list, $\mathbf{y}_i \triangleq \{y_{i1}, \ldots, y_{iN_t}\}$, whose elements are $y_{in} \in \mathbb{R}_+$ for $n = 1, \ldots, N_t$, contains the highest current bid for each task in the task set. If agent i believes that there is no winner for task n, $y_{in} = 0$.

Time Stamps: The time stamp list, $\mathbf{s}_i \triangleq \{s_{i1}, \ldots, s_{iN_u}\}$, whose elements are $s_{in} \in \mathbb{R}_+$ for $n = 1, \ldots, N_u$, indicates the amount of time elapsed since agent i received an information update from each of the other agents in the network. The time stamp list is used to keep track of the age of a given piece of information for tie-breaking purposes. New information is trusted over old information.

2.2 Phase I: Task Selection

In the first phase of CBBA, *task selection*, each agent iteratively adds tasks to its bundle. The selection process is *sequentially greedy*; given an initial bundle, $\mathbf{b}_i$, agent i adds the task that will result in the largest marginal increase in score, and repeats the process until the bundle is full, or no more tasks can be added. Each time agent i adds a task j to its bundle, it enters its own index into the winning agent list, $z_{ij} \leftarrow i$, and its corresponding bid into the winning bids list, $y_{ij} \leftarrow S_j(i, t_{ij})$. Agent i is only allowed to add task j to its bundle if agent i can place a higher bid than the current highest bidder, y_{ij}.

Agent i computes the marginal scores for each task as

$$c_{ij}(\mathbf{p}_i) = \begin{cases} 0 & \text{if } j \in \mathbf{p}_i \\ \max_{n \leq l_b} S_{path}(\mathbf{p}_i \oplus_n j) - S_{path}(\mathbf{p}_i) & \text{otherwise} \end{cases} \tag{2.1}$$

where $c_{ij}(\mathbf{p}_i)$ is the marginal score for agent i performing task j, $S_{path}(\mathbf{p}_i)$ is the score for agent i performing the tasks in the path $\mathbf{p}_i$, and $\oplus_n$ denotes the operation that inserts the second list right after the n^{th} element of the first list.

The task selection process for agent i is given by Algorithm 1, where its inputs are the initial bundle $\mathbf{b}_i$, path $\mathbf{p}_i$, and the most recent list of winning bids available $\mathbf{y}_i$.

Notice that the calculation of the arrival times are not explicitly described in Algorithm 1. Instead, the arrival times are uniquely defined by the path, $\mathbf{p}_i$.

$$\boldsymbol{\tau}_i = \arg\max_{\boldsymbol{\tau}_i \in \mathbb{R}_+^{l_b}} \sum_{n=1}^{l_b} S_{p_{in}}(i, \tau_{in}) \tag{2.2}$$

$$\text{subject to:} \quad \tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_{p_{in}}) < \tau_{in} \quad \text{for } n = 1$$

$$\tau_{i(n-1)} + d_{p_{i(n-1)}} + \Delta(\text{tloc}_{p_{i(n-1)}}, \text{tloc}_{p_{in}}) < \tau_{in} \quad \text{for } n = \{2, \ldots, l_b\}$$

where $S_{p_{in}}(\cdot)$ is the score function for the task located in the n^{th} position of the path of agent i, $d_{p_{in}}$ is the duration of the task located at the n^{th} place in agent i's path, $\Delta(A, B)$ is the time required to travel from the location A to the location B, aloc_i is

Algorithm 1 Task Selection Process for Agent i, Original CBBA

1. Calculate marginal scores for all tasks

$$c_{ij}(\mathbf{p}_i) = \begin{cases} 0 & \text{if } j \in \mathbf{p}_i \\ \max_{n \leq l_b} S_{path}(\mathbf{p}_i \oplus_n j) - S_{path}(\mathbf{p}_i) & \text{otherwise} \end{cases}$$

2. Determine which tasks are winnable

$$h_{ij} = \mathbb{I}(c_{ij} > y_{ij}) \quad \forall j \in \mathcal{J}$$

3. Select the index of the best eligible task, $j^\star$, and select best location in the plan to insert the task, $n_j^\star$

$$j^\star = \arg\max_{j \in \mathcal{J}} c_{ij} \cdot h_{ij}$$

$$n_j^\star = \arg\max_{n \in \{0,\ldots,l_b\}} S_{path}(\mathbf{p}_i \oplus_n j^\star)$$

4. If $c_{ij^\star} \leq 0$, then **return**. otherwise, continue to 5.

5. Update agent information

$$\mathbf{b}_i = \mathbf{b}_i \oplus_{l_b} j^\star$$

$$\mathbf{p}_i = \mathbf{p}_i \oplus_{n_j^\star} j^\star$$

6. Update shared information vectors

$$y_{i(j^\star)} = c_{ij^\star}$$

$$z_{i(j^\star)} = i$$

7. **if** $l_b = L_t$, **then return**, otherwise, go to 1.

---

the current location of agent i, tloc_j is the location of task j, and τ_{cur} is the current time. The constraints ensure that an agent has sufficient time to complete each task and travel from each task location to the next, provided that the durations are known, and the travel times are deterministic.

The score for a given path, $S_{path}(\mathbf{p}_i)$ can be written similarly:

$$S_{path}(\mathbf{p}_i) = \max_{\boldsymbol{\tau}_i \in \mathbb{R}_+^{l_b}} \sum_{n=1}^{l_b} S_{p_{in}}(i, \tau_{in}) \tag{2.3}$$

$$\text{subject to:} \qquad \tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_{p_{in}}) < \tau_{in} \qquad \text{for } n = 1$$

$$\tau_{i(n-1)} + d_{p_{i(n-1)}} + \Delta(\text{tloc}_{p_{i(n-1)}}, \text{tloc}_{p_{in}}) < \tau_{in} \quad \text{for } n = \{2, \ldots, l_b\}$$

The solution to (2.2) and (2.3) may be difficult to find if the score function for each task is a general function of time, since it is an optimization over an entire set of parameters. However, if the score function S_j is monotonically decreasing with respect to time for all tasks, the optimal solution can be found easily in polynomial time with respect to the length of the path. The procedure is to first select the time for the task that appears first in the path as

$$\tau_{i1} = \tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_{p_{i1}}) \tag{2.4}$$

Then, for each subsequent task, choose the time to be:

$$\tau_{in} = \tau_{i(n-1)} + d_{p_{i(n-1)}} + \Delta(\text{tloc}_{p_{i(n-1)}}, \text{tloc}_{p_{in}}) \tag{2.5}$$

The rational, is that if the score function for each task is monotonically decreasing, and the order in which tasks are to be executed is given, then the tasks should each be executed as early as possible.

2.3 Phase II: Consensus

After all agents complete a round of the task selection phase, agents communicate with each other to resolve conflicting assignments within the team. After receiving information from neighboring agents about the winning agents and corresponding winning bids, each agent can determine if it has been outbid for any task in its bundle. Since the bundle building recursion (Section 2.2) depends at each iteration upon the tasks in the bundle up to that point, if an agent is outbid for a task, it must release it and all subsequent tasks from its bundle. If the subsequent tasks are not released, then the current best scores computed for those tasks would be overly conservative, possibly leading to a degradation in performance. It is better therefore to release all tasks after the outbid task and redo the task selection process to add

these tasks (or possibly better ones) back into the bundle.

This consensus phase assumes that each pair of neighboring agents synchronously share the following information vectors: the winning agent list $\mathbf{z}_i$, the winning bids list $\mathbf{y}_i$, and the list of time stamps $\mathbf{s}_i$ representing the time stamps of the last information updates received from all the other agents. The time stamp list for any agent i is updated using the following equation,

$$s_{ik} = \begin{cases} \tau_r, & \text{if } g_{ik} = 1 \\ \max\{s_{mk} \mid m \in \mathcal{I}, g_{im} = 1\} & \text{otherwise,} \end{cases} \tag{2.6}$$

which states that the time stamp s_{ik} that agent i has about agent k is equal to the message reception time τ_r if there is a direct link between agents i and k (i.e. $g_{ik} = 1$ in the network graph), and is otherwise determined by taking the latest time stamp about agent k from the set of agent i's neighboring agents.

For each message that is passed between a sender k and a receiver i, a set of actions is executed by agent i to update its information vectors using the received information. These actions involve comparing its vectors $\mathbf{z}_i$, $\mathbf{y}_i$, and $\mathbf{s}_i$ to those of agent k to determine which agent's information is the most up-to-date for each task. There are three possible actions that agent i can take for each task j:

1. **Update**: $z_{ij} = z_{kj}$, $y_{ij} = y_{kj}$
2. **Reset**: $z_{ij} = \emptyset$, $y_{ij} = 0$
3. **Leave**: $z_{ij} = z_{ij}$, $y_{ij} = y_{ij}$.

The decision rules for this synchronous communication protocol were presented in [44] and are provided for convenience in Table A.1 in Appendix A. The first two columns of the table indicate the agent that each of the sender k and receiver i believes to be the current winner for a given task; the third column indicates the action that the receiver should take, where the default action is "Leave".

If either of the winning agents or winning bids lists ($\mathbf{z}_i$ or $\mathbf{y}_i$) are changed as an outcome of the communication, the agent must check if any of the updated or reset

tasks were in its bundle. If so, those tasks, along with all others added to the bundle after them, are released. Thus if $\bar{n}$ is the location of the first outbid task in the bundle ($\bar{n} = \min\{n \mid z_{i(b_{in})} \neq i\}$ with b_{in} denoting the n^{th} entry of the bundle), then for all bundle locations $n \geq \bar{n}$, with corresponding task indices b_{in}, the following updates are made:

$$z_{i(b_{in})} = \emptyset \quad \text{and} \quad y_{i(b_{in})} = 0 \tag{2.7}$$

The bundle is then truncated to remove these tasks,

$$\mathbf{b}_i \leftarrow \{b_{i1}, \ldots, b_{i(\bar{n}-1)}\} \tag{2.8}$$

and the corresponding entries are removed from the path and times vectors as well. From here, the algorithm returns to the first phase where new tasks can be added to the bundle. CBBA iterates between these two phases until the information lists, $\mathbf{z}_i$ and $\mathbf{y}_i$ stop changing for all agents in the network.

2.4 Convergence and Performance Properties

2.4.1 Convergence

It has been previously shown that if the scoring function satisfies a certain condition, called *diminishing marginal gain* (DMG), CBBA is guaranteed to produce a conflict-free assignment and converge in at most $\max\{N_t, L_t N_u\}D$ iterations, where D is the network diameter (always less than N_u) [44]. The DMG property states that the score for a task cannot increase as other elements are added to the set before it. In other words,

$$c_{ij}(\mathbf{p}_i) \geq c_{ij}(\mathbf{p}_i \oplus_n m) \tag{2.9}$$

for all $\mathbf{p}_i$, n, m, and j, where $m \neq j$ and $m, j \notin \mathbf{p}_i$. The convergence proof is given in [44] if more details are desired by the reader. The proof assumes that the network is strongly connected (Each agent can communicate with all other agents

through some communication path), but not necessarily fully connected (Each agent can communicate directly with all other agents).

2.4.2 Performance

CBBA is a suboptimal task assignment algorithm, but it posses a performance guarantee. Given a strongly connected network of agents, each with accurate situational awareness, CBBA converges to an assignment with a score that is at least 50 percent of the optimal assignment score [44]. The performance guarantee is empirically shown to be very conservative; numerical results demonstrate that CBBA achieves assignments that are typically 95 percent of optimal or better for randomly generated test cases [44].

2.5 CBBA with Time Windows

The CBBA with Time Windows algorithm extension [34] is designed to provide task assignments when each task is constrained to begin inside of a specific time window of validity. To begin, some terminology is defined:

Score function: The score function from the original CBBA can be written $S_j(a_j, t_j)$ and it gives the score an agent receives from task j when it arrives at the task at time t_j. This score is typically composed of two parts: 1) the nominal reward for the task, $R_j(a_j)$, which is a function of a_j, the index of the agent assigned to task j, and 2) the discount function, which is a function of the arrival time for task j, t_j. For example, for time discounted cases this quantity can be defined as $S_j(a_j, t_j) = e^{-\lambda \cdot t_j} R_j(a_j)$, where λ is a discount parameter that determines how quickly the score decays with time.

Time Window: In a more general formulation, each task j is given a time window of validity defined as $[\tau_j^{\text{start}}, \tau_j^{\text{end}}]$. A task begun outside this window gains no reward,

so the score function may be augmented to include an additional term

$$S_j(a_j, t_j) = e^{-\lambda \cdot (t_j - \tau_j^{\text{start}})} \cdot R_j(a_j) \cdot \mathbb{I}(t_j \in [\tau_j^{\text{start}}, \tau_j^{\text{end}}])$$

where $\mathbb{I}(\cdot)$ is the indicator function which returns 1 if the argument is true, and 0 otherwise. Alternatively, the time window can be treated as an additional constraint:

$$\begin{aligned}
\boldsymbol{\tau}_i = \arg\max_{\boldsymbol{\tau}_i \in \mathbb{R}_+^{l_b}} & \textstyle\sum_{n=1}^{l_b} S_{p_{in}}(i, \tau_{in}) && (2.10)\\
\text{subject to:} \quad \tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_{p_{in}}) &< \tau_{in} && \text{for } n = 1\\
\tau_{i(n-1)} + d_{p_{i(n-1)}} + \Delta(\text{tloc}_{p_{i(n-1)}}, \text{tloc}_{p_{in}}) &< \tau_{in} && \text{for } n = \{2, \ldots, l_b\}\\
\tau_{in} &\geq \tau_{p_{in}}^{\text{start}} && \text{for } n = \{1, \ldots, l_b\}\\
\tau_{in} &\leq \tau_{p_{in}}^{\text{end}} && \text{for } n = \{1, \ldots, l_b\}
\end{aligned}$$

During the task selection phase of CBBA, a constraint is added such that adding a task to agent i's path is not allowed to alter the starting times of each of the tasks already in the path. With the this restriction, $c_{ij}(\mathbf{p}_i)$ can be written

$$c_{ij}(\mathbf{p}_i) = \max_{t \in \tau_{ij}^{\text{allowed}}(\mathbf{p}_i)} S_j(i, t) \tag{2.11}$$

and the start time,

$$\tau_{ij}(\mathbf{p}_i) = \arg\max_{t \in \tau_{ij}^{\text{allowed}}(\mathbf{p}_i)} S_j(i, t) \tag{2.12}$$

where $\tau_{ij}^{\text{allowed}}(\mathbf{p}_i)$ is the set of allowable start times for task j, conditioned on the current path, $\mathbf{p}_i$, and is calculated

$$\tau_{ij}^{\text{allowed}}(\mathbf{p}_i) = [\tau_j^{\text{start}}, \tau_j^{\text{end}}] \setminus \Omega_{i1}(\mathbf{p}_i, j) \setminus \Omega_{i2}(\mathbf{p}_i, j) \cdots \setminus \Omega_{il_b}(\mathbf{p}_i, j) \tag{2.13}$$

where $\Omega_{in}(\mathbf{p}_i, j)$ is the interval of time that is invalid as a start time for task j because of the n^{th} task in path $\mathbf{p}_i$, and $\setminus$ is the set difference operator. Specifically, task j is not allowed to begin inside $\Omega_{in}(\mathbf{p}_i, j)$, because it would be temporally too close to τ_{in}, meaning that agent i would be either late to task p_{in}, or late to task j.

$\Omega_{in}(\mathbf{p}_i, j)$ is given by

$$\Omega_{in}(\mathbf{p}_i, j) = [\tau_{in} - d_j - \Delta(\text{tloc}_j, \text{tloc}_{p_{in}}), \tau_{in} + d_{p_{in}} + \Delta(\text{tloc}_{p_{in}}, \text{tloc}_j)] \tag{2.14}$$

To verify that the time-windows framework satisfies the DMG property it must be shown that for all $j \notin \mathbf{p}_i$, $m \notin \mathbf{p}_i$, and $n_m \in \{0, \ldots, l_b\}$

$$c_{ij}(\mathbf{p}_i) \geq c_{ij}(\mathbf{p}_i \oplus_{n_m} m) \tag{2.15}$$

Notice that

$$\tau_{ij}^{\text{allowed}}(\mathbf{p}_i \oplus_{n_m} m) = \left(\tau_{ij}^{\text{allowed}}(\mathbf{p}_i)\right) \setminus \left(\Omega_{i(n_m)}(\mathbf{p}_i \oplus_{n_m} m, j)\right) \tag{2.16}$$

and so $\tau_{ij}^{\text{allowed}}(\mathbf{p}_i \oplus_{n_m} m)$ is necessarily a subset of $\tau_{ij}^{\text{allowed}}(\mathbf{p}_i)$ for all $n_m = \{1, \ldots, l_b\}$. Therefore, $c_{ij}(\mathbf{p}_i \oplus_{n_m} m)$ must be no greater than $c_{ij}(\mathbf{p}_i)$, hence DMG is preserved. Therefore, given a strongly connected network of agents, and a score function that satisfies DMG, CBBA with Time Windows will converge to a conflict-free solution in $\max\{N_t, L_t N_u\}D$ or fewer iterations, where D is the network diameter.

Solving (2.11) requires continuous optimization over a non-convex space, which can be computationally expensive for large problems. However, the problem can be decomposed into a set of smaller problems, where the task j is inserted into the path $\mathbf{p}_i$ *after* each position, n_j, and the marginal score is taken to be the maximum over the calculated values.

$$t_{ij}(n_j|\mathbf{p}_i) = \arg\max_{t_{ij} \in \mathbb{R}_+} S_j(i, t_{ij}) \tag{2.17}$$

$$\begin{aligned}
\text{subject to:} \quad & \tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_j) < t_{ij} && \text{if } n_j = 0 \\
& \tau_{i(n_j)} + d_{p_{i(n_j)}} + \Delta(\text{tloc}_{p_{i(n_j)}}, \text{tloc}_j) < t_{ij} && \text{if } n_j > 0 \\
& t_{ij} + d_j + \Delta(\text{tloc}_j, \text{tloc}_{p_{i(n_j+1)}}) < \tau_{i(n_j+1)} && \text{if } n_j < l_b \\
& \tau_{i(n_j)} < t_{ij} && \text{if } n_j > 0 \\
& t_{ij} < \tau_{i(n_j+1)} && \text{if } n_j < l_b
\end{aligned}$$

$$t_{ij} \geq \tau_j^{\text{start}}$$
$$t_{ij} \leq \tau_j^{\text{end}}$$

If a constraint feasible solution to (2.17) cannot be found, then the n_j^{th} location is not a valid place to insert task j into $\mathbf{p}_i$. If no valid location can be found, the marginal score for adding task j to $\mathbf{p}_i$ is zero.

An efficient algorithm is provided next for calculating the marginal score, c_{ij}, the visit time, $\tau_{i(n_j)}$ and the optimal index $n_j^\star$ for a task j given agent i's path $\mathbf{p}_i$. The algorithm assumes that the score function has some unique time τ_{ij}^{opt} that results in the maximum reward for visiting that task. Furthermore, the score is strictly monotonically increasing for time less than τ_{ij}^{opt}, and strictly monotonically decreasing for time greater than τ_{ij}^{opt}.

If the score structure for each task has these properties, then Algorithm 2 and Algorithm 3 are used for the task selection phase, and the consensus process is unchanged from the original CBBA. The algorithm as it is presented here is the baseline algorithm from which the coupled-constraint extension is described.

Notice that Algorithm 2 and Algorithm 3 feature a feasibility vector, F_{ij}. This vector comprises Booleans for each location j in path $\mathbf{p}_i$. Each Boolean $F_{ij}(n)$ indicates whether or not it is feasible to insert task j into the n^{th} location of path $\mathbf{p}_i$. The vector is initialized at the beginning of each task selection phase, and effectively prunes the decision space as the task selection process continues. Once the algorithm determines that an insertion location for task j is infeasible, that location does not need to be checked again in this phase, so it is pruned from the list by setting its Boolean to false.

2.6 Summary

In this chapter explains the baseline task assignment algorithm that is used in the remainder of this thesis. CBBA comprises two phases. In this first phase, task selection, agents build a *bundle* by sequentially selecting tasks which result in the greatest

Algorithm 2 Task Selection Process for Agent i, CBBA w/ TW

1. Initialize Feasibility Vector

$$F_{ij}(k) = \text{true } \forall\ k = \{0, \ldots, l_b\}$$

2. Determine which marginal scores are to be calculated, set all others to zero

$$\mathcal{J}_i^{\text{calc}} = \mathcal{J} \setminus \mathbf{p}_i$$

$$c_{ij} = 0 \quad \forall j \in \mathcal{J}, j \notin \mathcal{J}_i^{\text{calc}}$$

3. Calculate marginal scores for all tasks, $j \in \mathcal{J}_i^{\text{calc}}$ using Algorithm 3. Inputs: $\mathbf{p}_i$, S_j, and $[\tau_j^{\text{start}}, \tau_j^{\text{end}}]$ Outputs: c_{ij}, n_{ij}, t_{ij} for each task j

4. Determine which tasks are winnable

$$h_{ij} = \mathbb{I}(c_{ij} > y_{ij}) \quad \forall j \in \mathcal{J}$$

5. Select the index of the best eligible task, $j^\star$

$$j^\star = \arg\max_{j \in \mathcal{J}} c_{ij} \cdot h_{ij}$$

6. If $c_{ij^\star} \leq 0$, then **return**. otherwise, continue.

7. Update agent information

$$\mathbf{b}_i = \mathbf{b}_i \oplus_{l_b} j^\star$$

$$\mathbf{p}_i = \mathbf{p}_i \oplus_{n_{ij^\star}} j^\star$$

$$\boldsymbol{\tau}_i = \boldsymbol{\tau}_i \oplus_{n_{ij^\star}} t_{ij^\star}$$

8. Update shared information vectors

$$y_{i(j^\star)} = c_{ij^\star}$$

$$z_{i(j^\star)} = i$$

9. Update the feasibility vector

$$F_{ij} = F_{ij} \oplus_{n_j} \text{true}$$

10. **if** $l_b = L_t$, **then return**, otherwise, go to 2.

Algorithm 3 Calculate marginal score of task j given $\mathbf{p}_i$ for agent i

1. **For All** $n \in \{0, \ldots, l_b\}$ **where** $F_{ij}(n) = \text{true}$

   (a) Find τ^{earliest} and τ^{latest} such that agent i has sufficient time to carry out the task before j and the task after j.

$$\tau^{\text{earliest}} = \begin{cases} \max\left((\tau_j^{\text{start}}), (\tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_j))\right) & \text{if } n = 0 \\ \max\left((\tau_j^{\text{start}}), (\tau_{in} + d_{p_{in}} + \Delta(\text{tloc}_{p_{in}}, \text{tloc}_j))\right) & \text{otherwise} \end{cases}$$

$$\tau^{\text{latest}} = \begin{cases} \tau_j^{\text{end}} & \text{if } n = l_b \\ \min\left((\tau_j^{\text{end}}), (\tau_{i(n+1)} - d_j - \Delta(\text{tloc}_j, \text{tloc}_{p_{i(n+1)}}))\right) & \text{otherwise} \end{cases}$$

   (b) Initialize score and time for n location to be zero, which are rewritten if $[\tau^{\text{earliest}}, \tau^{\text{latest}}]$ is feasible.

$$s(n) = 0 \qquad t(n) = \emptyset$$

   (c) **if** $(\tau^{\text{earliest}} > \tau^{\text{latest}})$ **then**

$$F_{ij}(n) = \text{false}$$

   else

   i. **if** $(\tau^{\text{earliest}} \leq \tau_{ij}^{\text{opt}} \wedge \tau^{\text{latest}} \geq \tau_{ij}^{\text{opt}})$ **then**

$$s(n) = S_j(i, \tau_{ij}^{\text{opt}}) \qquad t(n) = \tau_{ij}^{\text{opt}}$$

   ii. **if** $(\tau^{\text{earliest}} \leq \tau_{ij}^{\text{opt}} \wedge \tau^{\text{latest}} \leq \tau_{ij}^{\text{opt}})$ **then**

$$s(n) = S_j(i, \tau^{\text{latest}}) \qquad t(n) = \tau^{\text{latest}}$$

   iii. **if** $(\tau^{\text{earliest}} \geq \tau_{ij}^{\text{opt}} \wedge \tau^{\text{latest}} \geq \tau_{ij}^{\text{opt}})$ **then**

$$s(n) = S_j(i, \tau^{\text{earliest}}) \qquad t(n) = \tau^{\text{earliest}}$$

2. Record marginal score for task j, the location to insert it if it is chosen, and the time to perform it if it is chosen.

$$c_{ij} = \max_{n \in \{0,\ldots,l_b\}} s(n) \qquad n_{ij} = \arg\max_{n \in \{0,\ldots,l_b\}} s(n) \qquad t_{ij} = t(n_{ij})$$

score increase for their bundle. In the second phase, consensus, agents communicate with neighboring agents to resolve conflicts in the assignment. CBBA is guaranteed to converge to a conflict-free assignment that is at least 50 percent of optimal under mild assumptions. CBBA with Time Windows is also introduced in this chapter, which enables the algorithm to produce assignments for time-sensitive tasks. CBBA with Time Windows is shown to satisfy sufficient conditions to guarantee convergence.

Chapter 3

CBBA with Coupled Constraints

This chapter presents an algorithmic extension to CBBA which enables handling of coupled constraints in the task set. CBBA with Time Windows, as presented in Section 2.5, is the baseline algorithm from which this algorithm is developed. The new framework is named the Coupled-Constraint Consensus Based Bundle Algorithm (CCBBA).

3.1 Task Allocation with Constraints

Consider a bounded environment containing a network of, N_u autonomous agents, and a set of N_t tasks. For notational convenience, both agents and tasks are given integer indices. The set of all agent indices is denoted $\mathcal{I} \triangleq \{1, \ldots, N_u\}$, while the set of all task indices is denoted $\mathcal{J} \triangleq \{1, \ldots, N_t\}$.

Each task, $j \in \mathcal{J}$, is given a score function, $S_j(a_j, t_j) \in \mathbb{R}_+$ which represents the value that task adds to the mission as a function of $a_j \in \{\mathcal{I} \cup \emptyset\}$, the index of the agent assigned to task j, and $t_j \in \{\mathbb{R}_{\geq 0} \cup \emptyset\}$ the time the agent plans to arrive at the j^{th} task location. The symbol $\emptyset$ denotes a null entry in the corresponding set. The goal of the task planner is to assign agents to tasks in such a way that the cumulative score over all tasks in the task set is maximized. The task assignment problem at a

given mission time has an objective function expressed as

$$\max \sum_{j=1}^{N_t} S_j(a_j, t_j) \tag{3.1}$$

Note that in this formulation, there exists no requirement to assign an agent to every task, and in fact doing so may be infeasible. However, a task only earns a positive score if an agent is assigned to it, $S_j(a_j, t_j) > 0$ only if $a_j \neq \emptyset$. Therefore, the task planner must determine an appropriate subset of tasks to assign, and select the agent, and visit time for each assigned task. The assignment is thus completely determined by selecting a_j and t_j for each task $j \in \mathcal{J}$.

The problem is subject to two types of constraints: 1) non-coupled constraints, and 2) coupled constraints. The defining characteristic of a non-coupled constraint is that it affects the options available to a given agent i independently of decisions made with respect to all other agents. Non-coupled constraints include:

1. *Capability constraints*: Each task has a requirement, and each agent has a set of capabilities. Compatibility matrix, $\mathcal{M}$ captures each agent's fitness for performing the task. If entry $\mathcal{M}(i,j) > 0$ agent i is allowed to perform task j. If entry $\mathcal{M}(i,j) = 0$ agent i is not allowed to perform task j.

2. *Time window of validity*: Each task j, has a time window of validity given by $[\tau_j^{\text{start}}, \tau_j^{\text{end}}]$. The task must be started on the interval of time specified by the time window. If a task is not time sensitive, it is given a time window of $[0, \infty]$.

3. *Vehicle dynamics*: Each agent has a set of dynamics that impose a set of constraints on that agent. Agent i has a maximum speed, $v_i^{\max}$, a maximum acceleration, $\dot{v}_i^{\max}$, and a minimum turn radius $r_i^{\min}$. These parameters determine the minimum amount of time required between each task the agent performs.

4. *Fuel constraints*: At a given time t, agent i has remaining fuel mass, $m_{\text{fuel}}^i(t)$, and has nominal fuel consumption rate, $\dot{m}_{\text{fuel}}^i$. These parameters determine the maximum remaining time aloft for agent i.

Coupled constraints include any situation where the decisions regarding one agent alter the options available to another agent. Often, in a complex mission, there are a variety of such constraints due to rules of engagement or simply the nature of the tasks involved. Coupled constraints include:

- *Assignment constraints*: Assignment constraints deal with the relationship between tasks with respect to which combination of assignments are permitted. Examples:

    1. *Conflict-freeness:* Each task may have at most one agent assigned to it. (Note that this is a simplifying assumption: mission activities requiring or benefiting from multiple agents may be represented by multiple instances of the same task.)
    2. *Unilateral Dependency:* A task, A is dependent on another task B, but task B is not dependent on A. In other words, task A cannot be assigned unless B is also assigned, but B may be assigned independently of task A.
    3. *Mutual Dependency:* A task A is dependent on another task B, and task B is dependent on task A. Task A and task B must be assigned together or not at all.
    4. *Mutually Exclusive:* A task A cannot be assigned if another task B is assigned, and task B cannot be assigned if task A is assigned.

- *Temporal constraints*- Temporal constraints include any specified relationship between the chosen visit times within a subset of tasks. Common temporal constraints include, but are not limited to:

    1. *Simultaneous:* task A and B must begin at the same time.
    2. *Before:* task A must end before task B begins.
    3. *After:* task A must begin after task B ends.
    4. *During:* task A must begin while task B is in progress.

5. *Not during:* task A must either end before task B begins, or begin after task B ends.

6. *Between:* task A must begin after task B ends and end before task C begins.

The baseline CBBA is capable of handling non-coupled constraints including capability constraints, time windows, and basic vehicle dynamics such as top speed. Note that minimum turning radius is not explicitly accounted for in this framework. However, if the minimum distance between tasks is much larger then the greatest minimum turning radius of the agents, then Euclidean distance is a good approximation for path length between tasks. That assumption is made here.

The baseline algorithm is also capable of guaranteeing conflict-free assignments, assuming a strongly connected network, which is one type of coupled constraint. However, it is not able to account for the other types of coupling which are common in complex missions. This chapter describes the necessary machinery for enforcing the following types of coupled constraints within the CBBA framework: 1) unilateral dependency constraints, 2) mutual dependency constraints, 3) mutual exclusions, and 4) temporal constraints.

The procedure for enforcing these coupled constraints is split between the task selection phase of CBBA, and the conflict resolution phase. The additional protocols for handling coupled constraints which are specific to the task selection phase are described in Sections 3.3, 3.4, 3.5, and 3.6. The additional protocols which are specific to the conflict resolution phase are described in Section 3.7.

3.2 Task Set Partitioning

For book-keeping purposes, the task set is partitioned into sub-groups of tasks that share coupled constraints. Each of these sub-groups is called an *activity.* For notational convenience, each task in an activity is referred to as an *element* of that activity, and is given an index, $q \in \mathbb{Z}_+$. Each task in the task set can be uniquely

identified by its activity number and element index, and the notation j_q is used to indicate the task associated with the q^{th} element of activity j. The set of all tasks in the task set is still denoted $\mathcal{J}$. The set of all activities in the set is denoted $\mathcal{A}$, while the set of all elements in an activity j is denoted $\mathcal{A}_j$. Each pair of elements in an activity may a coupled constraint between them but does not have to. However, by construction, coupled constraints do not exist between tasks belonging to different activities with the exception of the conflict-freeness constraint.

An example is shown in Figure 3-1. In this example, task B may not be assigned unless task A is assigned, but task A may be assigned independently of task B. Task C may not be assigned unless task D is assigned, and task D may not be assigned unless task C is assigned. The task pair (C, E) may not both be assigned, and the task pair (D, E) may not both be assigned. This particular task set can be grouped into two activities, with two and three elements respectively. The task set $\mathcal{J} = \{A, B, C, D, E\}$ is now written $\mathcal{J} = \{1_1, 1_2, 2_1, 2_2, 2_3\}$.

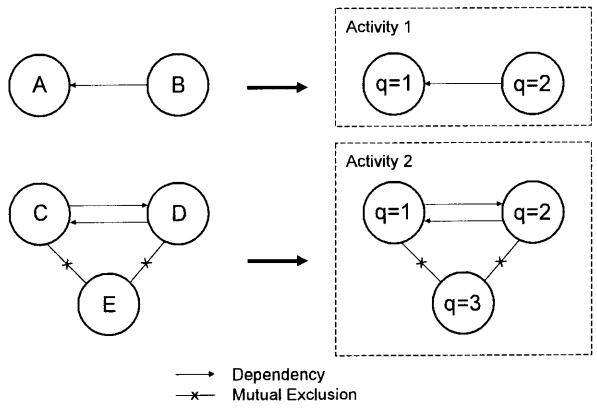

Figure 3-1: Example of task set partitioning

For each activity $j \in \mathcal{A}$, the constraint structure can be compactly written in the form of a *Dependency Matrix*, $\mathcal{D}_j$ where the row column entry, (q, u), describes the relationship between the q^{th} element of activity j and the u^{th} element of activity j. The notation for encoding the constraints is provided in Table 3.1. Note: the diagonal entries of $\mathcal{D}_j$ are defined to be 0 for all activities.

In the example shown in Figure 3-1, activity 1 has two elements: task 1_1 which has

Table 3.1: Code for Dependency Matrix Entry $\mathcal{D}_j(q,u)$

1	element u depends on element q
0	element u may be assigned independently of element q
-1	element q and u are mutually exclusive
$a \in \{2,3,\dots\}$	element u requires either element q or another element with the same code, a. Entries are used sequentially: 3 is not used unless 2 is used, etc.

no dependency constraints, and task 1_2 which depends on task 1_1. The relationship is referred to as a unilateral dependency constraint, and it is represented in the dependency matrix as $\mathcal{D}_1(1,2) = 1$, and $\mathcal{D}_1(2,1) = 0$. Activity 2 has three elements: task 2_1 and task 2_2 are mutually dependent on one another, while task 2_3 is mutually exclusive with both tasks 2_1 and 2_2. The dependency matrices from the example in Figure 3-1 are given by:

$$\mathcal{D}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathcal{D}_2 = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

Each task in the task set is given a *bidding strategy* which represents how the bidding protocol is handled. Consider a task j_q which is the q^{th} element of activity j. The positive entries of the q^{th} column in $\mathcal{D}_j$ indicate which elements of activity j task j_q depends on. If all tasks which task j_q depends on do not also depend on task j_q, then task j_q is given a *pessimistic bidding strategy*, meaning that an agent is not permitted to bid on task j_q unless all dependencies of task j_q are filled. Details of the pessimistic bidding strategy are described in Section 3.3.

Returning to the example, $\mathcal{D}_1$ reveals that task 1_2 depends only on task 1_1, and task 1_1 does not depend on 1_2 because $\mathcal{D}_1(2,1) = 0$. Therefore, task 1_2 would be given a pessimistic bidding strategy, and an agent wishing to add task 1_2 to its bundle would not be permitted to do so unless it had confirmation of another agent currently winning task 1_1.

Now consider an agent that is able to benefit by adding task 2_1 to its bundle. Task 2_1 is mutually dependent on task 2_2, but if all agents wait to bid on these tasks until the other task has a winner, then neither task will ever be assigned. The paradox is that neither agent has incentive to bid without confirmation from a "partnering" agent.

Therefore, an *optimistic bidding strategy* is developed for tasks which are mutually dependent on another task. The optimistic bidding strategy does not require agents to wait until all dependencies of a task are filled before bidding on it. Details of the protocol are described in Section 3.4.

The bidding strategy for each task $j_q \in \mathcal{J}$ is determined from the dependency matrix for the activity to which the task belongs according to

$$\text{bidStrat}_{j_q} = \begin{cases} \text{optimistic} & \text{if } \exists\ u \text{ s.t. } \mathcal{D}_j(u,q) \geq 1 \wedge \mathcal{D}_j(q,u) = 1 \\ \text{pessimistic} & \text{otherwise} \end{cases} \tag{3.2}$$

3.3 Pessimistic Bidding Strategy

Task j_q is given a pessimistic bidding strategy if it is only dependent upon tasks which do not depend on j_q. CBBA is modified in the task selection phase such that an agent is only allowed to bid on the q^{th} element of task j if all dependencies for j_q have a current winner.

In the baseline CBBA, at the beginning of the task selection phase, an agent i calculates the marginal score for each of the tasks in the task set that are not in their bundle. In Coupled-Constraint CBBA, a step is added before this where an agent determines which tasks it is allowed to bid on. This is accomplished by comparing the number of satisfied constraints with the total number of constraints for each task. The total number of constraints that is required to be satisfied for a task j_q is given by

$$N_{\text{req}}(j_q) = \sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\mathcal{D}_j(u,q) = 1\right) + \max\left(0, \max_{u=\{1,\ldots,|\mathcal{A}_j|\}} \left(\mathcal{D}_j(u,q)\right) - 1\right) \tag{3.3}$$

which counts the number of dependencies associated with j_q by examining the q^{th}

column of $\mathcal{D}_j$. Every entry equal to 1 in the q^{th} column of $\mathcal{D}_j$ counts as one constraint that needs satisfied, but for every entry larger than 1, only the first instance with that value is counted.

Example: The first column of $\mathcal{D}_j$ is given by $\{0,1,2,2,3,1,3,3\}^T$. This indicates that task j_1 may only be bid on if the following task combination has current winners: j_2 and j_6 and (j_3 or j_4) and (j_5 or j_7 or j_8). The total number of constraints that must be satisfied is 4 in this case as calculated by Equation (3.3).

The number of satisfied constraints is a function of an agent's current knowledge. Agent i calculating the number of satisfied constraints for task j_q uses Equation (3.4), which is conditioned on $\mathbf{z}_i$ and given by

$$\begin{aligned} n^{\text{sat}}(j_q|\mathbf{z}_i) = & \sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\left(z_{i(j_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}_j(u,q) = 1\right)\right) + \\ & \mathbb{I}\left(\textstyle\sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\left(z_{i(j_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}_j(u,q) = 2\right)\right) \geq 1\right) + \\ & \mathbb{I}\left(\textstyle\sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\left(z_{i(j_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}_j(u,q) = 3\right)\right) \geq 1\right) + \ldots \end{aligned} \tag{3.4}$$

The pessimistic bidding strategy specifies that agent i is allowed to bid on task j_q only if all of the dependency constraints for task j_q are satisfied. The Boolean $\text{canBid}_i(j_q)$ is used to keep track of this permission and is given by

$$\text{canBid}_i(j_q) = \begin{cases} \text{true} & \text{if } n^{\text{sat}}(j_q|\mathbf{z}_i) = N_{\text{req}}(j_q) \\ \text{false} & \text{otherwise} \end{cases} \tag{3.5}$$

If $\text{canBid}_i(j_q)$ is true, then the marginal score for j_q is calculated to determine if this task should be added to agent i's bundle. If $\text{canBid}_i(j_q)$ is false, then the marginal score for j_q is set to 0.

3.4 Optimistic Bidding Strategy

Tasks which are mutually dependent on at least one other task are given an optimistic bidding strategy which is a technique adapted from the Decentralized Task Elimination algorithm described in [36]. This section describes the bidding process for such

tasks.

Consider a task j_q with an optimistic bidding strategy. An agent who can benefit from adding task j_q to its bundle may do so, even if the number of satisfied constraints is smaller than the number required. The agent then keeps track of the number of iterations that pass where at least one of the tasks which j_q depends on remains unassigned. If too many iterations pass, the agent releases task j_q from its bundle. The task is released with the assumption that the other agents in the network are not interested in selecting the tasks j_q depends on, because other tasks are more valuable to these agents. To prevent an agent from repeatedly bidding on a task only to release it once it discovers no other agents are interested in performing the tasks which it depends on, the number of attempts is limited. Once an agent has run out of attempts on a particular task, they are no longer permitted to bid on that task unless all of the required constraints are satisfied. Thus the agent has transitioned to a pessimistic bidding strategy for this task. To formalize the technique, the following definitions are introduced:

Number of Iterations in Constraint Violation: $\boldsymbol{\nu}_i \triangleq \{\nu_{i1}, \ldots, \nu_{iN_t}\}$ is the list which keeps track of the number of CBBA iterations which have passed with agent i violating a constraint. Element notation $\nu_{i(j_q)}$ is used to indicate the number of iterations agent i has been winning element q of activity j while at least one other element in j is unassigned.

Permission to Bid Solo: $\mathbf{w}_i^{\text{solo}} \triangleq \{w_{i1}^{\text{solo}}, \ldots, w_{iN_t}^{\text{solo}}\}$ indicates which elements agent i is allowed to bid on as the first agent. The list is initialized to contain positive integers representing the number of attempts an agent is given to win the particular element. If $w_{i(j_q)}^{\text{solo}} > 0$, agent i is permitted to bid on task j_q even if no other elements of j have a winning agent. If a task j_q is released due to a timeout, or a timing constraint violation, $w_{i(j_q)}^{\text{solo}}$ is decremented by one.

Permission to Bid Any: $\mathbf{w}_i^{\text{any}} \triangleq \{w_{i1}^{\text{any}}, \ldots, w_{iN_t}^{\text{any}}\}$ indicates which tasks agent i is allowed to bid on given that at least one of the dependency constraints is satisfied for

that task. The array is initialized to contain positive integers in a similar manner to $\mathbf{w}_i^{\text{solo}}$, except the initial values are typically larger in magnitude. If $w_{i(j_q)}^{\text{any}} > 0$, and any other element of j has a winner, agent i is permitted to bid on task j_q. If an element j_q is released due to a timeout, or a timing constraint violation, $w_{i(j_q)}^{\text{any}}$ is decremented by one.

In addition to the information arrays, for each element q of an activity j, a *timeout* parameter, o_{j_q} is defined. At each iteration, agent i increments $\nu_{i(j_q)}$ for each task, j_q, for which agent i is the winner, but the other elements belonging to activity j are not filled. If at any time, $\nu_{i(j_q)}$ exceeds o_{j_q}, the task j_q must be released from $\mathbf{b}_i$, and the values $w_{i(j_q)}^{\text{solo}}$ and $w_{i(j_q)}^{\text{any}}$, are each decremented by 1.

An agent i is allowed to bid on task j_q if the following Boolean is true:

$$\text{canBid}_i(j_q) = \begin{cases} \text{true} & \text{if } \left(w_{i(j_q)}^{\text{any}} > 0 \wedge n^{\text{sat}}(j_q|\mathbf{z}_i) > 0\right) \vee \\ & \quad \left(w_{i(j_q)}^{\text{solo}} > 0\right) \vee \left(n^{\text{sat}}(j_q|\mathbf{z}_i) = N_{\text{req}}(j_q)\right) \\ \text{false} & \text{otherwise} \end{cases} \tag{3.6}$$

Thus if $w_{i(j_q)}^{\text{solo}} > 0$, then the agent has not yet exhausted its attempts to acquire the proper number of partnering agents for task j_q, and so it is allowed to bid even if 0 dependency constraints are satisfied. If $w_{i(j_q)}^{\text{solo}} = 0$, but $w_{i(j_q)}^{\text{any}} > 0$, then the agent has not yet exhausted its attempts to acquire the proper number of partnering agents, but at least one dependency constraint must be met in order for the agent to bid on task j_q. If both $w_{i(j_q)}^{\text{solo}} = 0$ and $w_{i(j_q)}^{\text{any}} = 0$, then agent i has exhausted its attempts to bid optimistically on task j_q, and may only bid if all of the dependency constraints for task j_q are satisfied.

Selection of o_{j_q} The parameter, o_{j_q} sets the timeout length for task j_q. An agent is allowed to keep task j_q in its bundle for at most o_{j_q} iterations while the number of satisfied constraints is less than the number required. The choice of o_{j_q} should ideally be a function of the expected network topology at the time task j_q is instantiated.

Consider an activity j with two mutually dependent elements, q and u. Suppose

agent i bids on task j_q and agent k bids on task j_u. If o_{j_q} is chosen too small, then agent i may release task j_q even though agent k bids on task j_u, because the information regarding agent k's bid took more than o_{j_q} iterations to reach agent i. Thus selecting o_{j_q} too small may lead to performance degradation.

Now consider the case where no agent is available to bid on task j_u. If o_{j_q} is chosen too large, then agent i waits for an unnecessarily large number of iterations before releasing task j_q. Thus selecting large timeout parameters may lead to longer convergence times.

Initialization of $\mathbf{w}_i^{\text{solo}}$ and $\mathbf{w}_i^{\text{any}}$ The initial values in $\mathbf{w}_i^{\text{solo}}$ and $\mathbf{w}_i^{\text{any}}$ are chosen as mission parameters. Selecting a higher initial value for $w_{i(j_q)}^{\text{solo}}$ gives agent i more attempts to bid on task j_q, however may adversely affect the runtime of the algorithm. Selecting too small of a value may reduce performance.

Example: Suppose $w_{i(j_q)}^{\text{solo}}$ and $w_{i(j_q)}^{\text{any}}$ are both initialized to 1. Then agent i has one attempt to bid optimistically on task j_q. If agent i is ever forced to release task j_q due a timeout or because of a temporal constraint violation, then $w_{i(j_q)}^{\text{solo}}$ and $w_{i(j_q)}^{\text{any}}$ are each decremented by 1, making them both 0. At this point in time agent i can no longer bid optimistically on task j_q; it has to wait until all dependency constraints are satisfied before rebidding. Agent i is forced to bid pessimistically on task j_q once $w_{i(j_q)}^{\text{solo}}$ and $w_{i(j_q)}^{\text{any}}$ are both 0. If an agent is forced to bid pessimistically too soon, the assignment may not reach its full potential.

3.5 Mutual Exclusions

In the baseline CBBA, an agent must be able to outbid the current winner of a task in order to be eligible to bid on that task. In Coupled-Constraint CBBA, to bid on task j_q, the marginal score for j_q must be greater than all of the tasks which are mutexed with task j_q. Therefore, task j_q is eligible to be bid on by agent i if Condition (3.7) is satisfied.

$$\left(c_{i(j_q)} > y_{i(j_u)} \vee \mathcal{D}(u,q) \neq -1\right) \ \forall u \in \mathcal{A}_j \tag{3.7}$$

3.6 Satisfying Temporal Constraints

Often in complex missions, there exist timing constraints between some of the tasks in the task set. The Coupled-Constraint CBBA framework allows a timing relationship to be specified between any pair of tasks belonging to the same activity. The pairwise timing constraints can be written compactly in the form of a *temporal constraint matrix*, $\mathcal{T}_j \in \mathbb{R}^{(|\mathcal{A}_j|\times|\mathcal{A}_j|)}$. The (q,u) entry of $\mathcal{T}_j$ specifies the maximum amount of time task j_q can begin after task j_u begins. Thus task j_u must begin at least $\mathcal{T}_j(q,u)$ before task j_q begins and at most $\mathcal{T}_j(u,q)$ after task j_q begins, which can be interpreted as the *relative time window* imposed on task j_u by task j_q. There exists no sign restriction on the entries of the temporal constraint matrix, which means that the relative time window a task j_q imposes on another task j_u does not have to contain the start time for task j_q. This makes it possible to specify constraints like *before*, or *after* mentioned in Section 3.1. If no timing constraint exists between task j_q and j_u, then $\mathcal{T}_j(q,u) = \mathcal{T}_j(u,q) = \infty$. The diagonal entries of matrix $\mathcal{T}_j$ are 0 by definition.

Note that a task j_q is only permitted to impose timing constraints on tasks which depend on j_q. If there is no dependency relationship between q and u, a timing constraint is not permitted: If $\mathcal{D}_j(q,u) \leq 0$ and $\mathcal{D}_j(u,q) \leq 0$, then necessarily $\mathcal{T}_j(q,u) = \mathcal{T}_j(u,q) = \infty$.

To satisfy coupled timing constraints, it is necessary that each agent be aware of the scheduled times of arrival of each of the tasks which have winners. Therefore, a new information list is introduced, $\mathbf{tz}_i \triangleq \{tz_{i1}, \ldots, tz_{i(N_t)}\}$. Entries are denoted $tz_{i(j_q)}$ for $j_q \in \mathcal{J}$, and the list keeps track of the arrival times for each task in $\mathbf{z}_i$ that has a winner. $tz_{i(j_q)} = \emptyset$ if task j_q has no winner according to agent i.

In the task selection phase of CBBA, agent i must calculate the interval of time that is valid for each task under consideration to determine if that task is feasible given their current path, and to determine the marginal score for that task. Agent i considering task j_q calculates the permissible start time by intersecting the original time window for task j_q with each of the coupled timing constraints imposed by the agents winning each of the tasks upon which task j_q depends. Agent i only considers

constraints imposed by elements of activity j with current winners according to agent i's knowledge, namely $\mathbf{z}_i$ and $\mathbf{tz}_i$.

Agent i needs to compute the interval $[\tau_{j_q}^{\min}, \tau_{j_q}^{\max}]$ where $\tau_{j_q}^{\min}$ is the minimum allowed start time for task j_q given the current knowledge of agent i and $\tau_{j_q}^{\max}$ is the maximum allowed start time for task j_q given the current knowledge of agent i. Recall that the original time window is given by $[\tau_{j_q}^{\text{start}}, \tau_{j_q}^{\text{end}}]$.

$\tau_{j_q}^{\min}$ is calculated by

$$\tau_{j_q}^{\min} = \max\left(\max_{u=\{1,\ldots,|\mathcal{A}_j|\},u\neq q} t_{\text{const}}^{\min}(j_u, j_q|\mathbf{z}_i), \tau_{j_q}^{\text{start}}\right) \tag{3.8}$$

where $t_{\text{const}}^{\min}(j_u, j_q|\mathbf{z}_i)$ is the constraint imposed on the start time of task j_q by the agent winning task j_u. It is given by equation (3.9), and is calculated for all elements of activity j except element q. Notice that the constraint is only active if agent i believes there is an agent winning task j_u, and j_q is dependent on j_u. $\tau_{j_q}^{\min}$ is taken to be the tightest active constraint including the original time window.

$$t_{\text{const}}^{\min}(j_u, j_q|\mathbf{z}_i) = \begin{cases} tz_{iu} - \mathcal{T}_j(u,q) & \text{if } z_{i(j_u)} \neq \emptyset \wedge \mathcal{D}_j(u,q) > 0 \\ -\infty & \text{otherwise} \end{cases} \tag{3.9}$$

Similarly, $\tau_{j_q}^{\max}$ is given by

$$\tau_{j_q}^{\max} = \min\left(\min_{u=\{1,\ldots,|\mathcal{A}_j|\},u\neq q} t_{\text{const}}^{\max}(j_u, j_q|\mathbf{z}_i), \tau_{j_q}^{\text{end}}\right) \tag{3.10}$$

where $t_{\text{const}}^{\max}(j_u, j_q|\mathbf{z}_i)$ is the constraint imposed on the start time of task j_q by the agent winning task j_u. It is given by Equation (3.11), and is again calculated for all elements of activity j except element q. The constraint is only active if agent i believes there is an agent winning task j_u, and j_q is dependent on j_u. $\tau_{j_q}^{\max}$ is taken to be the tightest active constraint including the original time window.

$$t_{\text{const}}^{\max}(j_u, j_q|\mathbf{z}_i) = \begin{cases} tz_{iu} + \mathcal{T}_j(q,u) & \text{if } z_{i(j_u)} \neq \emptyset \wedge \mathcal{D}_j(u,q) > 0 \\ \infty & \text{otherwise} \end{cases} \tag{3.11}$$

3.7 Enforcing Constraints

The purpose of the CBBA consensus phase is to resolve constraint violations in the task assignment. In the baseline CBBA, the consensus phase is only concerned with the conflict-freeness constraint. In the new framework, the consensus phase is augmented to enforce the additional coupled constraints: unilateral dependency constraints, mutual dependency constraints, mutual exclusions, and temporal constraints.

Tasks with Pessimistic Bidding Strategy During the consensus process, an agent i may find that another agent in the network outbid it for some task j_q in its bundle. When this happens, j_q as well as all tasks after j_q in bundle $\mathbf{b}_i$ are released. These tasks which are released may be depended on by other agents to satisfy the constraints associated with the tasks in their bundles. It is therefore necessary for each agent i to verify at each iteration, that the tasks in their bundle $\mathbf{b}_i$ which have a pessimistic bidding strategy have all dependency constraints satisfied. If an agent finds that it is winning a task j_q which depends on some task which is no longer assigned, it must release task j_q as well as all tasks in their bundle after j_q.

Tasks with Optimistic Bidding Strategy The protocol for optimistic bidding requires that each agent keep track of the number of iterations they have been winning a task with at least one violated dependency constraint. At each iteration, agent i counts the number of satisfied constraints for each task in its bundle, and compares it to the required number of satisfied constraints for that task. If it is less, $\nu_{i(j_q)}$ is incremented for each appropriate j_q.

Agent i also checks $\nu_{i(j_q)}$ for each task in its bundle. If $\nu_{i(j_q)} = o_{j_q}$ for any task, then j_q is released from the agent's bundle and $w^{\text{solo}}_{i(j_q)}$ and $w^{\text{any}}_{i(j_q)}$ are each decremented by 1. At this point, agent i has waited for o_{j_q} iterations of CBBA in hopes of acquiring all of the partnering agents that task j_q requires. The agent infers that if the constraints are not satisfied by this time, then the other agents in the network are either unable to select the tasks j_q depends on or choose not to select them because other tasks

result in a greater score. Task j_q is worth nothing to agent i unless all dependency constraints are met, and so it releases that task so it can select other tasks. Agent i also releases any task appearing in its bundle after j_q, but does not decrement $w^{\text{solo}}_{i(j_q)}$ or $w^{\text{any}}_{i(j_q)}$ for those tasks.

Enforcing Mutexes An agent i is allowed to bid on a mutexed task j_q as long as it can place a bid larger than the current winning bid of all tasks that are mutexed with j_q, and larger than the winning bid for task j_q itself. The agent placing the bid on task j_q assumes that the other agents will release the tasks mutexed with task j_q once they are informed that they were "outbid." Because of this protocol, it is possible for an agent to discover that it is currently winning a task j_u which is mutexed with another task j_q which it also believes to be assigned. Therefore it is necessary for each agent to evaluate whether they are winning a task which is mutexed with another assigned task at every iteration. Agent i is required to release task j_q from its bundle if it discovers another task j_u where $\mathcal{D}_i(u,q) = -1$ and $y_{i(j_u)} \geq y_{i(j_q)}$. Thus agent i is permitted to keep task j_q at a given iteration only if

$$\left(y_{i(j_q)} > y_{i(j_u)} \vee \mathcal{D}(u,q) \neq -1\right) \ \forall u \in \mathcal{A}_j, u \neq q \tag{3.12}$$

Temporal Constraints Temporal constraints may become violated during the task assignment process for a variety of reasons. Recall that the task selection phase of CBBA is performed by agent i independently of all other agents, except for the information agent i posses in $\mathbf{z}_i$ and $\mathbf{y}_i$. Therefore, it is possible for several agents to bid on elements of an activity in the same bidding round, possibly resulting in conflicting arrival times. Additionally, when an agent selects the start time for a task j_q, it is only required to consider the constraints imposed by the tasks which j_q depends on. There may exist an agent k who is currently winning another task j_u which unilaterally depends on j_q. Now the start time chosen by agent i for task j_q may invalidate the start time already selected for task j_u by agent k.

Therefore, it is necessary for each agent to check the temporal constraints for all

tasks in their bundle at each iteration. The method developed for enforcing timing constraints is described in this section for two cases: 1) temporal conflicts between tasks with unilateral dependency constraints, and 2) temporal conflicts between tasks with mutual dependency constraints. The process is as follows:

At each iteration, agent i checks each task $j_q \in \mathbf{b}_i$ for a temporal constraint violation.

$$\text{tempSat}(j_q) = \begin{cases} \text{true} & \text{if } \Big(\big(tz_{i(j_q)} \leq tz_{i(j_u)} + \mathcal{T}(q,u)\big) \wedge \big(tz_{i(j_u)} \leq tz_{i(j_q)} + \mathcal{T}(u,q)\big) \Big) \\ & \quad \forall u \in \{1,\ldots,|\mathcal{A}_j|\}, z_{i(j_u)} \neq \emptyset \\ \text{false} & \text{otherwise} \end{cases} \tag{3.13}$$

If $\text{tempSat}(j_q) = \textit{true}$ the agent i keeps task j_q as it is. However, if a task j_u is found whose start time is in violation with the start time for j_q, then agent i *may* have to release task j_q.

If task j_q unilaterally depends on j_u, then agent i releases j_q, because it is violating a constraint imposed by a task assumed to be higher priority. The agent may re-bid on task j_q in the next iteration, as long as it can select a start time that does satisfy the constraints.

If task j_u unilaterally depends on j_q, then agent i keeps task j_q in its bundle. The agent assumes that the agent winning task j_u will also realize that j_q and j_u are conflicted, and release task j_u.

If task j_q and j_u are mutually dependent, then a special procedure is followed: Assume that the score for a given task is monotonically decreasing within its time window. Then the agent arriving earliest (with respect to the start time of the task) is required to release that task. In this instance, it is assumed that the agent arriving later in their task's time window would have chosen an earlier start time if it were possible, since the score decays with time.

Given that agent i is winning task j_q and agent k is winning task j_u, and the start times are conflicted, then agent i releases task j_q if

$$tz_{i(j_q)} - \tau_{j_q}^{\text{start}} \leq tz_{i(j_u)} - \tau_{j_u}^{\text{start}} \tag{3.14}$$

Now each of the cases for the (j_q, j_u) task pair have been accounted for when there is a temporal constraint violation between them. If an agent is required to release a task j_q because of a temporal constraint violation, and task j_q has an optimistic bidding strategy, then $w^{\text{solo}}_{i(j_q)}$ and $w^{\text{any}}_{i(j_q)}$ are each decremented by one.

The entire procedure for enforcing constraints (both dependency constraints, and temporal constraints) is described by Algorithm 4, which is appended to the consensus phase of CBBA.

3.8 Algorithm Changes

CBBA with Time Windows becomes Coupled-Constraint CBBA when the following changes are made:

1. All subscripts j are changed to j_q because tasks now belong to activities
2. Step 2 of Algorithm 2 is changed such that $j_q \in \mathcal{J}_i^{\text{calc}}$ for all $j_q \in \mathcal{J}, \text{canBid}_i(j_q) = \text{true}$, where $\text{canBid}_i(j_q)$ is calculated by Equation (3.5) or Equation (3.6).
3. Step 3 of Algorithm 2 is changed such that $[\tau_j^{\min}, \tau_j^{\max}]$, calculated by Equations (3.8),(3.9),(3.9), and (3.10) is used in place of $[\tau_j^{\text{start}}, \tau_j^{\text{end}}]$.
4. Step 8 includes $tz_{ij^\star} = t_{ij^\star}$
5. Algorithm 4 is appended to the consensus phase of CBBA with Time Windows.

3.9 Examples of Encoding Real-World Constraints

The Coupled-Constraint CBBA framework possesses the capability to handle many constraint structures that would appear in real-world settings. However, encoding the constraints into the dependency matrix and the temporal constraint matrix may not be intuitive. This section provides several different examples of mission scenarios, and describes the corresponding constraints matrices.

Algorithm 4 Enforce Constraints and Update Permissions for Cooperative CBBA

1. **for all** $j_q \in \mathbf{b}_i$

   (a) Determine number of agents winning elements of j other than j_q

$$\begin{aligned} n^{\text{sat}}(j_q|\mathbf{z}_i) = \; & \textstyle\sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\left(z_{i(j_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}_j(u,q) = 1\right)\right) + \\ & \mathbb{I}\left(\textstyle\sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\left(z_{i(j_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}_j(u,q) = 2\right)\right) \geq 1\right) + \\ & \mathbb{I}\left(\textstyle\sum_{u=1}^{|\mathcal{A}_j|} \mathbb{I}\left(\left(z_{i(j_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}_j(u,q) = 3\right)\right) \geq 1\right) + \\ & \vdots \end{aligned}$$

   (b) $\text{mutexSat} = \begin{cases} \text{true} & \text{if } \left(y_{i(j_q)} > y_{i(j_u)} \vee \mathcal{D}(u,q) \neq -1 \; \forall u \in \mathcal{A}_j, u \neq q\right) \\ \text{false} & \text{otherwise} \end{cases}$

   (c) **if** task j_q has pessimistic bidding strategy **then**

      i. $$\text{depSat} = \begin{cases} \text{true} & \text{if } n^{\text{sat}}(j_q|\mathbf{z}_i) = N_{req}(j_q) \\ \text{false} & \text{otherwise} \end{cases}$$

      where $N_{\text{req}}(j_q)$ is calculated by Eq (3.3)

      ii. $$\text{tempSat} = \begin{cases} \text{true} & \text{if } \Big(\left(tz_{i(j_q)} \leq tz_{i(j_u)} + \mathcal{T}(q,u)\right) \wedge \left(tz_{i(j_u)} \leq tz_{i(j_q)} + \mathcal{T}(u,q)\right) \Big) \\ & \quad \vee \left(\mathcal{D}_j(u,q) \leq 0\right) \; \forall u \in \{1,\ldots,|\mathcal{A}_j|\}, z_{i(j_u)} \neq \emptyset \\ \text{false} & \text{otherwise} \end{cases}$$

      iii. **if** $\neg\text{mutexSat} \vee \neg\text{depSat} \vee \neg\text{tempSat}$ **then**

$$z_{i(j_q)} = \emptyset \qquad y_{i(j_q)} = 0$$

   (d) **if** task j_q has optimistic bidding strategy **then**

      i. **if** $n^{\text{sat}}(j_q|\mathbf{z}_i) < N_{\text{req}}(j_q)$ **then**

$$\nu_{i(j_q)} = \nu_{i(j_q)} + 1$$

      ii. $$\text{depSat} = \begin{cases} \text{true} & \text{if } \nu_{i(j_q)} < o_{j_q} \\ \text{false} & \text{otherwise} \end{cases}$$

      iii. $$\text{tempSat} = \begin{cases} \text{true} & \text{if } \Big(\left(tz_{i(j_q)} \leq tz_{i(j_u)} + \mathcal{T}(q,u)\right) \wedge \left(tz_{i(j_u)} \leq tz_{i(j_q)} + \mathcal{T}(u,q)\right) \Big) \\ & \quad \vee \left(tz_{i(j_q)} - \tau_{j_q}^{\text{start}} > tz_{i(j_u)} - \tau_{j_u}^{\text{start}}\right) \vee \left(\mathcal{D}_j(u,q) \leq 0\right) \\ & \quad \forall u \in \{1,\ldots,|\mathcal{A}_j|\}, z_{i(j_u)} \neq \emptyset \\ \text{false} & \text{otherwise} \end{cases}$$

      iv. **if** $\neg\text{mutexSat} \vee \neg\text{depSat} \vee \neg\text{tempSat}$ **then**

$$z_{i(j_q)} = \emptyset \qquad y_{i(j_q)} = 0$$

$$w_{i(j_q)}^{\text{solo}} = w_{i(j_q)}^{\text{solo}} - 1 \qquad w_{i(j_q)}^{\text{any}} = w_{i(j_q)}^{\text{any}} - 1$$

Priority Constraints: Track, Engage, Assess Damage Consider an environment containing a group of hostile targets with a priori estimated positions. The mission commander desires that each target be first, tracked, then attacked, then observed for battle damage assessment (BDA). In this case, track is primary, attack is secondary, and BDA is tertiary. A task may not be performed unless the associated task of higher priority is performed first (A target must be tracked before it is attacked). However, performing a task does not obligate the team to perform the associated task of lower priority (A target that is tracked does not have to be attacked).

For each known hostile, an activity is instantiated with three elements. The first element of each activity is a track task; the second is an attack task, and the third is a BDA task. The dependency matrix corresponding to the j^{th} hostile is given by

$$\mathcal{D}_j = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

and the temporal constraint matrix is given by

$$\mathcal{T}_j = \begin{bmatrix} 0 & -d_{(j_1)} & -d_{(j_1)} - d_{(j_2)} \\ \infty & 0 & -d_{(j_2)} \\ \infty & \infty & 0 \end{bmatrix}$$

Required Cooperation Many real-world scenarios involve a set of tasks, all of which must be assigned for any of them to be valid, and they must have simultaneous arrival. These include:

1. Cooperative transport
2. Rendezvous
3. Surprise attack
4. Joint sensing

The dependency matrix for a set of mutually required tasks of arbitrary size is given by

$$\mathcal{D}_j = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix}$$

and the temporal constraint matrix is given by

$$\mathcal{T}_j = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

Super-Additive Score Structure Imagine a highly maneuverable, high value target. The target may be tracked by a single UAV. However, since the target is evasive, it may elude the UAV. Therefore, it is desirable to send two UAVs to track the target if possible, severely reducing the probability of being evaded. The value of sending a single agent is estimated to be 10, and the value of sending two agents is estimated to be 50. The score structure is called super-additive because the combined score is greater than the sum of the individuals.

The naive method of encoding the score structure includes two instances of the track task, a primary and secondary. The primary track task is worth 10, and can be assigned independently of other tasks; the secondary track task is worth 40, and it is unilaterally dependent on the primary. This representation can introduce suboptimality because an agent is not allowed to place a bid on the secondary task until an agent has placed a bid on the primary. Furthermore, an agent has little incentive to bid on the primary task, so both tasks may go unassigned.

An alternative method exists for encoding this super-additive score structure. Instead of two instances of the track task, three track tasks are instantiated: A, B, and C. Track task A and B are mutually dependent, and they are each worth 25. Track task C is mutexed with both task A and task B. It is worth 10. With this structure, agents can bid optimistically with the hope of getting a score of 25.

However, they only receive a reward of 25 if another agent agrees to cooperate with them on the track task. If an agent bids optimistically on task A, but finds no agent is available to perform task B, they are forced to release task A from their bundle. However, at that point the agent may choose to bid on task C, even though it is only worth 10. The dependency matrix for this track activity is given by

$$\mathcal{D}_{track} = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

and the temporal constraint matrix is given by

$$\mathcal{T}_{track} = \begin{bmatrix} 0 & 0 & \infty \\ 0 & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix}$$

Not During Consider a scientific operation, such as robotic planetary exploration, where a series of measurements are taken by a fleet of robots. Suppose there are two measurements that are to be taken: measurement task A, and measurement task B. Both measurements must be taken for either of them to be valid, however, they cannot be taken simultaneously, due to equipment interference. The constraints can be encoded by splitting one of the tasks into two instances. In this case, task B is split into B^- and B^+. In this example, measurement task A is the first element of the measurement activity, B^- is the second element, and B^+ is the third element. Task B^- is constrained such that it must end before task A begins, and task B^+ is constraints such that it must begin after task A ends. Since task B should not be performed twice, tasks B^- and B^+ are mutexed. The dependencies are written such that task B^- depends on task A, task B^+ depends on task A, and A depends on either B^- or B^+. The dependency constraint matrix for this measurement activity

is:

$$\mathcal{D}_{measure} = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & -1 \\ 2 & -1 & 0 \end{bmatrix}$$

and the temporal constraint matrix is given by

$$\mathcal{T}_{measure} = \begin{bmatrix} 0 & \infty & -d_A \\ -d_B & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix}$$

3.10 Numerical Results for Coupled-Constraint CBBA

3.10.1 Simulation Description

Coupled-Constraint CBBA is compared to the baseline CBBA using a simulation to determine the merit of the assignments achieved. The mission scenario that is simulated is a cooperative search, track, and engage mission. Participating in the mission, are three types of agents. Agents of type A are weaponized uninhabited aerial vehicles (WUAV), which are capable of engaging hostile targets. Agents of type B are Sensor UAVs (SUAVs), and they are capable of providing the WUAVs with accurate measurements during an engagement. Finally, agents of type C are SUAVs, but are only capable of image capture; they cannot provide measurements to a WUAV. Agent capabilities are described in the Table 3.2.

There exist several objects of interest in the mission environment. Some of the objects are confirmed hostiles, and it is desired that all hostiles be engaged. Some of the objects are of unconfirmed identity, and more intelligence is desired regarding these objects. The two different types of objects each have a specific activity associated with them.

Activity type 1 is a *service hostile* activity and includes the following: hostile targets may be engaged, which requires one agent of type A to strike the target and one agent of type B to provide position measurements. The strike portion of

Table 3.2: Agent Parameters for Simulated Mission Scenario

Agent Parameters		
Type A: WUAV	Units Available Capability Max Velocity	3 Weapons Release 50 m/s
Type B: SUAV 1	Units Available Capability Max Velocity	5 Sensing- Measurements to WUAV 25 m/s
Type C: SUAV 2	Units Available Capability Max Velocity	8 Sensing- Image only 15 m/s

the activity takes 120 seconds, and the WUAV and the SUAV must plan to arrive within 20 seconds of each other. If one type A and one type B agent are assigned to a service hostile activity, a type C agent may also be assigned to perform a battle damage assessment (BDA). The BDA takes 180 seconds, and must begin at least 60 seconds after the strike is completed. If there is no strike package available to engage the hostile target, a sensor UAV may be sent to observe the hostile, but it is assumed that the reward for performing the intelligence gathering task is much less than that for performing the strike task. The maximum score for a strike task is 100, the maximum score for a BDA is 50, and the maximum score for a intelligence gathering task is 10. The time window for the activity is 600 seconds.

Activity type 2 is a tracking activity and involves visiting one of the objects with unknown identity. This type of activity contains a single element which is an information gathering task that can be performed by either a type B agent or a type C agent. The duration is 60 seconds, and the maximum score for a tracking activity is 10.

Half of the activities in the mission are of type 1, and half of the activities are of type 2. The dependency constraint matrix for all activities of type 1 are given by $\mathcal{D}_A$ where $A \leq \frac{|A|}{2}$, and the dependency constraint matrix for all activities of type 2 are

given by $\mathcal{D}_B$ where $B > \frac{|\mathcal{A}|}{2}$.

$$\mathcal{D}_A = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \quad \mathcal{D}_B = \begin{bmatrix} 0 \end{bmatrix}$$

The temporal constraint matrix for all activities of type 1 are given by $\mathcal{T}_A$ where $A \leq \frac{|\mathcal{A}|}{2}$, and the temporal constraint matrix for all activities of type 2 are given by $\mathcal{T}_B$ where $B > \frac{|\mathcal{A}|}{2}$.

$$\mathcal{T}_A = \begin{bmatrix} 0 & 20 & -180 & \infty \\ 20 & 0 & -180 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{bmatrix} \quad \mathcal{T}_B = \begin{bmatrix} 0 \end{bmatrix}$$

The simulation environment is 10 kilometers by 10 kilometers. Activity locations are generated randomly, uniform in the environment. Agent initial positions are also chosen randomly, uniform in the environment. The number of each agent type is fixed: 3 of type A, 5 of type B, 8 of type C. The number of activities is varied as a parameter, with half of the activities being of type 1, and half type 2. The time windows for each activity are chosen such that the start time is random, uniform on [0,300] seconds. Also note that the maximum bundle length, L_t is set to 4 for all simulations.

Scoring The scoring for a given assignment is calculated such that only tasks which are assigned without constraint violation count toward the assignment score. Also, the baseline CBBA has no way to account for timing constraints, so the only way to enforce them is to shrink the time window for the two elements of the strike task to be 20 seconds long, and set the time window of the BDA to begin after the strike window ends.

3.10.2 Simulation Results and Discussion

A Monte Carlo simulation is conducted over missions with randomly generated initial conditions. The number of activities is varied as a parameter and 80 trials are simulated for each case. The simulation results show that CCBBA outperforms the baseline CBBA in terms of the assignment score, which is the objective function given by Equation (3.1). The average assignment score as a function of the number of activities is shown in Figure 3-2.

Notice that the score for the assignments generated by the Baseline CBBA increases as the number of activities increase, but only when the number of activities is about 10 or fewer. As the number of activities increases beyond 10, the score for CBBA-generated assignments flattens out, and actually begins to decrease when the number of activities exceeds approximately 20. The reason for this behavior is that when the ratio of tasks to agents, $\frac{N_t}{N_u}$ is very low, most of the tasks get assigned, but as the ratio increases, all tasks are not assigned (Recall that the bundle length is limited to $L_t = 4$ in this case). When a very high percentage of the tasks in the task set are assigned, the dependency constraints for the tasks are naturally satisfied, even if the algorithm is not explicitly enforcing the dependency constraints. However, as $\frac{N_t}{N_u}$ grows larger, each agent has increasingly more task options available. Given that an agent selects a task, j_q, the probability that all tasks which j_q depends on are also assigned decreases as $\frac{N_t}{N_u}$ increases when using the Baseline CBBA.

Example: Consider a simple scenario with two agents and 10,000 activities, each with two mutually dependent elements. The bundle length is fixed at 5. The probability that the agents both select the same activity is very low in this case.

Contrast this behavior to the behavior of the score for CCBBA-generated assignments. The score increases with the number of tasks at a much higher rate compared to CBBA, and does not posses this tendency to flatten out suddenly. The reason for this is that CCBBA is explicitly enforcing the coupled constraints, whereas CBBA is unable to do so.

The conclusion that can be drawn is that in order to maximize the score of the

assignment in a complex mission, the coupled constraints must be dealt with explicitly. Furthermore, as $\frac{N_t}{N_u}$ increases, the performance difference between CBBA and CCBBA becomes exaggerated, thus accounting for the coupling becomes increasingly important.

Figure 3-3 further validates the conclusions drawn from Figure 3-2. This figure compares the total number of tasks assigned by each algorithm as well as the number of feasible tasks assigned. A feasible task is defined to be a task which satisfies all constraints in the problem. Notice that the total number of tasks assigned is roughly the same for each approach. However, for the baseline CBBA, the percentage of tasks assigned which satisfy the constraints decreases as the number of activities increases, thus adversely affecting the assignment score for the baseline CBBA. The figure also shows that for CCBBA, all assigned tasks satisfy the constraints, which is reflected in the assignment score.

Figure 3-4 shows the increase in computation for CCBBA compared to CBBA. The metric for comparing computation is the required number of score calculations to arrive at an assignment summed over the whole network. The comparison of computation shows that for this mission scenario, the computation requirements are approximately linear for CBBA, and approximately quadratic for CCBBA.

Figure 3-5 compares the communication requirements for the two algorithms. The metric for comparing communication is the total number of messages parsed during the assignment process summed over the network. This figure shows that for this choice of mission parameters, CBBA has approximately constant communication requirements with respect to the number of tasks, whereas CCBBA has approximately linear communication requirements. The additional complexity is associated with the optimistic bidding process, because agents must put forth additional effort to discover that a task will not have the required number of satisfied constraints.

CBBA is known to have polynomial computation and communication requirements. Figure 3-4 and Figure 3-5 show that CCBBA is more computationally expensive compared to CBBA, and requires additional message passing. However, CCBBA preserves the polynomial time properties of CBBA. The penalty appears to be a

higher order polynomial in runtime complexity, and so there is a trade-off between enforcing the coupled constraints and generating assignments quickly.

The consequences of the additional computation and communication are that both the size of the network, and the number of tasks that can be handled by CCBBA are more restricted compared to CBBA. However, CCBBA still provides a tractable framework for complex missions involving large teams and many tasks. In this mission scenario, there were 16 agents and up to 30 activities, (75 tasks with coupling constraints), and the assignments were generated in less than 40 seconds. (Note that this runtime is for a simulation in the Matlab environment. The runtime would be significantly reduced for a C++ implementation)

Interesting work has been done on an asynchronous communication version of CBBA [35], which offers vast reduction in runtime, and significantly fewer messages required for convergence. Future work involves integrating the Coupled Constraint CBBA extension into the asynchronous CBBA framework. The result would be a very fast solution for solving constrained task assignment problems, and it could be implemented in real-world settings.

3.10.3 Convergence Properties

The baseline CBBA is guaranteed to converge if the score structure satisfies the DMG property described in Section 2.4. However, the convergence proof does not apply to CCBBA since an agent is sometimes required to release a task it was not outbid on, to satisfy the coupled constraints. Although a formal convergence proof has not yet been constructed for CCBBA, empirical evidence suggests that CCBBA preserves the robust convergence properties of CBBA. A total of 1120 task assignments were generated for the comparisons shown in Figures 3-2 through 3-5, and all cases converged 100 percent of the time. A formal proof is the subject of future work on this algorithm.

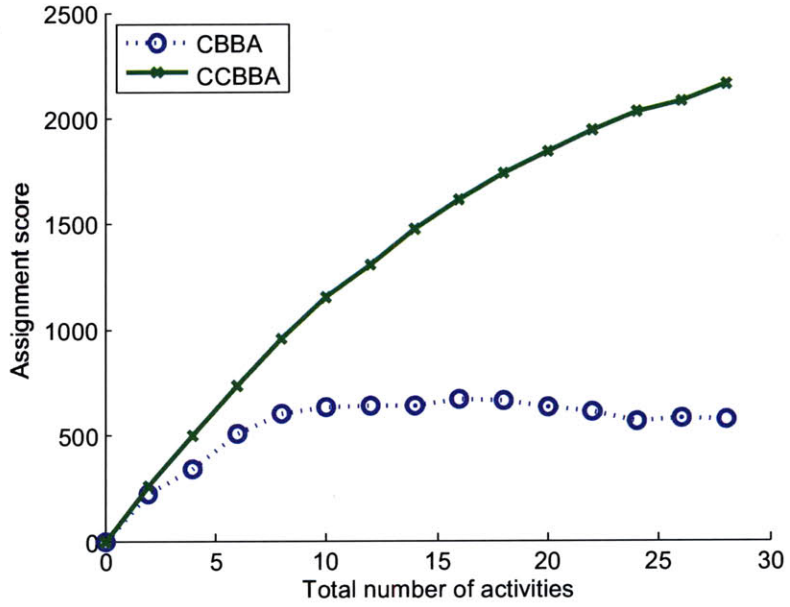

Figure 3-2: Cooperative CBBA vs. Baseline CBBA, Assignment Score

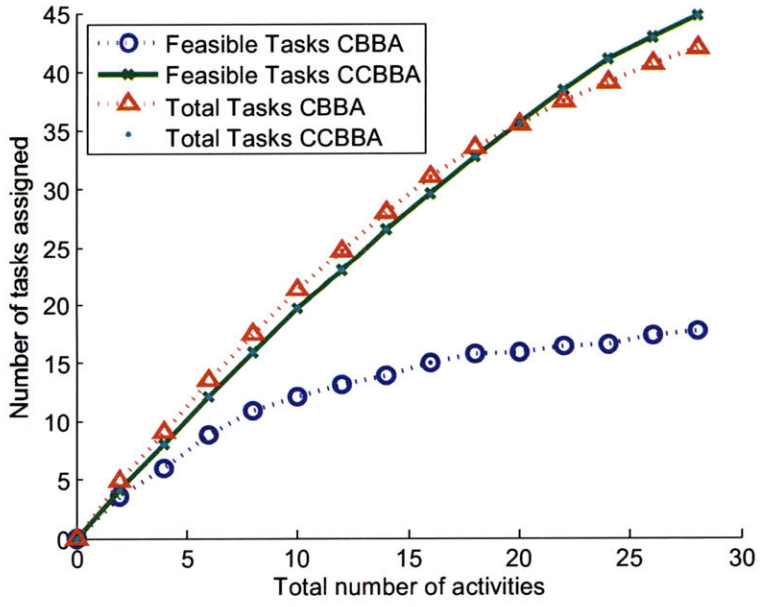

Figure 3-3: Cooperative CBBA vs. Baseline CBBA, Number of Tasks Assigned

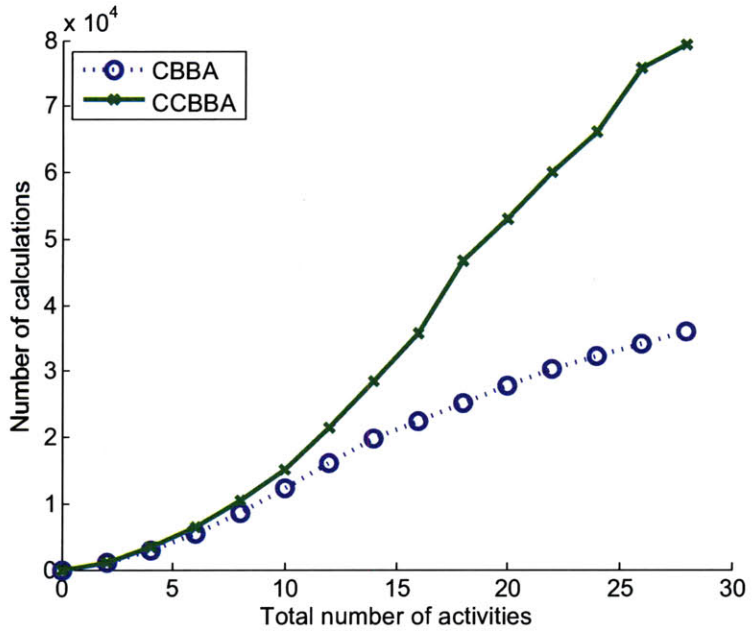

Figure 3-4: Cooperative CBBA vs. Baseline CBBA, Computation Complexity

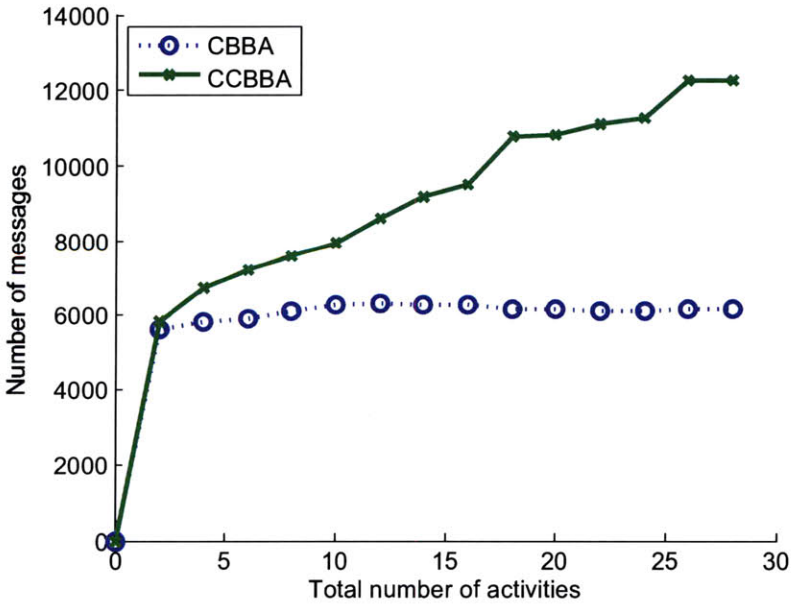

Figure 3-5: Cooperative CBBA vs. Baseline CBBA, Communication Complexity

3.11 Summary

This chapter presents an algorithm extension to CBBA for handling coupled constraints in the task set. The additional constraints handled by the new framework are unilateral dependency constraints, mutual dependency constraints, mutual exclusions, and temporal constraints. Coupled-Constraint CBBA (CCBBA) is introduced as an efficient mechanism for solving the constrained task assignment problem.

The task set is partitioned into *activities*, which are groups of tasks that share coupled constraints. Two bidding strategies are developed. If a task is not mutually dependent on any other task, it is given a pessimistic bidding strategy: agents wait to bid on the task until all of its dependency constraints are satisfied. If a task is mutually dependent on at least one other task, it is given an optimistic bidding strategy: agents may bid on a task even if all dependency constraints are not satisfied, as long as they have not exhausted their limited attempts on that task. An agents is also required to release a task if it has been winning the task for too long without the required number of satisfied constraints.

The quality of assignments for CCBBA are compared to CBBA in numerical simulations. The simulation results demonstrate that explicitly handling the coupling in the task set is necessary in complex missions to maximize the effectiveness of the assignments. CCBBA requires more computation, and more communication than CBBA, but the effects of this are expected to be reduced when the asynchronous version of CCBBA is developed.

Chapter 4

Satisfying Refueling Constraints with CBBA

Missions are often longer in duration than the maximum time the agents can operate on a single tank of fuel, or a single battery charge. Mission planning must therefore account for the agent's finite operational time, and schedule refuel times appropriately. This chapter discusses the process of efficiently scheduling refuel times as part of the task assignment process. Refuel is treated as a hard constraint, because in this context it is assumed that if an agent fails to refuel the agent is lost. The assumption is appropriate for many complex missions, in particular those where the agents are operating in unknown or hostile environments.

4.1 Problem Statement

Consider a world with N_u agents and N_t tasks. Recall that the set of all agent indices is denoted $\mathcal{I} \triangleq \{1, \ldots, N_u\}$, while the set of all task indices is denoted $\mathcal{J} \triangleq \{1, \ldots, N_t\}$. Each task, $j \in \mathcal{J}$, has a score function, $S_j(a_j, t_j)$ which represents the value that task adds to the mission as a function of $a_j \in \mathcal{I} \cup \{\emptyset\}$, the index of the agent assigned to task j, and $t_j \in \mathbb{R}_+ \cup \{\emptyset\}$ the time the agent plans to arrive at the j^{th} task location. The goal of the task planner is to assign agents to tasks in such a way that the cumulative score over all tasks in the task set is maximized. The objective function

is expressed

$$\max \sum_{j=1}^{N_t} S_j(a_j, t_j) \tag{4.1}$$

Each agent i in the network has a finite endurance that is a function of its maximum fuel capacity, $m^i_{fuelmax}$ and nominal fuel burn rate, $\dot{m}^i_{\text{fuel}}$ (assumed time invariant). At time t, agent i has fuel $m^i_{\text{fuel}}(t)$ and can remain operational for at most

$$t^i_{\text{remain}}(t) = \frac{m^i_{\text{fuel}}(t)}{\dot{m}^i_{\text{fuel}}} \tag{4.2}$$

before it needs to refuel. At time t, it is desired to assign the N_t tasks to the agents such that each agent has one refuel time t^i_{refuel} subject to

$$t^i_{\text{refuel}} \leq t + t^i_{\text{remain}}(t) - t_{\text{margin}} \tag{4.3}$$

where t_{margin} is a buffer for safety purposes. In this problem, it assumed that there are at least as many refueling berths at the base as there are agents, such that the base can service an arbitrary number of agents simultaneously.

4.2 Approach

The Consensus-Based Bundle Algorithm can be modified to handle situations where scheduling a particular event is a hard constraint. In the CBBA framework refueling may be treated as a task, but not in exactly the same sense as any other task. The refuel task differs by the fact that it is mandatory where as all other tasks are optional.

The index for the refuel task specific to agent i is denoted r_i. Note that parameters like location or duration that are associated with task r_i may differ between agents. Since the refuel task r_i is always a part of agent i's future plan, it can be kept track of separately, and does not need to be explicitly inserted into $\mathbf{p}_i$ or $\mathbf{b}_i$. The scheduled refuel time for agent i is denoted $\tau_{i(r_i)}$ to be consistent with the notation for other task start times.

During the task selection phase, an agent i may add tasks to its bundle $\mathbf{b}_i$ and to

its path $\mathbf{p}_i$ as long as the agent can also fit the refuel task into its path such that it can travel to each of the tasks on time. If a task is added to the path such that the refuel task no longer fits, the task is not permitted to be added, and thus its marginal score is zero.

The main difference between this version of CBBA and the baseline CBBA, is in the way the marginal scores for the tasks are calculated: Task j may be added to agent i's bundle only if j can be inserted into the path $\mathbf{p}_i$ such that the set of times that the refuel task can begin, denoted $\tau^{\text{allowed}}_{i(r_i)}$ is non-empty.

Given a path $\mathbf{p}_i$, $\tau^{\text{allowed}}_{i(r_i)}$ is calculated

$$\tau^{\text{allowed}}_{i(r_i)}(\mathbf{p}_i) = [t_0, t_0 + t^i_{\text{remain}}(t) - t_{\text{margin}}] \setminus \Omega_{i1}(\mathbf{p}_i, (r_i)) \setminus \Omega_{i2}(\mathbf{p}_i, (r_i)) \cdots \setminus \Omega_{i|\mathbf{p}_i|}(\mathbf{p}_i, (r_i)) \tag{4.4}$$

where $\Omega_{in}(\mathbf{p}_i, r_i)$ is the interval of time that is invalid as a start time for refuel task r_i because of the n^{th} task in path $\mathbf{p}_i$. Specifically, task r_i is not allowed to begin inside $\Omega_{in}(\mathbf{p}_i, r_i)$, because it would be temporally too close to τ_{in}, meaning that agent i would be either late to task p_{in}, or late to task r_i. $\Omega_{in}(\mathbf{p}_i, r_i)$ is given by

$$\Omega_{in}(\mathbf{p}_i, r_i) = [\tau_{in} - d_{r_i} - \Delta(\text{tloc}_{r_i}, \text{tloc}_{p_{in}}), \tau_{in} + d_{p_{in}} + \Delta(\text{tloc}_{p_{in}}, \text{tloc}_{r_i})] \tag{4.5}$$

Thus, task j may be added to agent i's path at location n_j only if

$$\tau^{\text{allowed}}_{i(r_i)}(\mathbf{p}_i \oplus_{n_j} j) \neq \emptyset \tag{4.6}$$

r_i is taken to be the maximum time over $\tau^{\text{allowed}}_{i(r_i)}(\mathbf{p}_i \oplus_{n_j} j)$, so that the agent can remain in service as long as possible while servicing as many tasks as possible. If no n_j can be found that satisfies condition (4.6), then marginal score $c_{ij} = 0$, because adding a task is not allowed to prevent an agent from refueling. The procedure for calculating marginal scores for CBBA with Refuel Constraints is given by Algorithm 5, which replaces Algorithm 3 in the CBBA with Time Windows framework. The consensus phase of CBBA with Refuel Constraints is unaltered from the original CBBA.

Algorithm 5 Calculate marginal score of task j given $\mathbf{p}_i$ for agent i, CBBA with Refuel Constraint

1. **For All** $n_j \in \{0, \ldots, (l_b)\}$ **where** $F_{ij}(n_j) = \text{true}$

   (a) Find τ^{earliest} and τ^{latest} such that agent i has sufficient time to carry out the task before j and the task after j.

$$\tau^{\text{earliest}} = \begin{cases} \max\left((\tau_j^{\text{start}}), (\tau_{cur} + \Delta(\text{aloc}_i, \text{tloc}_j))\right) & \text{if } n_j = 0 \\ \max\left((\tau_j^{\text{start}}), (\tau_{i(n_j)} + d_{p_{i(n_j)}} + \Delta(\text{tloc}_{p_{i(n_j)}}, \text{tloc}_j))\right) & \text{otherwise} \end{cases}$$

$$\tau^{\text{latest}} = \begin{cases} \tau_j^{\text{end}} & \text{if } n_j = |\mathbf{p}_i| \\ \min\left((\tau_j^{\text{end}}), (\tau_{i(n_j+1)} - d_j - \Delta(\text{tloc}_j, \text{tloc}_{p_{i(n_j+1)}}))\right) & \text{otherwise} \end{cases}$$

   (b) Initialize score and time for n_j location to be zero, which are rewritten if $[\tau^{\text{earliest}}, \tau^{\text{latest}}]$ is feasible.

$$s(n_j) = 0; \quad t(n_j) = \emptyset; \quad t^{\text{refuel}}(n_j) = \emptyset$$

   (c) **if** $(\tau^{\text{earliest}} > \tau^{\text{latest}})$ **then**

$$F_{ij}(n_j) = \text{false}$$

   else

   i.

$$s(n_j) = \max_{t \in [\tau^{\text{earliest}}, \tau^{\text{latest}}]} S_j(i,t)$$

$$t(n_j) = \arg\max_{t \in [\tau^{\text{earliest}}, \tau^{\text{latest}}]} S_j(i,t)$$

   ii.

$$\tau_{i(r_i)}^{\text{allowed}}(\mathbf{p}_i) = [t_0, t_0 + t_{\text{remain}}^i(t) - t_{\text{margin}}] \setminus \Omega_{i1}(\mathbf{p}_i, (r_i)) \setminus \Omega_{i2}(\mathbf{p}_i, (r_i)) \cdots \setminus \Omega_{i|\mathbf{p}_i|}(\mathbf{p}_i, (r_i))$$

   iii. **if** $(\tau_{i(r_i)}^{\text{allowed}} \neq \emptyset)$ **then**

$$t^{\text{rf}}(n_j) = \max \tau_{i(r_i)}^{\text{allowed}}$$

   else

$$s(n_j) = 0; \quad t(n_j) = \emptyset; \quad t^{\text{refuel}}(n_j) = \emptyset$$

2. Record marginal score for task j, the location to insert it if chosen, and the time to perform it if chosen, and the refuel time if j is chosen.

$$c_{ij} = \max_{n_j \in \{0,\ldots,|\mathbf{p}_i|\}} s(n_j) \quad n_{ij} = \arg\max_{n_j \in \{0,\ldots,|\mathbf{p}_i|\}} s(n_j) \quad t_{ij} = t(n_{ij}) \quad t_{i(r_1)} = t^{\text{rf}}(n_{ij})$$

4.3 Numerical Comparison of Refuel Time Schedulers

A Monte Carlo simulation is performed to study the quality of the assignments generated by CBBA with Refuel Constraints. The algorithm is compared to a refuel algorithm introduced in [31], which is based on CBBA as well, but requires solving a Mixed Integer Linear Program (MILP) during the task selection phase.

The simulation environment was developed as part of the Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance (OPS-USERS) program. A brief summary of the OPS-USERS mission scenario is provided here; see Section 5.2.1 for a more complete description of the OPS-USERS architecture.

4.3.1 Summary of OPS-USERS Mission Setting

The OPS-USERS mission scenario is multi-objective. The environment contains some number of targets, unknown a priori, and the agents are required to find as many of the targets as possible before the end of the mission. State estimates of each target are also desired, so the agents are required to keep track of each target's estimated position and velocity. Therefore, the mission planner must balance the amount of effort spent searching for new targets, and the amount of effort spent keeping accurate state estimates of the known targets.

To enable adequate time for searching, targets are not required to be tracked continuously; instead, track tasks are generated for each known target of interest. A track task consists of observing a target with the agent's sensor package. Tracking a target for a period of time decreases the uncertainty associated with that target's position and velocity. At the end of a track task, the tracking agent also instantiates a new track task so that the target will be revisited in the future. The desired revisit time is calculated based on the expected growth rate of the target's position uncertainty. The task assignment algorithm is responsible for planning which agent

should perform each track task and decide when each agent should arrive at their tasks subject to the dynamic constraints.

When agents have no tasks that require their immediate attention, their default action is to search. Search is conducted via a receding horizon optimized trajectory calculated over a dynamic probability map which characterizes the position estimates of targets yet-to-be found. Search strategy is the subject of Chapter 5, and the reader is referred to this chapter if more details are desired.

The purpose of this study is to quantify the quality of assignments generated by CBBA using each of the two refuel schedulers in question. The primary responsibility of the task assignment algorithm in this study is to assign track tasks while meeting all refuel constraints. Therefore, the primary figure of merit is the percentage of time each target is tracked after it is found. The secondary metric for evaluation is the average reaction time to newly discovered tasks.

4.3.2 Parameters of Mission Simulation

For this mission scenario, there are 10 targets, 7 of which require tracking, and both of these numbers are unknown to the agents a priori. Target positions are initialized randomly, and their speeds are a random constant, drawn from a uniform distribution of 5-15 meters per second. The team of autonomous agents comprises one Weaponized UAV (WUAV), one autonomous watercraft, and two Sensor UAVs. Agents are intensionally given small fuel capacity, to explore the effects of the two refueling strategies. Maximum endurance ranges from 200-400 seconds and the mission duration is 30 minutes. Table 4.1 lists important agent parameters for the mission scenario tested. A total of 40 Missions were simulated for each of the algorithms tested, and in the interest of fairness, the same initial conditions were used for both.

4.3.3 Simulation Results

Detailed log files are recorded for each simulated mission, and important mission metrics are then extracted. The metrics are as follows

Table 4.1: Agent Parameters for Simulated Mission Scenario, Refueling Algorithm Comparison

Agent Parameters		
Type A: WUAV	Units Available	1
	Capability	Weapons Release, Detecting
	Max Velocity	100 m/s
	Max Endurance	400 sec
Type B: Autonomous Watercraft	Units Available	1
	Capability	Detecting, Tracking
	Max Velocity	25 m/s
	Max Endurance	200 sec
Type C: Sensor UAV	Units Available	2
	Capability	Detecting, Tracking
	Max Velocity	75 m/s
	Max Endurance	300 sec

1. Figure 4-1 gives a histogram of the percentage of time each target was tracked after it was found. In the OPS-USERS framework a target is considered lost when its position uncertainty exceeds a set threshold. The threshold is based on the agents' sensor field of view and quantifies the maximum allowable uncertainty such that the target can be re-acquired. The *percentage of time targets tracked* metric is defined to be the amount of time a target is not lost, normalized by the amount of time it is known to exist. For a given mission, this metric is averaged over all targets found.

2. Figure 4-2 gives a histogram of the average reaction to each newly discovered target. Reaction time is defined to be the amount of time between a target's discovery and the first time it is tracked. If the agent that makes the discovery is able to track the target right away, the reaction time is zero. However, the discovering agent is not always able to perform the track task because of constraints in the problem such as refueling. Reaction time for a mission is averaged over all targets found.

3. Figure 4-3 provides a histogram of the number of targets found during each mission. Finally,

4. Figure 4-4 provides a histogram of the percentage of the environment that was searched during each mission. Percentage of environment searched is defined as the total number of cells that were completely observed at least once during the mission normalized by the total number of cells in the environment.

Each performance metric is averaged across the 40 trials and presented in Table 4.2. The results of the simulation show similar performance in the primary metric, percentage of time targets tracked. Figure 4-1 shows that the distribution of time targets are tracked is statistically similar for both refuel schedulers. However, Table 4.2 shows that there is a 25 percent decrease in average reaction time to new targets when using CBBA with Refuel Constraints versus using the MILP refuel scheduler. Consider the distributions of reaction times displayed in Figure 4-2. The reaction time for CBBA with Refuel Constraints is more concentrated toward the smaller reaction times, whereas the distribution for the MILP refuel scheduler has a longer tail, extending out to 450 seconds. This indicates that the refueling time scheduler presented in this chapter performs approximately equal to, and in some aspects better than, the MILP refuel scheduler.

Figures 4-3 and 4-4 show that the search performance suffered slightly as a result of using CBBA with Refuel Constraints instead of the original MILP. The cause of the performance degradation is likely tied to the trade-off between searching and tracking. When the system spends more effort tracking known targets, fewer resources are available to search for new targets.

The main benefit of CBBA with Refuel Constraints is the reduced computational load for the task planner compared to the MILP refuel scheduler. A short study involving 8 agents and 20 targets was conducted to explore the computational cost of both refuel strategies. It was observed that CBBA running the MILP refuel scheduler could spend as much as 300 ms in the task selection phase. On the contrary, CBBA with Refuel Constraints rarely spent more than 2-3 ms in the task selection phase. For this size of mission scenario, CBBA with Refuel Constraints can run two orders of magnitude faster than the original refuel algorithm. This study indicates that CBBA with Refuel Constraints allows scalability to much larger problems with more agents

Table 4.2: Average Performance Metrics for Refuel Algorithm Comparison

Metric	MILP	CBBA with Refueling Constraints
Percent total area covered	64.9%	51.6%
Avgerage number of targets found	9.25	8.70
Percent time targets tracked after discovery	30.1%	31.5%
Reaction time to new target	99.5 sec	74.4 sec

and more tasks compared to the MILP refuel scheduler.

4.4 Summary

This chapter presents a method for solving the task assignment problem under refuel constraints. The new method is compared to an existing approach, which requires solving a Mixed Integer Linear Program at every iteration. Numerical results confirm that the new strategy outperforms the original strategy in the quality of assignments. Furthermore, the computational complexity is greatly reduced enabling scalability to larger problems.

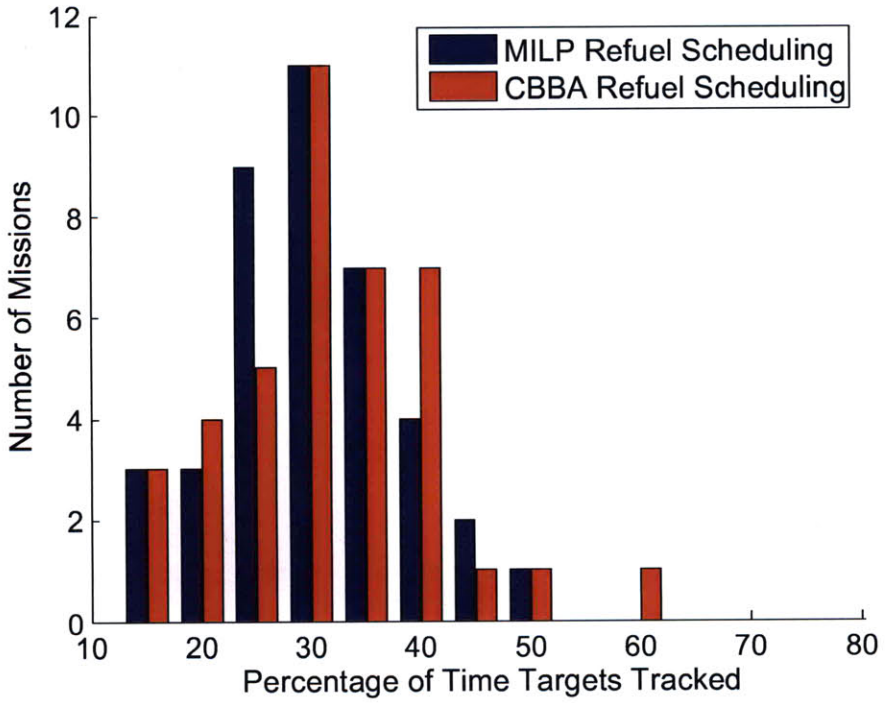

Figure 4-1: Percentage of Time Targets Tracked

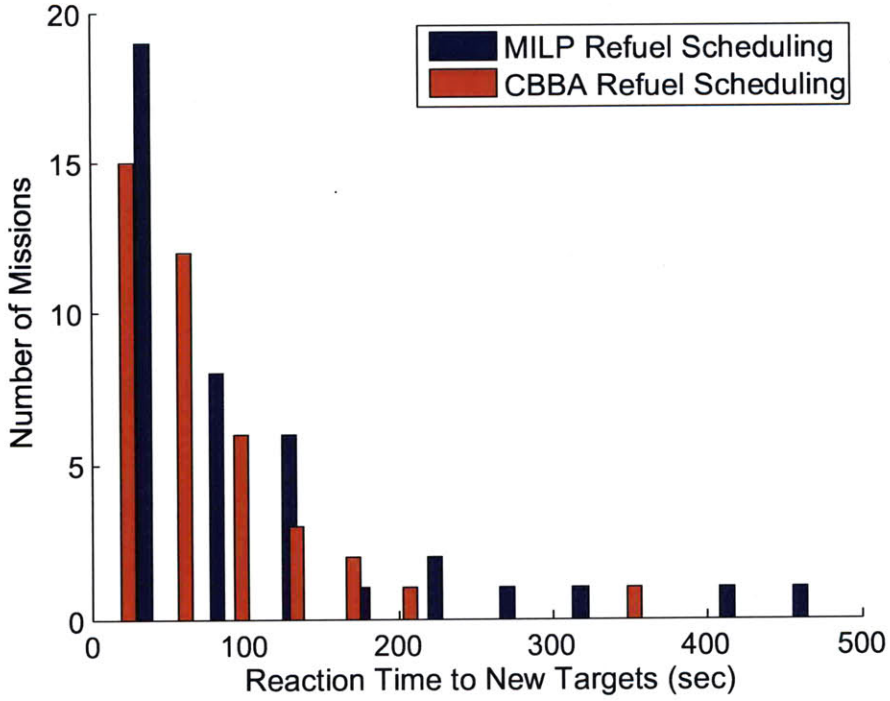

Figure 4-2: Reaction Time to New Targets

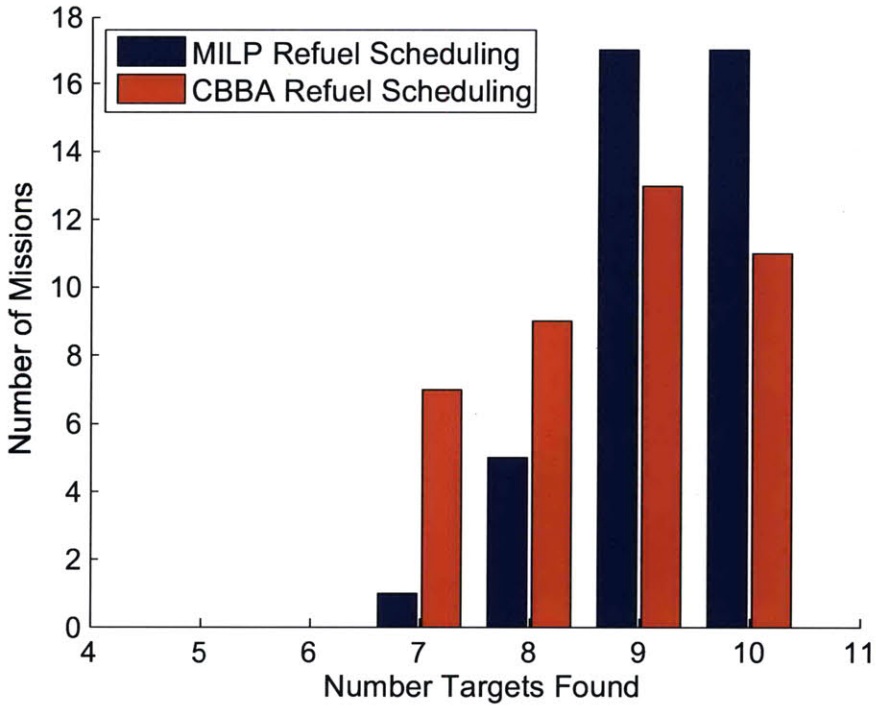

Figure 4-3: Total Number of Targets Found

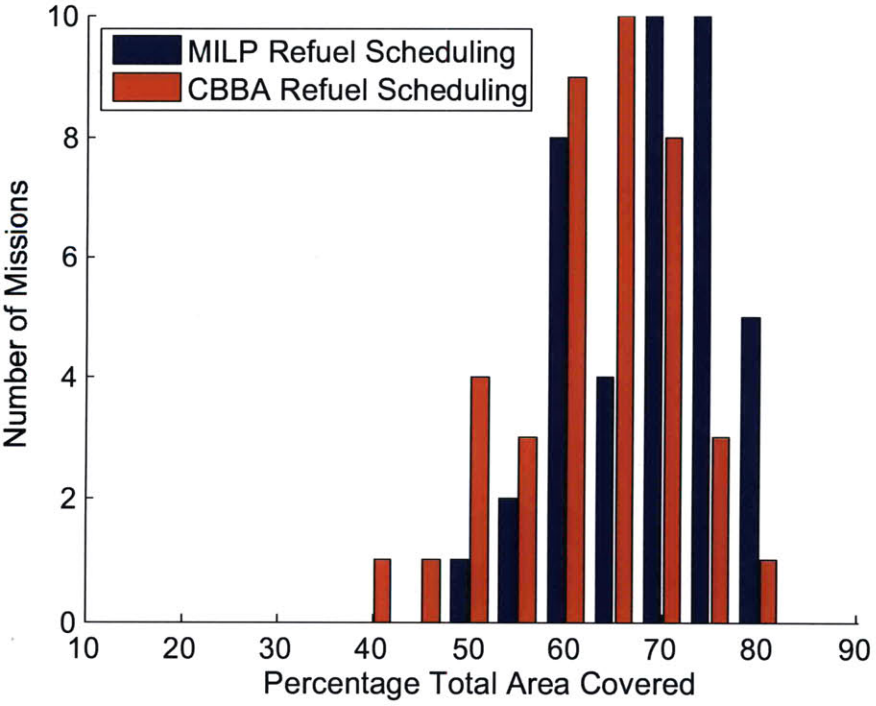

Figure 4-4: Percentage of Total Area Covered

Chapter 5

UAV Search Strategy Comparison

5.1 Introduction and Background

Current UAV operations are largely focused on Intelligence, Surveillance, and Reconnaissance (ISR). Critical in ISR type missions, is rapidly locating objects of interest in the environment. This chapter compares several strategies for a team of autonomous agents conducting search in a partially known environment. A baseline local search framework is extended to include global knowledge into the decision making process, and this extension is compared to several other search strategies. The strategies in the comparison include the original local search algorithm, local search plus a supervisory human model, global search protocol, global search plus human model, randomized search, and systematic search. Each of these strategies are described in detail in Section 5.2.

5.1.1 Problem Statement

Consider an environment containing a specific number of targets, N_{tgts}, where N_{tgts} is unknown a priori to the planning system. Also in this environment are N_u agents, which are deployed to search for the targets. The agents are autonomous, but possibly under supervisory control of a human operator. The mission also may be multi-objective. The mission may require that: targets are classified once found, certain

targets are revisited and tracked, hostile targets are engaged and neutralized, friendly targets are protected, etc. The focus of this study is search performance, and the study of other performance metrics are the topic of separate studies not presented here.

The metrics for success in a cooperative search mission are as follow:

- Number of targets found by end of mission- desired to be equal to the number of targets in the environment
- Time required to find each target- desired to be as small as possible for each target
- Percentage of search area covered by end of mission- desired to be 100 percent
- Average uncertainty of each cell during mission- desired to be as low as possible

For this study, the search algorithm performance is based on when each target is discovered, if ever. The objective function used for the comparison, can be written as

$$P_{\text{alg}} = \frac{\int_{t_0}^{t_f} f(t)dt}{(t_f - t_0)(N_{\text{tgts}})} \tag{5.1}$$

where $f(t)$ is the number of targets discovered on the time interval from t_0 to t. The upper bound of P_{alg}, the normalized performance of a given algorithm, is 1.0 which occurs if all targets are inside the agents' collective field of view at time, t_0. Performance is lowerbounded by 0.0, which occurs if the agents never find a target during the mission. (Note: P_{alg} would be 0.5 if the agents discovered all targets uniformly in time throughout the mission, or if the agents discovered half the targets at the beginning and never found anything else.)

Constraints in the problem include:

- Vehicle dynamics
  - Maximum speed
  - Minimum turning radius

- Refueling - vehicles must visit refueling base location before fuel is depleted
- Environment boundaries - vehicles must stay inside convex environment

5.2 Description of Algorithms

5.2.1 System Architecture

The simulation environment used in the search algorithm comparison experiment is the environment developed for the Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance (OPS-USERS) program. The OPS-USERS architecture is specifically designed to meet the challenges associated with an automated decision-making system with a human in-the-loop. Two key challenges are: 1) balancing the roles and responsibilities of the human operator and the automated planner, and 2) balancing resource allocation between searching for new targets, and tracking previously found targets. The system attempts to rely on the relative strengths of both humans, and smart machines. The basic system architecture is divided into two major components. The Distributed Tactical Planner is actually a network of Onboard Planning Modules (OPM) working together to achieve a common mission objective. The Ground Control Station consists of a Central Mission Manager (CMM), which is used as a rapid analysis tool, and the Human Interface (HI), by which the human operator interacts with the automated system. Figure 5-1 shows the OPS-USERS system architecture.

Onboard Planning Module

The Onboard Planning Module (OPM) is the heart of the OPS-USERS system. Each autonomous agent is equipped with an OPM, which is responsible for the high-level decision making for each agent. The OPM runs two major decision making processes: 1) The task planner is responsible for deciding which tasks each agent should perform and when they should plan to arrive at each of those tasks. 2) The search algorithm is responsible for planning a trajectory that maximizes information gain.

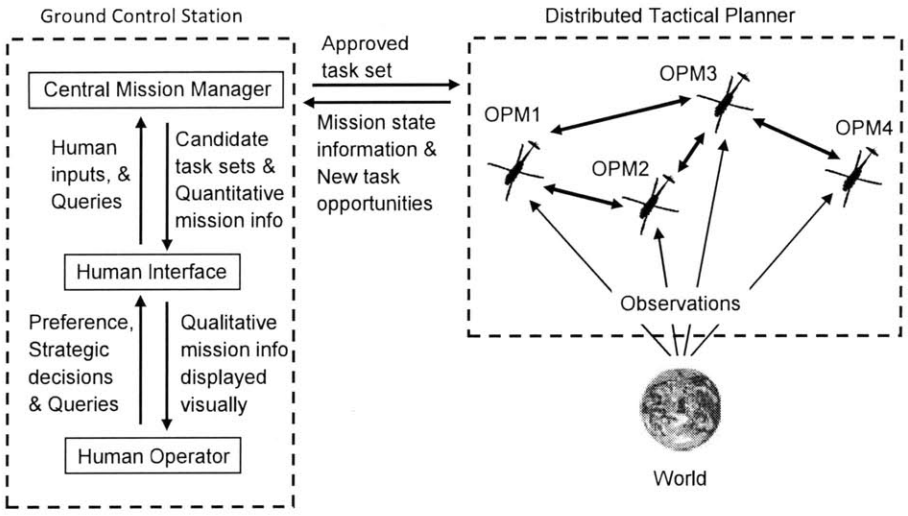

Figure 5-1: OPS-USERS high-level system architecture

Tasks may include a variety of activities, including tracking targets that are already found (track tasks), changing the location from which local search trajectories are generated (search tasks), engaging hostile targets, or refueling. Each of these tasks are given a *score* which represents an estimate of how important that particular task is to the overall mission. Task scores are typically a function of which agent performs the task, and the time at which the task is performed. The task planner used in OPS-USERS is the Consensus Based Bundle Algorithm (CBBA). See Chapter 2 for details.

In the OPS-USERS framework, search is a *spare time strategy*, that is, active search only takes place when a vehicle has no immediate task it needs to do. Spare time is defined as

$$\text{isSpareTime}(t) = \begin{cases} \text{true} & \text{if } (t_{\text{trans}}) < (t_{\text{nextTask}} - t) \\ \text{false} & \text{otherwise} \end{cases} \tag{5.2}$$

where t is the current time, t_{trans} is the time required to travel from the current location to the next task location along the shortest route, t_{nextTask} is the time the vehicle is scheduled to arrive at the next task in its plan.

Central Mission Manager and Human Interface

The Central Mission Manager (CMM) is a specialized planning module used between the human interface and the distributed tactical planner. In a mission, it is often infeasible to assign all tasks due to mission specific constraints. The CMM runs a local centralized version of the task planner to determine a feasible subset of tasks that the network of agents can carry out. The operator is then able to conduct rapid analysis on the feasibility of performing a specific group of tasks, and is able to modify the set of tasks under consideration. Every task set must be accepted by the operator before those tasks are released to the distributed tactical planner. This planning done at a more strategic level does not concern the operator with the mechanics of how the tasks will be carried out, but allows the operator to decide which tasks should be included in the plan.

The Human Interface (HI) conveys information to the operator such as agent positions, agent fuel state, target probability distribution, known target locations, etc. The HI also allows the operator to specify their preferences. The operator is able to specify points of interest through the HI. A location chosen by the operator becomes the location for a search task that is later given to the task planner if the operator accepts a plan that includes it. In this way, the operator can guide the search process if he or she chooses.

5.2.2 Local Search Algorithm

The *Local Search Algorithm* used during the search strategy comparison experiment is the search algorithm developed for the OPS-USERS program. The objective of the local search algorithm is to locate objects in the environment known as *search targets.* Search targets are fictitious entities used to guide the agents to regions in the world where there is high probability of discovering a physical target. Search targets can be initialized before the start of a mission with a priority level, environment type, and expected velocity, and multiple search targets may be instantiated at one time.

The local search algorithm is based on a set of cognitive maps stored in the

agents' memories. The map represents the world, and is discretized into cells. Each cell contains a vector of probabilities, where the entries of the vector represent the probability that a particular search target is present in that cell. When an agent has spare time, i.e. no tasks to do, the agent may perform search. The local search trajectory is generated based on the probability map. The agent builds up a search tree which is depth limited to a predefined planning horizon. The quality of each search path is based on which cells could be observed with the agent's sensor by following that path. The score for a given cell, i is calculated by

$$s_{\text{cell}}(i) = \sum_{k \in \text{Stgt}} \left(\text{priority}_k \cdot p_i(k)\right) \tag{5.3}$$

where priority_k represents the relative importance of search target k. $p_i(k)$ is the probability that cell i contains search target k. The score for a given path is calculated by

$$s_{\text{path}}(j) = \sum_{i \in I_j} \left(s_{\text{cell}}(i) \cdot \text{perc}_j(i)\right) - \text{pen}_{45} \cdot n_{j45} - \text{pen}_{90} \cdot n_{j90} \tag{5.4}$$

where I_j is the set of all cells observable on path j of the search tree. $\text{perc}_j(i)$ is the percentage of cell i that can be observed by following path j. pen_{45} and pen_{90} are time penalties associated with making 45 degree turns and 90 degree turns respectively, and n_{j45} and n_{j90} are the number of 45 and 90 degree turns in path j. The agent then selects the path that maximizes $s_{\text{path}}(j)$.

The search target probability distributions are updated in two ways. First, if an agent observes a cell or knows of another agent that observes a cell with no target, that cell's probability of containing a target is reduced. If perfect sensing is assumed, the probability in that cell is set to

$$p_i^+(k) = p_i^-(k) \cdot (1 - \text{perc}_j(i)) \tag{5.5}$$

where $p_i^+(k)$ is the probability that cell i contains search target k after the observation is made, $p_i^-(k)$ is the probability that cell i contains search target k before the observation is made, and $\text{perc}_j(i)$ is the percentage of cell i that was observed.

Since targets are assumed mobile, the search target probabilities also diffuse with time. The target model assumes that target motion is independent of agent activity. Targets are neither cooperative (attempt to minimize the time until they are found) or adversarial (try to maximize the time until they are found). The diffusion model is described by

$$p_i^{\text{new}}(k) = \min\left(p_i^{\text{old}}(k) + \sum_{j \in \mathcal{N}(i,k)} p_j^{\text{old}}(k) \cdot p_{\text{trans}}(j,i) \cdot \mathbb{I}(p_i^{\text{old}}(k) < p_j^{\text{old}}(k)), \min_{j \in \mathcal{N}(i,k)} (p_j^{\text{old}}(k))\right) \tag{5.6}$$

where $p_i^{\text{new}}(k)$ is the probability that search target k is in cell i after the propagation, $p_i^{\text{old}}(k)$ is the probability that search target k is in cell i before the propagation, $N(i,k)$ is the set of cells that neighbor cell i, and are not obstacles for the k^{th} search target environment type, $p_{\text{trans}}(j,i)$ is the probability that a target travels from cell j to cell i during one time step, and $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the argument is true, and 0 otherwise.

The propagation can be thought of as a "high-to-low" mechanism where probabilities only flow from a cell with higher probability to cells with lower probability.

Coordination in the search problem context refers to planning paths that produce maximum information gain collectively. This is typically accomplished by agents choosing paths that do not cover the same region in space. The coordination strategy in OPS-USERS is as follows:

Before the mission begins, each agent is given a priority level based on their id number. At each time step during the mission, each agent communicates its current trajectory to every other agent. If agent i has spare time, meaning the agent has more than enough time to get to its next task, it generates a search path. For each agent of higher priority $j < i$, agent i determines which cells agent j will observe along its planned trajectory, and these cells are added to a list inFOV. For each agent of lower priority $j > i$, agent i determines which cells agent j will observe only along the frozen trajectory (execution horizon), and these cells are also added to the list inFOV. The cells in inFOV are then considered already searched when the search tree is constructed.

5.2.3 Global Search Strategy

The purpose of developing a global search strategy is to improve search performance by increasing the effective planning horizon. The two key elements of this search protocol are

1. intelligently deciding whether an agent should plan a locally optimal trajectory from where they are (use local search algorithm), or move to a different location and plan their trajectory from there, and

2. deciding where an agent should begin their new locally planned trajectory should they choose to abandon their current location.

The global search algorithm selects task locations in a decentralized manner, and assigns scores to each task that represent how useful it would be to search the space around that task location. Each agent also keeps an estimate of the value of traversing a locally planned search trajectory. The critical criterion for abandoning the locally planned search trajectory in favor of traveling to a new task location instead, is sufficient information gained at the task location under consideration. Therefore, agents are only eligible for tasks that have a higher score than their current threshold.

The metric that quantifies the quality of the information gained by a local search trajectory is

$$T_{\text{localSearch}} = \|\mathcal{P}_{\text{opt}}\|_{\infty} \tag{5.7}$$

where $T_{localSearch}$ is the local search threshold, and $\mathcal{P}_{opt}$ is the set of scores associated with each of the cells visible along the trajectory planned by the local search algorithm. The cell scores are calculated based on equation 5.3. The threshold represents the score of the most valuable cell observable along the locally optimal search path.

The second key element of the global search strategy involves selecting candidate locations to search. For an environment that is discretized into N_{cells} cells, enumerating each cell as a candidate search task location may be computationally prohibitive for large N_{cells}. Instead, the environment is partitioned into *sampling zones*. The number of sampling zones, N_{sz} is a mission parameter, and is chosen before the mis-

sion begins based on the size of the environment and the available computational capability. Each sampling zone is comprised of N_{scan} contiguous cells, where typically $N_{\text{scan}} \approx N_{\text{cells}}/N_{sz}$. During a mission, sample zones are compared to determine which zones should contain search tasks, and only one search task may exist per zone. The value associated with a search task in zone z is computed

$$v(z) = \frac{\sum_{i \in z} (s_{\text{cell}}(i))}{N_{\text{scan}}} \tag{5.8}$$

where $s_{\text{cell}}(i)$, the score for cell i, is computed just as it was in the local search strategy, except there is an added weighting based on the time since the cell was last visited. Equation 5.3 is modified to become

$$s_{\text{cell}}(i) = \sum_{k \in \text{Stgt}} (\text{priority}_k \cdot p_i(k)) \cdot \left(1 + \frac{t - t_{\text{lastVisit}}(i)}{E[t_f - t_0]}\right) \tag{5.9}$$

where t is the current time, $t_{\text{lastVisit}}(i)$ is the time cell i was last visited, and $E[t_f - t_0]$ is the expected mission time. The additional time weighting favors visiting cells which have not been visited recently over cells that have. Note: At the start of a mission, $t_{\text{lastVisit}}$ may be initialized to a large negative number, $t_{\text{lastVisit}} \ll -E[t_f - t_0]$, which bias the search toward cells that have never been visited.

The process of selecting desirable locations to search is distributed across the network of agents. The responsibility partitioning at a given time step is based on a Voronoi diagram. The diagram is constructed using the agent locations as the Voronoi sites. Agent i is responsible of posting possible search tasks in zone z according to

$$z \in z_i \text{ iff } d_{iz} < d_{jz} \ \forall j \in \mathcal{I}, j \neq i \tag{5.10}$$

where z_i is the set of zones for which agent i is responsible, d_{iz} is the distance from agent i to the centroid of zone z, d_{jz} is the distance from agent j to the centroid of zone z, and $\mathcal{I}$ is the set of all agents in the network.

In figure 5-2, $N_{\text{cells}} = 108$, $N_{sz} = 12$, $N_{\text{scan}} = 9$, and there are 5 agents. Using rule 5.10, the Table 5.1 is generated.

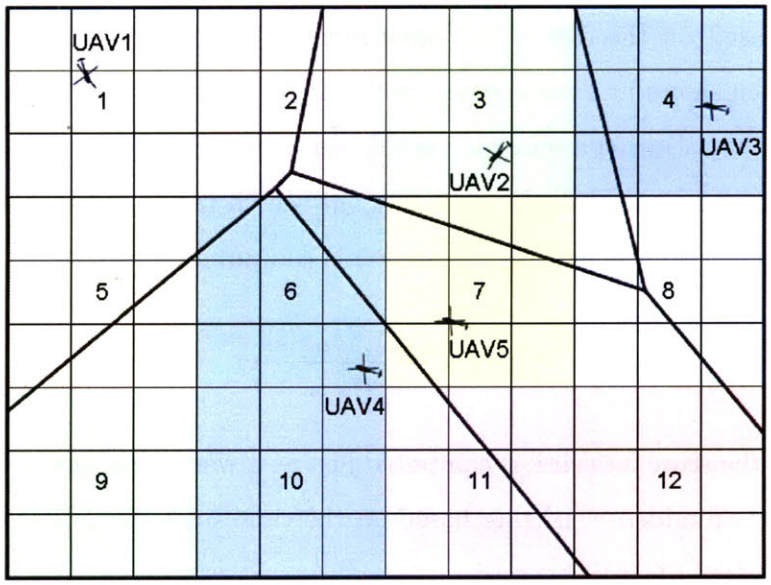

Figure 5-2: Voronoi regions for agents searching an environment

Table 5.1: Corresponding zones for a five agent fleet

Agent	Zones responsible for
UAV 1	1,2,5
UAV 2	3
UAV 3	4,8
UAV 4	6,9,10,11
UAV 5	7,12

At each iteration of the search task generator, each agent computes the mean and standard deviation of the sampling zone values

$$\mu_Z = \frac{\sum_{z \in Z} (v(z))}{N_{sz}} \tag{5.11}$$

where μ_Z is the arithmetic mean of the values of the sampling zones, and Z is the set of all sampling zones, and

$$\sigma_Z = \sqrt{\frac{1}{N_{sz} - 1} \sum_{z \in Z} (v(z) - \mu_Z)^2} \tag{5.12}$$

where σ_Z is the standard deviation in the values of the sampling zones.

After agent i has determined the elements in z_i and computed μ_Z and σ_Z, it selects zones that satisfy

$$v(z) > \mu_Z + \alpha_\sigma \cdot \sigma_Z \tag{5.13}$$

where α_σ is a tuning parameter which sets the sensitivity of the task generation protocol. If the condition is satisfied for zone z, and no task exists in zone z, then a task is created. If a task already exists in zone z, its value is updated.

The decentralized strategy requires that each agent carry a position estimate for every other agent in the network. Agents must therefore periodically communicate their positions to each other, or be informed by an outside source, i.e. satellite. The procedure for global search is shown in algorithm 6.

5.2.4 Human Operator Model

For the purpose of comparison, it is desired to conduct mission simulations both with and without the human operator. Human subjects were unavailable for this study, so a simple model of how operators interact with the planning system was constructed, with an emphasis on the search responsibilities of an operator. The model is based on data collected during a short-duration, high-workload human experiment which was developed and run by the Humans and Automation Laboratory (HAL) at MIT. Key elements of the human operator model are: frequency of new search task generation (how often), and search task location selection (where).

Data regarding the mission times at which operators selected search tasks was analyzed to find an accurate distribution reflecting the time between search task location selection. In figure 5-3, a normalized histogram of the time between new search task instantiation is plotted for a human supervising an OPS-USERS mission. The distribution is approximately log-normal with parameters mean, $\mu = 3.32$, and variance, $\sigma^2 = 0.706$. To recreate a realistic frequency of task generation in the human model, the actual time between search task generation is draw from a log-normal probability distribution. For example: If a new task location is chosen by the human model at time t, the next time at which a new location will be chosen is $t + dt$

Algorithm 6 Global Search Algorithm

procedure Main()
  for $t = [1 : missionTime]$ **do**
    if t mod $planRate_{taskGen} == 0$ **then**
      $z_{me} \leftarrow calculateVoronoi(agents, Z)$;
      $postNewTasks(z_{me}, Z, taskSet)$
    end if
    if $hasSpareTime == true$ **then**
      $generateLocalSearch(gridWorld)$;
    else
      $execute\ tasks_{me}.pop()$;
    end if
  end for

procedure CalculateVoronoi(agents, Z)
  for all $z \in Z$ **do**
    $foundCloser \leftarrow false$;
    for all $a \in agents, a \neq me$ **do**
      $distThem = \sqrt{(a.x - z.x)^2 + (a.y - z.y)^2}$;
      $distMe = \sqrt{(me.x - z.x)^2 + (me.y - z.y)^2}$;
      if $distThem < distMe$ **then**
        $foundCloser \leftarrow true$;
        break;
      end if
    end for
    if $\neg foundCloser$ **then**
      $z_{me}.add(z)$;
    end if
  end for
  return z_{me};

Procedure postNewTasks(z_{me}, Z, taskSet)
  $\mu_Z \leftarrow calcMean(Z)$;
  $\sigma_Z \leftarrow calcStdDev(Z)$;
  for all $z \in z_{me}$ **do**
    $value \leftarrow evaluateValue(z)$;
    if $value > \mu_Z + \alpha_\sigma \cdot \sigma_Z$ **then**
      $cell_{best} \leftarrow arg\max_{c \in z.cells} s_{cell}(c)$;
      $task_x \leftarrow cell_{best}.x$;
      $task_y \leftarrow cell_{best}.y$;
      if $\exists\ task_z \in taskSet$ **then**
        $updateTask(task_x, task_y, value)$;
      else
        $postTask(task_x, task_y, value)$;
      end if
    end if
  end for

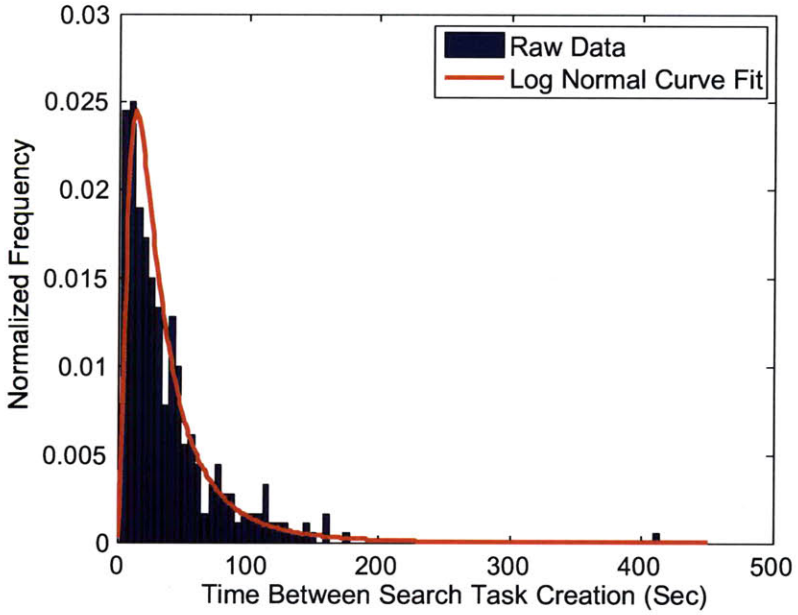

Figure 5-3: Distribution of time between search task creation for human operator

where dt is a random variable pulled from the probability distribution shown by the red line in figure 5-3.

The modeling of where human operators select search tasks is based on a qualitative analysis by the HAL team members who oversaw the short-duration, high-workload study. The human interface visually displays a cognitive map to the operator, where cells with high score are shown with a purple shading. The members reported that operators tended to select task locations in regions with large amounts of shading, or high-score cells. The members also reported that operators in the study tended to select search task location far from the agents, such that the agents would go to locations where the local search algorithm would not necessarily take them. Based on these general rules, task locations are chosen based on the following procedure:

1. Select map quadrant with largest number of cells that have higher-than-average probability of containing a search target. Add all cells in quadrant to $\text{Sample}_{\text{human}}$

2. Subtract any cell within r_{disk} of nearest agent from $\text{Sample}_{\text{human}}$

3. With uniform probability, randomly draw cell from $\text{Sample}_{\text{human}}$

5.3 Simulation Parameters and Environment Description

A total of six search strategies are compared in this experiment. The strategies in the comparison are

1. Local search with no operator

2. Local search with human operator model

3. Global search with no operator

4. Global search with human operator model

5. Systematic search

6. Random walk

Strategies 5 and 6 are benchmarks used for comparison. The systematic search implemented for this experiment is a "lawn mower" search path. The world is divided into N_u rectangles, and each agent traverses its own rectangle by sweeping back and forth in a similar fashion as someone mowing a lawn. When an agent needs to refuel, they leave their sweeping path to do so, and return to the same spot and resume.

The random walk is implemented as an approximation to Brownian motion. Each time through the planning loop, a cell is chosen at random from the 8 cells that surround the agent's current cell. The randomly chosen cell becomes the agent's next waypoint.

Monte Carlo simulation are run to compare the performance of the six strategies. A total of 50 missions are simulated for each of the six strategies, and initial conditions are randomly generated for each of the 50 missions. The mission objective is to find

Table 5.2: Simulation parameters, environment

Parameter	Value
Dimensions	12.5 x 7.5 km
Cell size	150 x 150 m
Number of sampling zones, N_{sz}	70
Cells per zone, N_{scan}	81
Base location	(0,0) (Environment center)
Mission duration	1000 seconds

Table 5.3: Simulation parameters, agents

Parameter	Value
Number of agents, N_u	3
Environment type	Air
Cruise speed	100 m/s
Cruise altitude	1100 m
Sensor type	Camera
Sensor footprint	910 x 680 m, rectangle
Initial position	Random uniform on 2000 m square around base
Maximum time between refuel	400 sec
Initial fuel	Random uniform on [200,400] sec

Table 5.4: Simulation parameters, targets

Parameter	Value
Number of targets, N_{tgts}	10
Environment type	Ground
Cruise speed	Random uniform on [3,5] m/s
Initial position	Random uniform over ground cells

as many targets as possible as early in the mission as possible. In this mission, once a target is found, it is not required to be tracked, since this study is entirely scored on search performance. Important mission parameters are listed in the following tables. Table 5.2 lists parameters associated with the mission and simulation environment. Table 5.3 lists parameters associated with the autonomous agents. Table 5.4 lists parameters associated with the targets of interest.

Two studies are conducted with differing initial information. In study 1, the planner is informed that all targets are ground vehicles, so the a priori probability

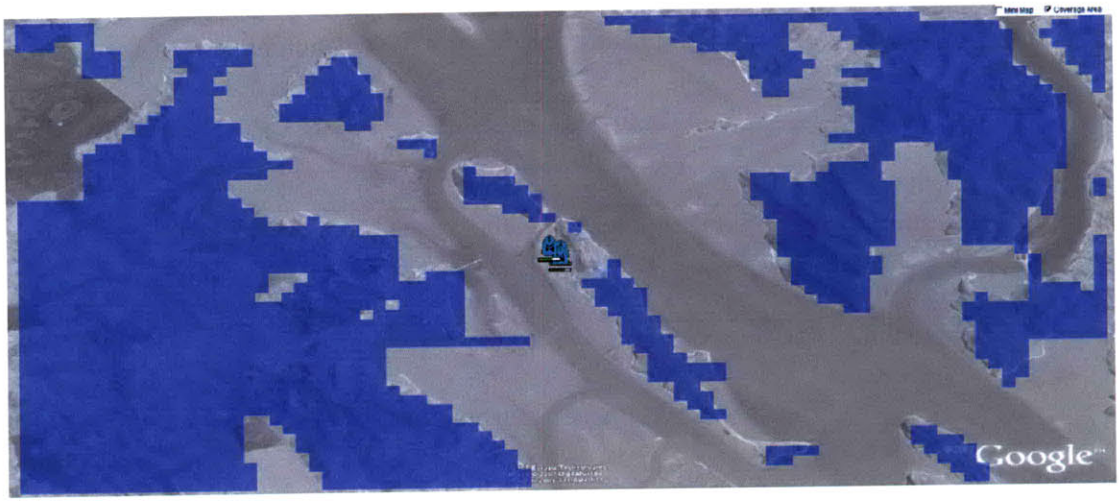

Figure 5-4: A priori probability distribution for informed initial conditions study

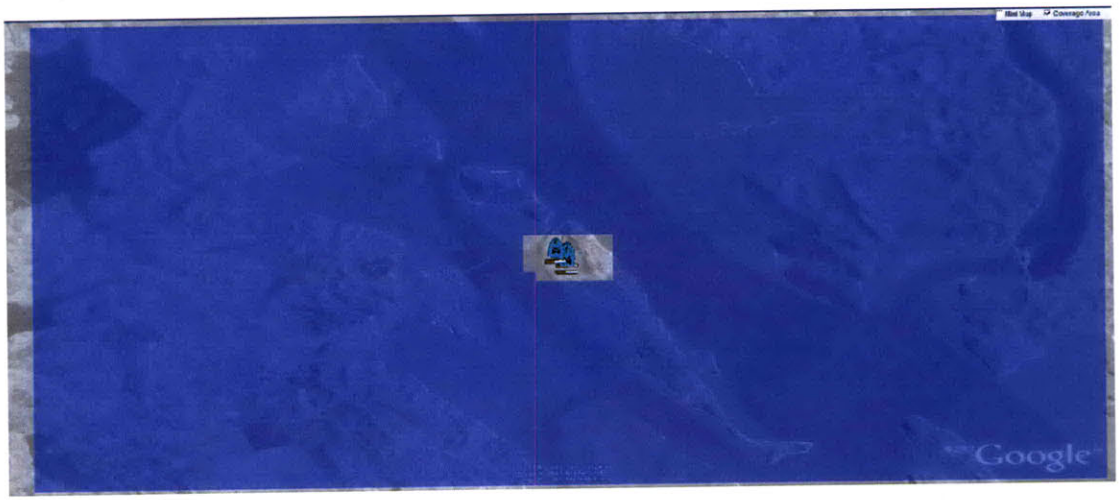

Figure 5-5: A priori probability distribution for uninformed initial conditions study

distribution is uniform over only the ground cells. Study 1 is referred to as the *informed initial conditions* study, and the initial probability distribution is shown in 5-4. In study 2, the planner is not informed that all targets are ground vehicles, so the a priori probability distribution is uniform over all cells. Study 2 is referred to as the *uninformed initial conditions* study, and the initial probability distribution is shown in 5-5.

5.4 Numerical Comparison of Performance

The simulation performance results are presented here. The results from the informed initial conditions study are presented first. Figure 5-6 shows the number of targets found as a function of mission time, averaged over the 50 trials, and Figure 5-7 presents a histogram of the number of targets found by the end of the mission. All error bars in this section show the 95 percent confidence interval for the corresponding value.

The random walk search performs extremely poorly compared to the other search strategies studied. The random walk is included in the study not because it is expected to perform well, but because its performance gives insight into the difficulty of the problem. In this scenario, the targets are sufficiently difficult to find that a random walk search discovers fewer than one target on average by the end of the mission. Furthermore, the random walk never found more than three targets for the 50 missions simulated, as seen in Figure 5-7. The lawn mower search performs much better than the random walk, but is still significantly worse than the four optimized search strategies in the study. Notice in Figure 5-6 that both the random walk and the lawn mower search seem to find targets at a linear rate with respect to time, whereas the other four strategies are linear during about the first quarter of the mission, but then gradually level out. The difference is because the optimized search strategies are always attempting to search high probability regions first, whereas the lawn mower pattern covers the environment in a predefined fashion.

Figure 5-7 shows that both the global search strategy and the lawn mower search strategy found all 10 targets for 49 of the 50 missions, and always found at least 9 targets by the end of the mission. This indicates that these two techniques are very reliable in terms of the number of agents found by the end of the mission. However, the global search strategy performs much better in terms of minimizing the time required to find each target as seen from Figure 5-7.

Of the four optimized search strategies, the global search strategy performs the best as shown in Figure 5-6. The curve showing the number of targets found as a

function of mission time for the global search algorithm lies near or above the curves for all of the other strategies presented, and the error bars show that the difference is statistically significant. The local search strategy alone performs the worst of the four optimized search strategies, and the remaining two cases, local plus human model, and global plus human model, fall in between and are not statistically different.

The study demonstrates the benefit of extending the effective planning horizon through either a centralized human operator, or a global search protocol, and in this case, the global search protocol is more effective than the human operator model. However, the study supports that mixing the roles of the human operator with what the automated planner is trying to accomplish is not necessarily beneficial: the helpful benefits of each do not simple sum together. In this case, both the human operator and the global search task generator are selecting locations to seek out, and cause interference with each other. As a result, the global search plus human strategy performs worse than the global search strategy alone.

The results from the uninformed initial conditions study are presented next. Figure 5-8 shows the number of targets found as a function of mission time, averaged over the 50 trials for the uninformed initial conditions study. Figure 5-9 presents a histogram of the number of targets found by the end of the mission. Finally, Figure 5-10 illustrates the relative performance of each algorithm averaged over 50 trials, comparing both informed and uninformed initial condition cases. The score that is depicted is described in equation 5.1, and it represents how quickly each of the targets are found on average. The scores in Figure 5-10 are calculated by integrating each of the curves in Figures 5-6 and 5-8.

The results of the uninformed initial conditions study are very similar in nature to the results from the informed initial conditions study. The main difference here is that the performance difference is smaller between all of the strategies considered. The random walk, and the lawn mower search pattern performance are independent of the initial conditions of the probability map, because they do not use it for path planning. However, each of the optimized search strategies suffer a performance loss for uninformed initial conditions compared to informed initial conditions as shown

in Figure 5-10. The performance difference for the optimized search strategies exists because they each use the probability map to guide the search. Therefore, being informed that a certain group of cells in the environment has a zero probability of containing a target is very useful for such algorithms. With this information, they can avoid wasting time searching needlessly.

Overall, the two studies confirm that the global search strategy is an effective method for extending the planning horizon of the local search algorithm, and results in a greater number of targets found during a mission, as well as reduces the amount of time required to find a given number of targets. The studies also show that a human can be effective in providing useful information to an automated system, but the integration must be done with care to avoid conflicting decisions when there is overlap in the responsibilities [45].

5.5 Summary

This chapter presents a baseline local search strategy which is a receding horizon optimization over a dynamic probability map. A global search strategy is developed which extends the capabilities of the local search algorithm by examining the entire probability map, and generating search tasks in a decentralized manner. The strategies are compared both with a human operator model, and without a human operator model, and the simulation results demonstrate the that global search algorithm without the human model performs the best in terms of number of targets found.

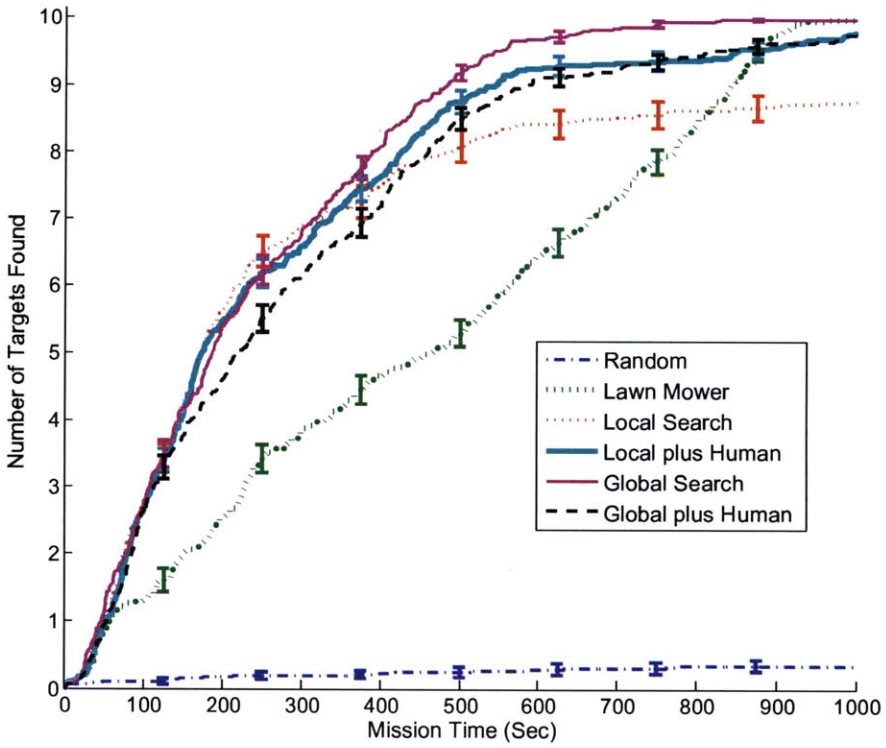

Figure 5-6: Average number of targets found as a function of time, informed

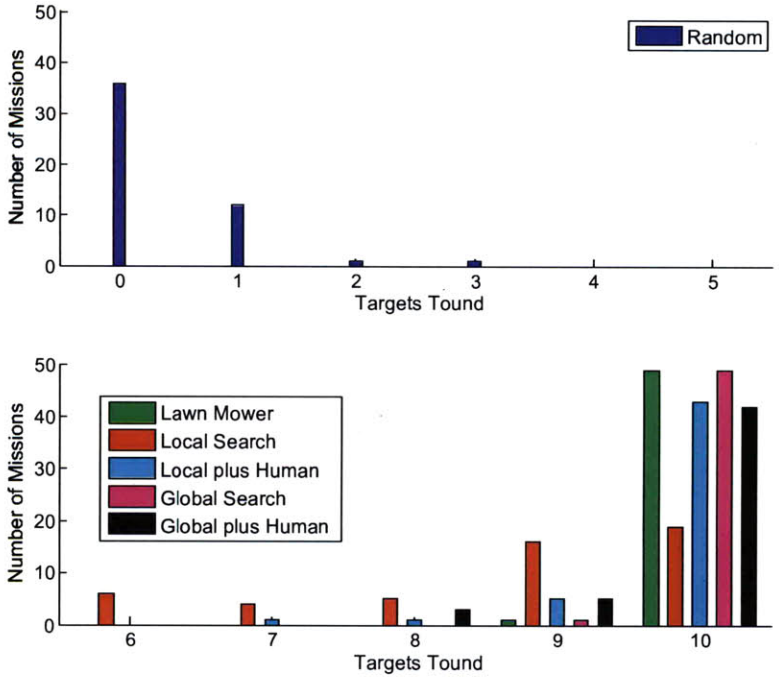

Figure 5-7: Histogram of total number of targets found, informed

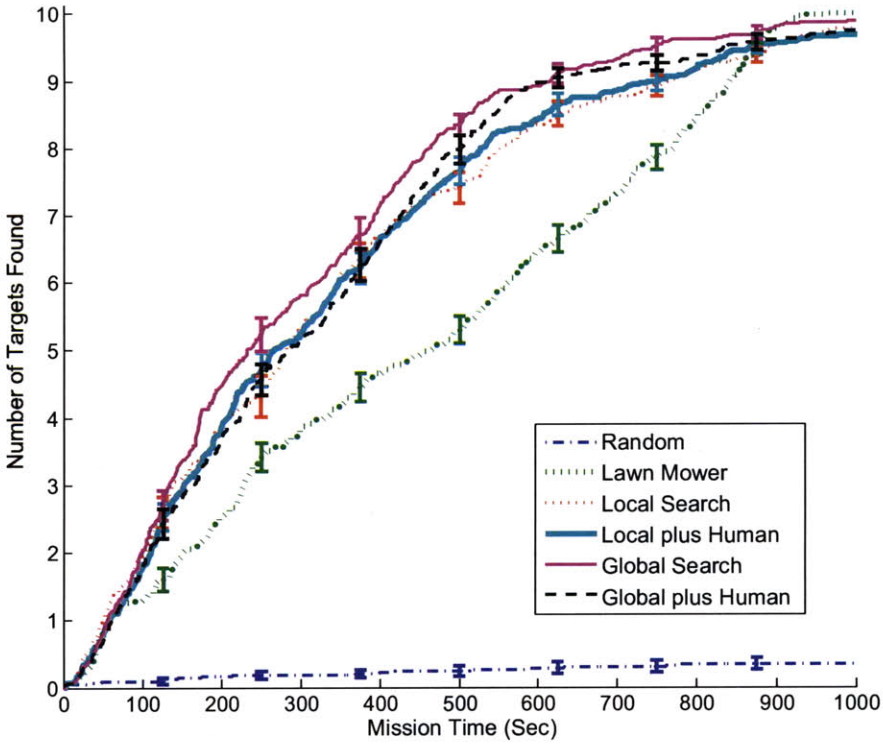

Figure 5-8: Average number of targets found as a function of time, uninformed

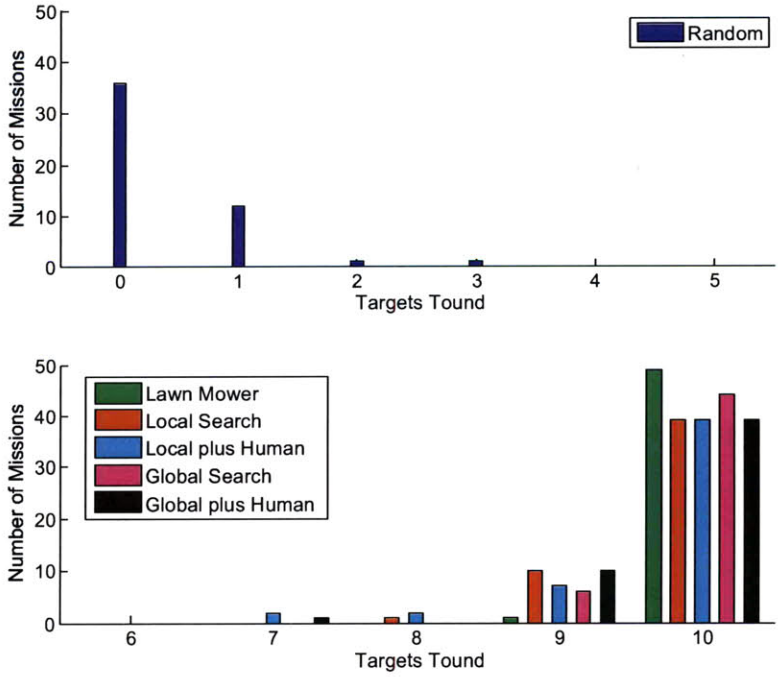

Figure 5-9: Histogram of total number of targets found, uninformed

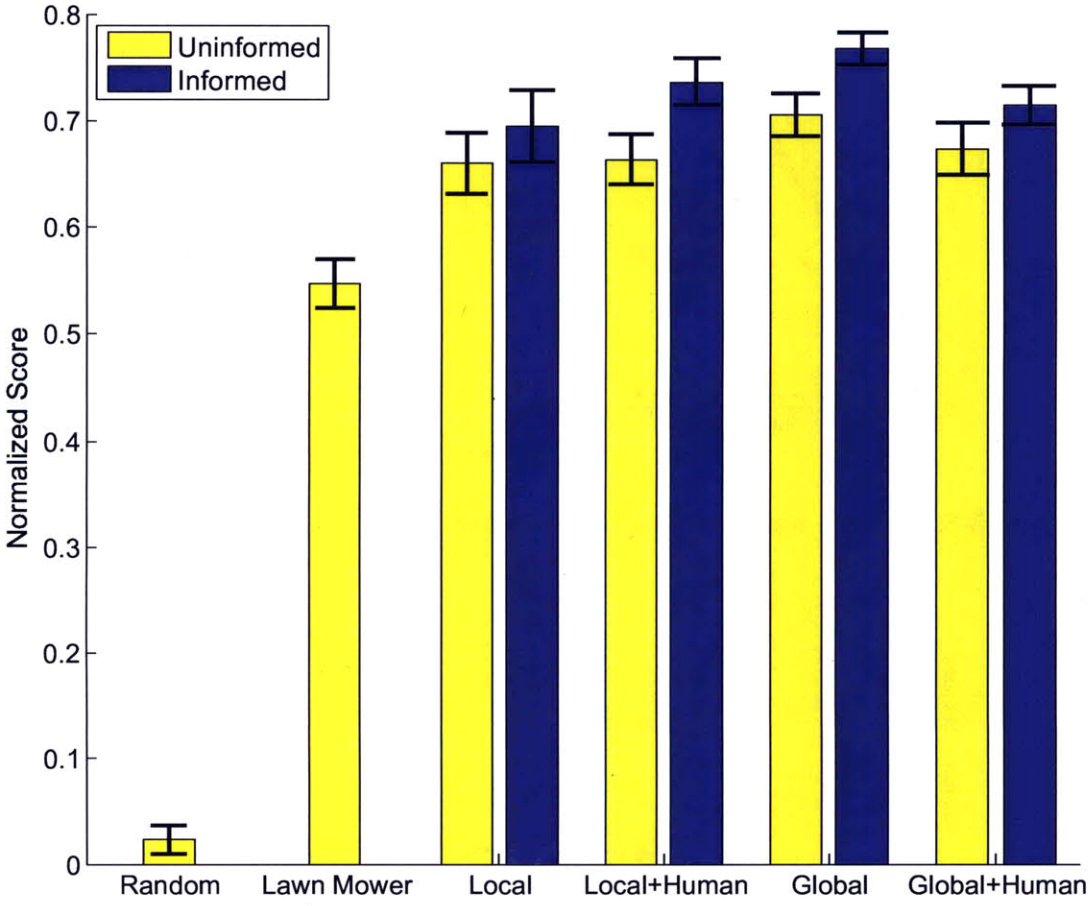

Figure 5-10: Performance score for each strategy tested

Chapter 6

Conclusions

The objectives of this thesis are to: 1) develop a decentralized task planning method that is capable of handling coupled constraints, and 2) develop a decentralized search strategy that improves upon existing approaches, and mitigates some of the effects of myopic trajectory generation. Both of these objectives were accomplished in this thesis. The Coupled-Constraint CBBA is introduced in Chapter 3, and a global search protocol is discussed in Chapter 5.

6.1 Thesis Summary

Chapter 1 of this thesis introduces the two thesis objectives, and discusses some of the previous work in the area of mission planning for autonomous agents. There currently exist methods for satisfying general coupled constraints, but most of these techniques are centralized; a decentralized approach is desirable to overcome some of the communication limitations and robustness issues common in centralized planning. Current autonomous search frameworks often utilized cognitive maps for trajectory generation, and plan trajectories according to a receding horizon optimization. The approach is tractable for real-time path planning, but can suffer in performance due to near-sightedness. Therefore, a method of extending the planning horizon is desirable.

Chapter 2 explains the baseline task assignment algorithm. CBBA comprises two phases. In this first phase, task selection, agents build a *bundle* by sequentially select-

ing tasks which result in the greatest score increase for their bundle. In the second phase, consensus, agents communicate with neighboring agents to resolve conflicts in the assignment. CBBA is guaranteed to converge to a conflict-free assignment that is at least 50 percent of optimal if the network if strongly connected, and the score structures satisfies a property call diminishing margin gain (DMG). CBBA with Time Windows is also introduced in this chapter, which enables the algorithm to produce assignments for time-sensitive tasks. CBBA with Time Windows is shown to satisfy sufficient conditions to guarantee convergence.

Chapter 3 presents an algorithm extension to CBBA for handling coupled constraints in the task set. The additional constraints handled by the new framework are unilateral dependency constraints, mutual dependency constraints, mutual exclusions, and temporal constraints. Coupled-Constraint CBBA (CCBBA) is introduced as an efficient mechanism for solving the constrained task assignment problem.

The task set is partitioned into *activities*, which are groups of tasks that share coupled constraints. Two bidding strategies are developed. If a task is not mutually dependent on any other task, it is given a pessimistic bidding strategy: agents wait to bid on the task until all of its dependency constraints are satisfied. If a task is mutually dependent on at least one other task, it is given an optimistic bidding strategy: agents may bid on a task even if all dependency constraints are not satisfied, as long as they have not exhausted their limited attempts on that task. An agents is also required to release a task if it has been winning the task for too long without the required number of satisfied constraints.

The quality of assignments for CCBBA are compared to CBBA in numerical simulations. CCBBA is shown to produce superior assignments compared to CBBA for tasks sets with coupling. The results demonstrate that enforcing coupled constraints must be handled explicitly in a complex mission to achieve maximum performance; however, some additional computation and communication is required.

Chapter 4 presents a method for solving the task assignment problem under refuel constraints. The new method is compared to an existing approach, which requires solving a Mixed Integer Linear Program at every iteration. Numerical results confirm

that the new strategy outperforms the original strategy in the quality of assignments. Furthermore, the computational complexity is greatly reduced enabling scalability to larger problems.

Finally, Chapter 5 presents a baseline local search strategy, which is a receding horizon optimization over a dynamic probability map. A global search strategy is developed which extends the capabilities of the local search algorithm by examining the entire probability map, and generating search tasks in a decentralized manner. The strategies are compared both with a human operator model, and without a human operator model, and the simulation results demonstrate the that global search algorithm without the human model performs the best in terms of number of targets found.

6.2 Future Work

Coupled-Constraint CBBA can benefit from addition research in several areas. First, the convergence properties of CCBBA should be thoroughly studied to understand the algorithm's behavior. A sufficient condition to guarantee convergence, analogous to DMG for CBBA, would be useful. Also, the effects of the algorithm parameters should be studied. An experiment should be run to determine how the choice the timeout variables affect performance and runtime. An additional experiment should be run to study the effects of the initial values for $\mathbf{w}_i^{\text{solo}}$ and $\mathbf{w}_i^{\text{any}}$. Both experiments should be run across a wide variety of network topologies to understand key features and limitations of the algorithm.

Autonomous search can benefit from additional research as well. In particular, the problem of autonomous search with a human-in-the-loop should become better understood. A large-scale study should be conducted on how to best structure the roles and responsibilities between the human operator and the planning system for autonomous agents searching an enviroment.

Appendix A

Consensus Table for CBBA

Table A.1: CBBA Action rules for agent i based on communication with agent k regarding task j

Agent k (sender) thinks z_{kj} is	**Agent i (receiver) thinks z_{ij} is**	**Receiver's Action (default: leave)**
k	i	if $y_{kj} > y_{ij} \rightarrow$ update
	k	update
	$m \notin \{i, k\}$	if $s_{km} > s_{im}$ or $y_{kj} > y_{ij} \rightarrow$ update
	none	update
i	i	leave
	k	reset
	$m \notin \{i, k\}$	if $s_{km} > s_{im} \rightarrow$ reset
	none	leave
$m \notin \{i, k\}$	i	if $s_{km} > s_{im}$ and $y_{kj} > y_{ij} \rightarrow$ update
	k	if $s_{km} > s_{im} \rightarrow$ update else $\rightarrow$ reset
	m	$s_{km} > s_{im} \rightarrow$ update
	$n \notin \{i, k, m\}$	if $s_{km} > s_{im}$ and $s_{kn} > s_{in} \rightarrow$ update if $s_{km} > s_{im}$ and $y_{kj} > y_{ij} \rightarrow$ update if $s_{kn} > s_{in}$ and $s_{im} > s_{km} \rightarrow$ reset
	none	if $s_{km} > s_{im} \rightarrow$ update
none	i	leave
	k	update
	$m \notin \{i, k\}$	if $s_{km} > s_{im} \rightarrow$ update
	none	leave

Appendix B

OPS-USERS Demonstration and Results

To demonstrate the technologies developed during the OPS-USERS program, the planning system was implemented with hardware-in-the-loop into MIT's Real-Time Indoor Autonomous Vehicle Test Environment (RAVEN), at the Aerospace Controls Laboratory [46–49]. Several mission scenarios were tested, and data was collected to characterize the performance of each mission.

B.1 Description of Testbed

RAVEN features a 780 square foot testing space. In this space, a Vicon motion capture system is used to measure the state of each vehicle in the environment in real time. The room is suitable for experiments requiring a large space, and involving multiple vehicles. Figure B-1 shows a photo of the flying space.

B.1.1 Motion Capture System

The Vicon Motion Capture System uses a constellation of 12 Vicon cameras. Each Vicon camera has several components in addition to the optical camera itself. The strobe is an array of Light Emitting Diodes (LEDs) in the near-infrared portion of

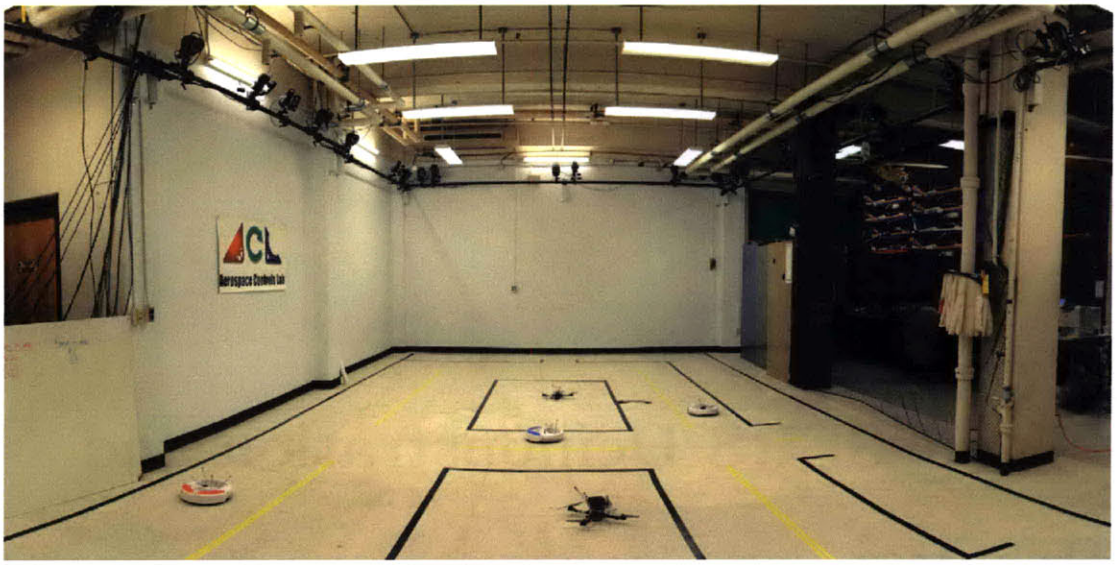

Figure B-1: RAVEN Flying Space

the electro-magnetic spectrum. It emits a cone of light in a predefined direction, which is absorbed, reflected, and scattered by objects in the environment. However, the environment also contains special reflective spheres, or markers which strongly reflect the light directly back to its source; these markers are assumed to be the most reflective objects in the environment. To measure marker positions, each camera has an optical sensor which detects portions of the captured image with large return of light. Also, each camera is equipped with a processing unit for calculating the centroid of each detected marker. Before an experiment, the system is calibrated to record the position and orientation of each of the cameras with respect to the user-defined origin. The system uses a right-handed Cartesian coordinate system. After calibration, the system stores x, y, z positions, as well as the direction cosine matrix for orientation for each camera.

To each vehicle to be tracked, a unique pattern of markers is affixed. The vehicles' marker patterns are entered into the motion capture system computer, and stored in memory. Once stored, the system can uniquely recognize each vehicle in the environment. During operation, the constellation of cameras simultaneously measure the position of the markers in their field of view. Each camera is able to measure marker position in two dimensions. At runtime, each camera streams the 2-D information

to a central processor, which compiles the data from all of the cameras. The data is processed in real time to determine the position of each marker in three-space. The system then matches the marker patterns in the environment to the marker patterns stored in memory, and reconstructs the position and orientation of each vehicle. RAVEN Data Stream (RDS) is an estimation process that runs in parallel to the motion capture system. RDS uses a Kalman filter to generate filtered state information for each vehicle. The Kalman filter uses a simple double integrator dynamics model, and uses Vicon data as the measurement vector. State information available from RDS includes position, velocity, orientation, and body rates, and is available at speeds up to 100 frames per second.

B.1.2 The RAVEN Vehicles

Two types of vehicles were used in RAVEN during the OPS-USERS demonstrations. Quadrotor helicopters (see Figure B-2) were used as the UxV autonomous agents. Quadrotors are ideal indoor test vehicles because they have hover capability, they are relatively small and light-weight, and they can carry a small payload, such as a camera. The flight code used to control each quadrotor is a quaternion based controller developed and implemented by ACL. The flight code runs on external computers which are linked to the vehicles via a wireless modem.

ACL also owns a fleet of ground vehicles. The Ground Platform for Unmanned Cooperative Control (GPUCC) vehicles served as the targets during the OPS-USERS tests. GPUCCs are modified Creates developed by iRobot. During missions, these vehicles followed predefined trajectories, which were unknown to the agents at the start of the mission. Each target has a different color marker on the top which allows it to be uniquely distinguished by both the unmanned agents and the human operator. Figure B-3 shows a GPUCC in the RAVEN test bed.

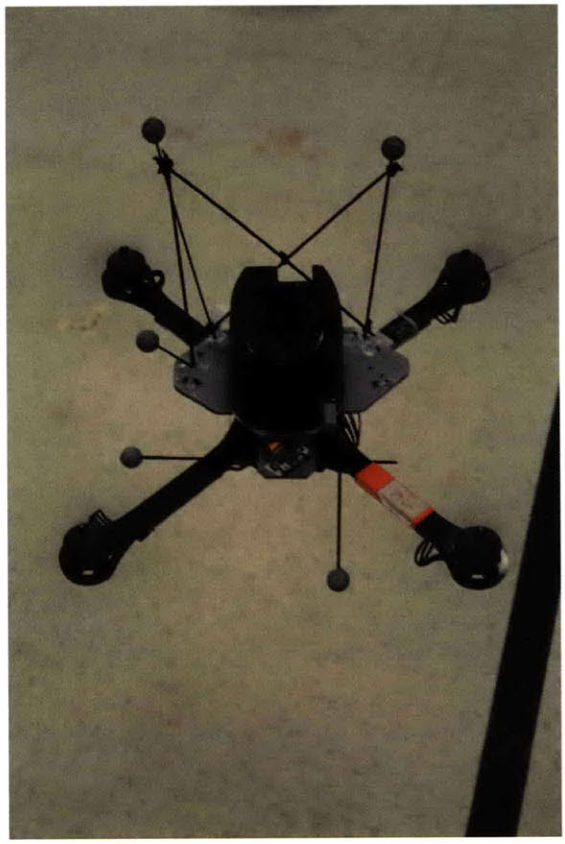

Figure B-2: Quadrotor Helicopter Equipped with Wireless Camera

B.2 OPS-USERS Demonstration

A variety of missions were flown in RAVEN spanning several weeks. Missions were flown with three autonomous agents, five targets, and one human operator. Parameters for a typical mission are described as follows: Two of the quadrotors were designated Sensor UAVs, and one was designated a Weaponized UAV (WUAV). The maximum speed of each vehicle was artificially limit to scale the environment appropriately. See Table B.1 for additional information on agent parameters. A total of five targets were in the environment during a mission. Two targets were hostile, two targets were friendly, and one target was unidentifiable. One of the five targets was a pop-up target and could not be detected before mission time 725.

Three refueling bases were created in the environment, one for each UxV. At the beginning of the mission, each UxV began at its own refueling base. After takeoff,

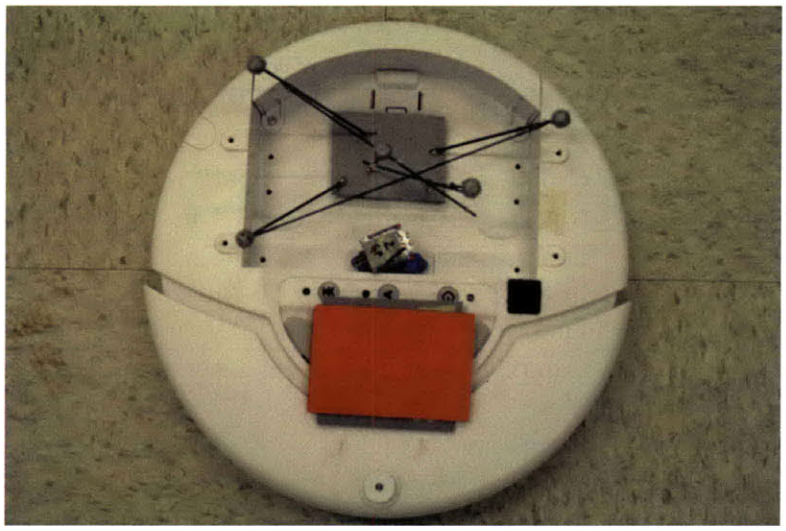

Figure B-3: Ground Platform for Unmanned Cooperative Control (GPUCC)

each UAV maintained a constant altitude, and followed the waypoints generated by the planner. The target trajectories were specified on the environment map before the mission, but that information was never revealed to the planning system. Figure B-4 shows the position of each of the refueling bases, as well as the target trajectories.

B.2.1 Mission Example

In this subsection, an example mission is shown, and key features of the OPS-USERS system are highlighted. At the beginning of the mission, the operator initiates the takeoff sequence for each of the UAVs. The UAVs then take-off automatically, and hover above their base until the mission is started. The operator has the option of selecting search tasks before the start of the mission. In Figure B-5, there is a prior probability distribution for target locations, so the operator makes the strategic high-level decision to create search tasks at the regions of high probability. If no such prior distribution is available, the operator may choose to place search tasks systematically, or per advice from the command center, or not at all.

As the mission gets under way, the planner schedules tasks for the agents, and they are carried out autonomously. In Figure B-6, the SUAVs are assigned to the search tasks that the operator requested, and in this case each SUAV is assigned three

Table B.1: Agent Parameters for Demonstrated Mission Scenario

Agent Parameters		
Type A: WUAV	Capability	Weapons Release, Detection
	Max Velocity	0.18 m/s
	Cruise Velocity	0.12 m/s
	Nominal Altitude	1.1 m
	Fuel Capacity	4.5 units
	Initial Fuel	4 units
	Fuel Consumption Rate	0.01 units/sec
Type B: Sensor UAV	Capability	Detection, Tracking
	Max Velocity	0.18 m/s
	Cruise Velocity	0.12 m/s
	Nominal Altitude	1.3 m
	Fuel Capacity	4.5 units
	Initial Fuel	3 units
	Fuel Consumption Rate	0.01 units/sec
Type C: Sensor UAV	Capability	Detection, Tracking
	Max Velocity	0.18 m/s
	Cruise Velocity	0.12 m/s
	Nominal Altitude	1.4 m
	Fuel Capacity	4.5 units
	Initial Fuel	2 units
	Fuel Consumption Rate	0.01 units/sec

of the six tasks. The WUAV, however is not assigned to any search tasks, because it was intentionally not given that capability. The WUAV is considered a high value asset. The planner also schedules refuel times based on the agents' fuel states and the set of tasks that have been assigned.

When an agent locates a target, the operator is prompted to identify the target and give it a priority level. In Figure B-7, the target in blue is a friendly of high priority. Unknown and hostile targets are tracked throughout the mission. The planner schedules revisit locations and times based on the target's speed and priority level.

Hostile targets are engaged by the agents if the operator enables that capability. In the Figure B-8, the WUAV is engaging the pink target. Before weapons release is initialized, the operator must approve the launch based on an image sent from the WUAV, to confirm that the target is indeed an enemy.

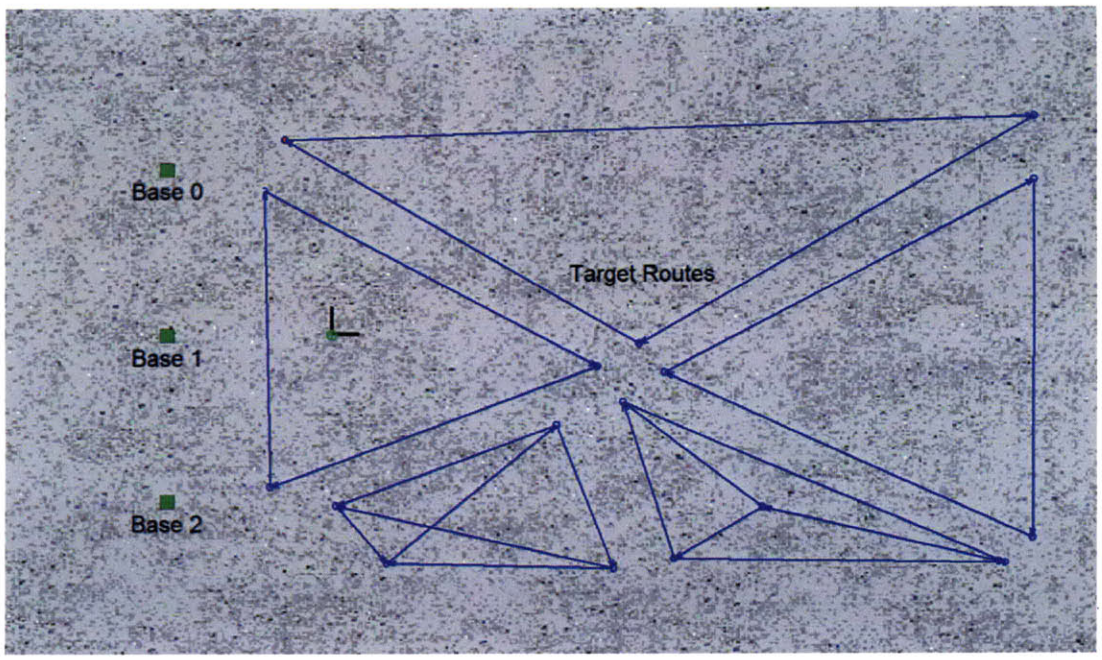

Figure B-4: Refuel Base Locations and Target Routes

Missions are typically longer than the flight endurance of the UAVs, so refueling is necessary. A refuel task consists of the agent flying to its base, landing, the ground crew switching the battery, and the UAV lifting off again. Figure B-9 shows UAV 1 being refueled by the ground crew while UAV 2 and the WUAV are still in the mission theater.

When a friendly target is declared high priority, a UAV may be assigned a protection task. During the protection task, the UAV patrols the area around the friendly, looking for other hostiles which may cause a threat to the friendly target. In Figure B-10, UAV 2 is patrolling the area around the blue target.

B.2.2 Mission Results

The results for one of the missions flown in RAVEN during the OPS-USERS demonstration are presented here. This mission represents the capabilities of the system as a whole. Table B.2 shows important mission statistics.

Figure B-11 indicates the times each of the targets were tracked during the mission. All five targets were found including the pop-up target. Figures B-12 and B-13 show

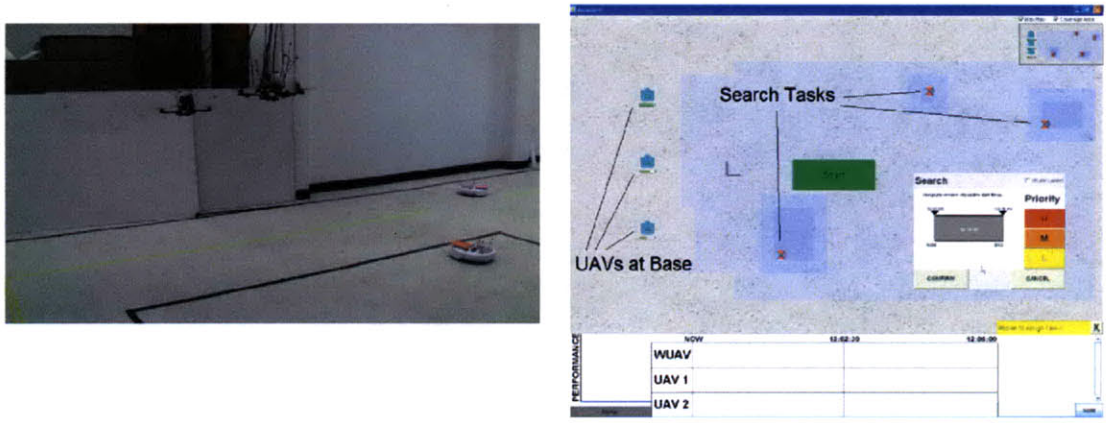

Figure B-5: Takeoff Sequence and Initial Search Task Selection

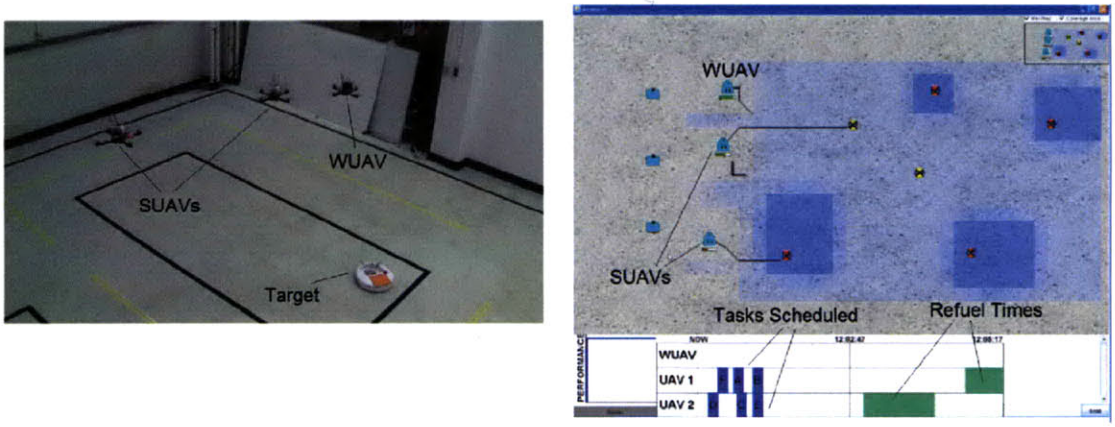

Figure B-6: Autonomous Task Allocation and Path Planning

the area searched during the mission. Even with refuel constraints, and very limited speed, the agents covered most of the environment by the end of the mission. The results of this mission demonstrate the capability of the OPS-USERS system to search and track in an unknown environment under human supervisory control. All five targets were found, and both hostiles were engaged.

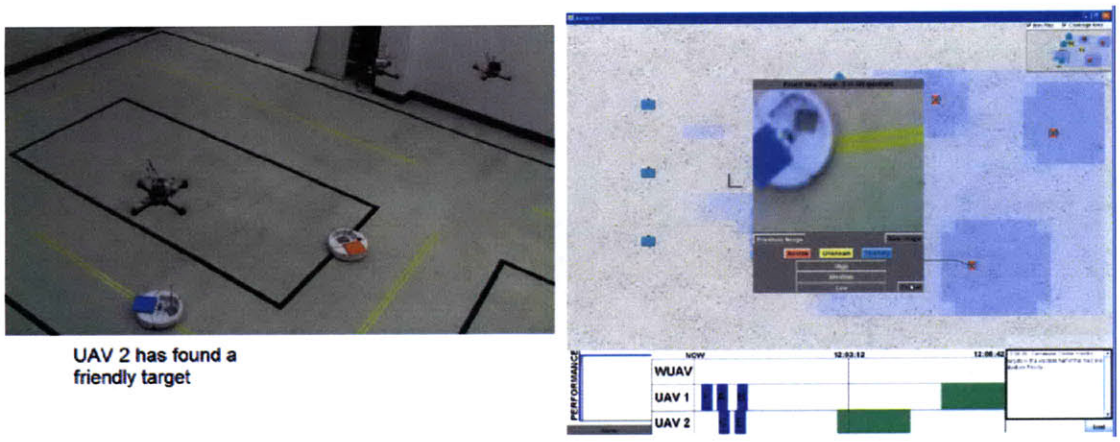

Figure B-7: New Target Discovered, Operator Prompted to Make Classification

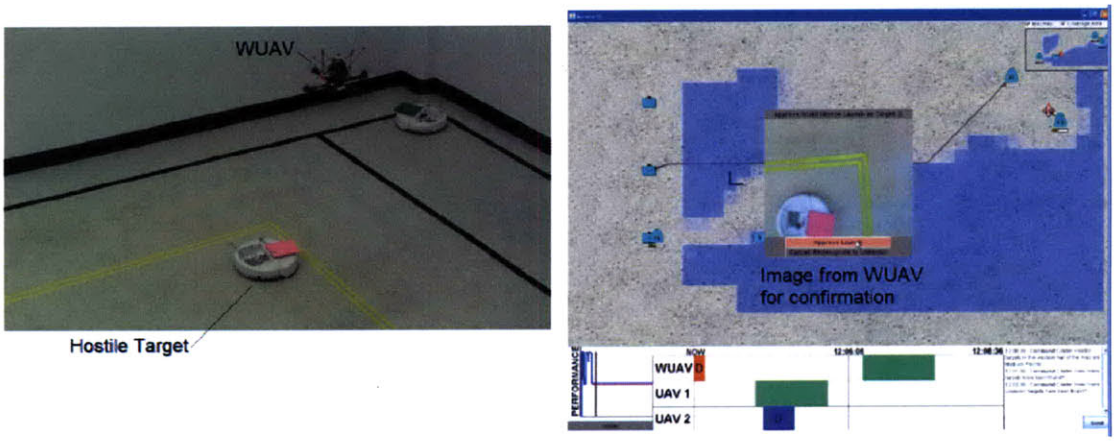

Figure B-8: WUAV Engages Hostile Target After Operator Approval

Table B.2: Performance Results from OPS-USERS Demonstration Mission

Performance	
Ground Searched	98.7 %
Total Area Searched	93.9 %
Targets Found	5/5
Averaged Time Targets Tracked	30.1 %
Average Response	20.7 sec
Hostiles Engaged	2/2

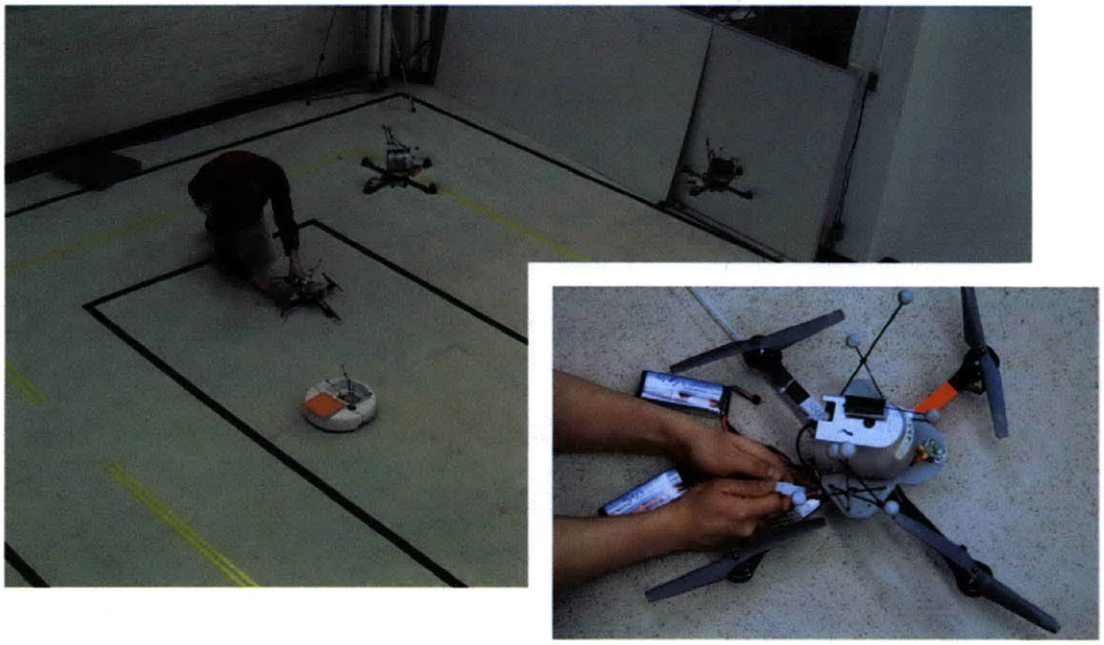

Figure B-9: UAV Refueling During OPS-USERS Mission

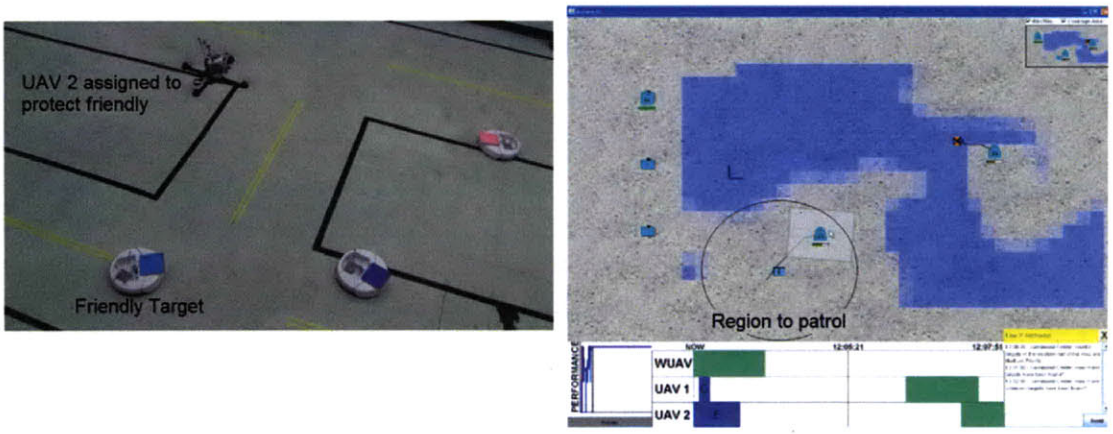

Figure B-10: Convoy Protection Task Execution

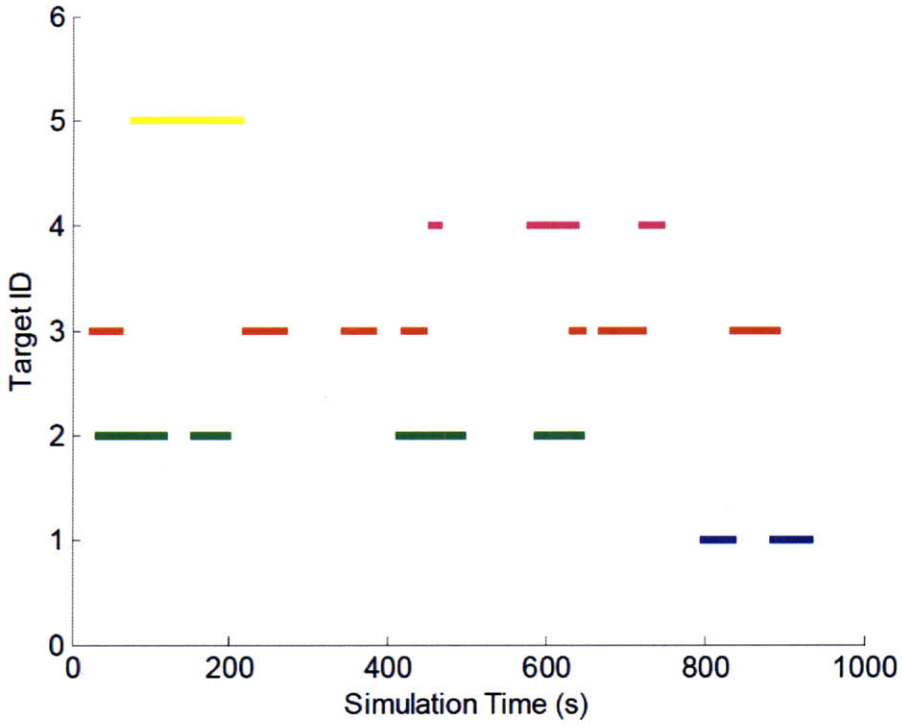

Figure B-11: Mission Time Targets Tracked

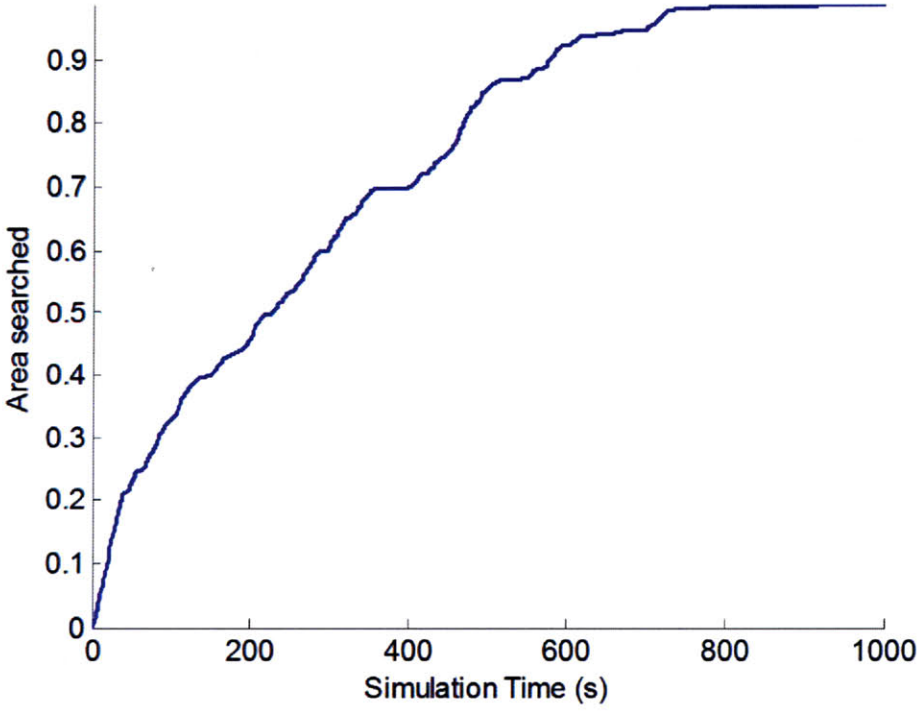

Figure B-12: Fraction of Ground Searched as Function of Time

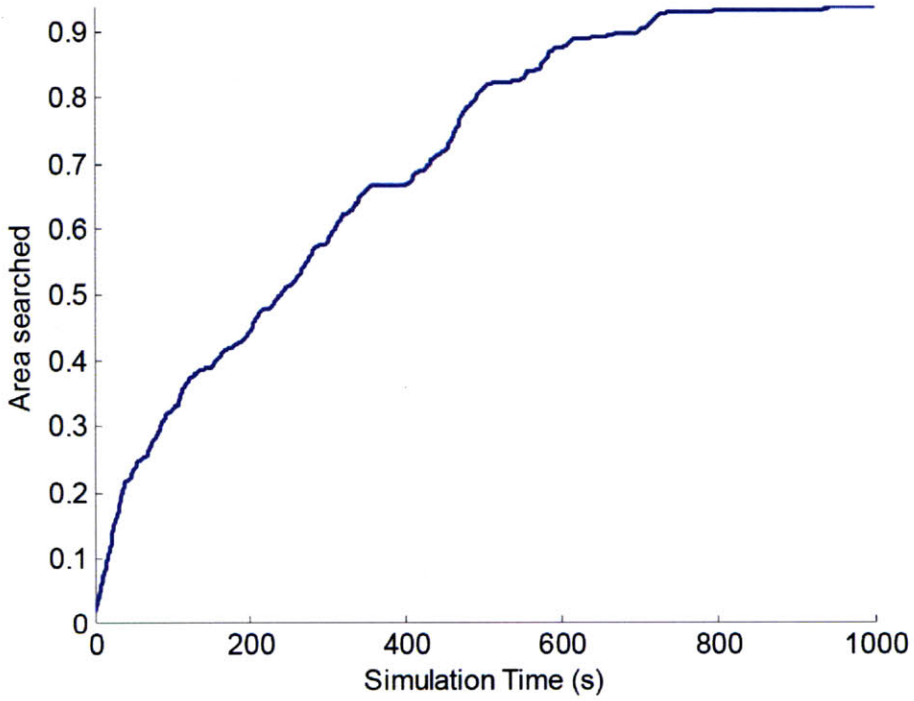

Figure B-13: Fraction of Total Environment Searched as Function of Time

Bibliography

[1] B. L. Brumitt and A. Stentz, "Dynamic mission planning for multiple mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2396–2401, 1996.

[2] M. Alighanbari, Y. Kuwata, and J. How, "Coordination and control of multiple uavs with timing constraints and loitering," in *American Control Conference*, vol. 6, pp. 5311–5316, 4–6 June 2003.

[3] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. K. Hedrick, "An overview of emerging results in cooperative uav control," in *In Proceedings of 43rd IEEE Conference on Decision and Control*, 2004.

[4] B. Bethke, J. P. How, and J. Vian, "Group health management of uav teams with applications to persistent surveillance," in *American Control Conference (ACC)*, pp. 3145–3150, 11-13 June 2008.

[5] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation." *Artificial Intelligence*, vol. 101, no. 1-2, May, 1998, pp. 165-200, 1998.

[6] J. Bellingham, M. Tillerson, A. Richards, and J. How, "Multi-Task Allocation and Path Planning for Cooperating UAVs," in *Proceedings of Conference of Cooperative Control and Optimization*, Nov. 2001.

[7] C. Cassandras and W. Li, "A receding horizon approach for solving some cooperative control problems," in *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[8] L. Xu and U. Ozguner, "Battle management for unmanned aerial vehicles," *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 4, pp. 3585–3590 vol.4, 9-12 Dec. 2003.

[9] M. Alighanbari, "Task assignment algorithms for teams of UAVs in dynamic environments," Master's thesis, Massachusetts Institute of Technology, 2004.

[10] M. Alighanbari, Y. Kuwata, and J. P. How, "Coordination and control of multiple uavs with timing constraints and loitering," in *American Control Conference (ACC)*, vol. 6, pp. 5311–5316 vol.6, 4-6 June 2003.

[11] S. Karaman and E. Frazzoli, "Vehicle routing with linear temporal logic specifications: Applications to multi-uav mission planning," in *AIAA Conf. on Guidance, Navigation, and Control*, (Honolulu, HI), 2008.

[12] T. W. McLain and R. W. Beard, "Coordination variables, coordination functions, and cooperative-timing missions," *Journal of Guidance, Control, and Dynamics*, vol. 28(1), pp. 150–161, 2005.

[13] T. Shima, S. J. Rasmussen, and P. Chandler, "Uav team decision and control using efficient collaborative estimation," in *Proceedings of the American Control Conference*, pp. 4107–4112 vol. 6, 8-10 June 2005.

[14] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control System Magazine*, vol. 27, pp. 71–82, April 2007.

[15] W. Ren, R. W. Beard, and D. B. Kingston, "Multi-agent Kalman consensus with relative uncertainty," in *Proceedings of the American Control Conference*, pp. 1865–1870 vol. 3, 8-10 June 2005.

[16] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1520–1533, Sept. 2004.

[17] M. Alighanbari and J. P. How, "An unbiased kalman consensus algorithm," in *American Control Conference (ACC)*, pp. 3519–3524, 14-16 June 2006.

[18] M. Alighanbari and J. P. How, "An Unbiased Kalman Consensus Algorithm," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 5, pp. 298–311, Sept 2008.

[19] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006.

[20] W. Ren and R. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on Automatic Control*, vol. 50, pp. 655–661, May 2005.

[21] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65 – 78, 2004.

[22] M. Alighanbari and J. How, "Decentralized task assignment for unmanned aerial vehicles," in *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC '05)*, pp. 5668–5673, 12–15 Dec. 2005.

[23] M. B. Dias and A. Stentz, "A free market architecture for distributed control of a multirobot system," in *In 6th International Conference on Intelligent Autonomous Systems IAS-6*, pp. 115–122, 2000.

[24] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94(7), pp. 1257–1270, 2006.

[25] D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems," tech. rep., MIT, 1989.

[26] D. P. Bertsekas, "Auction algorithms," in *Encyclopedia of Optimization*, Kluwer Academic Publishers, 2001.

[27] B. Gerkey and M. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18(5), pp. 758–768, 2002.

[28] M. J. Mataric, G. S. Sukhatme, and E. H. Ostergaard, "Multi-robot task allocation in uncertain environments." *Autonomous Robots*, vol. 14, no. 2-3, pp. 255-263, 2003.

[29] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23(9), pp. 939–954, 2004.

[30] D. A. Castanon and C. Wu, "Distributed algorithms for dynamic reassignment," *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1, pp. 13–18 Vol.1, 9-12 Dec. 2003.

[31] L. Brunet, "Consensus-Based Auctions for Decentralized Task Assignment," Master's thesis, Dept. of Aeronautics and Astronautics, MIT, June 2008.

[32] H. L. Choi, L. Brunet, and J. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, 2009.

[33] S. Ponda, H.-L. Choi, and J. P. How, "Predictive planning for heterogeneous human-robot teams," in *AIAA Infotech@Aerospace*, April 2010.

[34] S. Ponda, J. Redding, H.-L. Choi, B. Bethke, J. P. How, M. Vavrina, and J. L. Vian, "Decentralized planning for complex missions with dynamic communication constraints," in *American Control Conference*, 2010.

[35] L. B. Johnson, S. Ponda, H.-L. Choi, and J. P. How, "Improving the efficiency of a decentralized tasking algorithm for UAV teams with asynchronous communications," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2010.

[36] H.-L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," in *American Control Conference (ACC)*, July 2010.

[37] P. Chandler, M. Pachter, D. Swaroop, J. Fowler, J. Howlett, S. Rasmussen, and C. Schumacher, "Complexity in UAV Cooperative Control," in *Proceedings of the Americal Control Conference*, 2002.

[38] Y. Jin, A. Minai, and M. Polycarpou, "Cooperative Real-Time Search and Task Allocation in UAV Teams," in *Proceedings of the IEEE Conference on Decision and Control*, 2003.

[39] T. S. Joao Sousa and P. Varaiya, "Task planning and execution for uav teams," in *43rd IEEE Conference on Decision and Control*, 2004.

[40] C. Schumacher, P. Chandler, M. Pachter, and L. Pachter, "Constrained optimization for uav task assignment," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2004.

[41] M. Flint, M. Polycarpou, and E. Fernández-Gaucherand, "Cooperative path-planning for autonomous vehicles using dynamic programming," in *Proceedings of the IFAC World Congress*, (Barcelona, Spain), 2002.

[42] M. Flint, E. Fernandez-Gaucherand, and M. Polycarpou, "A probabilistic framework for passive cooperation among uavs performing a search," in *Proc. of the Sixteenth International Symposium on Mathematical Theory of Networks and Systems (MTNS2004)*, 2004.

[43] F. Bourgault, T. Furukawa, and H. F. Durrant-Whyte, "Optimal search for a lost target in a bayesian world," in *Field and Service Robotics, vol. 24, pp. 209-222*, 2006.

[44] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. on Robotics*, vol. 25 (4), pp. 912 – 926, 2009.

[45] M. Cummings, J. How, A. Whitten, and O. Toupet, "The impact of human-automation collaboration in decentralized multiple unmanned vehicle control," *IEEE Journal (submitted)*, Aug 2010.

[46] M. Valenti, B. Bethke, G. Fiore, J. P. How, and E. Feron, "Indoor multi-vehicle flight testbed for fault detection, isolation, and recovery," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, (Keystone, CO), August 2006.

[47] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, pp. 51–64, April 2008.

[48] MIT Real-time indoor Autonomous Vehicle test ENvironment (RAVEN), "RAVEN home page." `http://vertol.mit.edu/`, 2007.

[49] J. How, C. Fraser, K. Kulling, L. Bertuccelli, O. Toupet, L. Brunet, A. Bachrach, and N. Roy, "Increasing autonomy of UAVs," *IEEE Robotics & Automation Magazine*, vol. 16, pp. 43–51, June 2009.