Consensus-Based Auctions for Decentralized Task Assignment

by

Luc Brunet

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

June 2008

Author . . . .
Department of Aeronautics and Astronautics
May 23, 2008

Certified by . . . .
Jonathan P. How
Professor
Thesis Supervisor

Accepted by . . . .
Professor David L. Darmofal
Associate Department Head
Chair, Committee on Graduate Students

Consensus-Based Auctions for Decentralized Task Assignment

by

Luc Brunet

Submitted to the Department of Aeronautics and Astronautics
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis addresses the decentralized task assignment problem in cooperative autonomous search and track missions by presenting the Consensus-Based class of assignment algorithms. These algorithm make use of information consensus routines to converge on the assignment rather than the situational awareness of the fleet. A market-based approach is used as the mechanism for task selection, while the novel consensus stage of the algorithms allow for fast distributed conflict resolution. Three separate algorithms belonging to the Consensus-Based class of assignment strategies will be presented. The first is the Consensus-Based Auction Algorithm (CBAA), which is a single assignment auction strategy that is shown to be bounded within 50% of the optimal solution, while an upper-bound on convergence is presented. Two multi-assignment algorithms are then presented as extensions of the CBAA. The iterative CBAA executes the single assignment algorithm multiple times in order to build an assignment with multiple tasks. The second algorithm is the more general Consensus-Based Bundle Algorithm (CBBA) in which agents build a candidate bundle of tasks and bid on each task individually based on the improvement in score achieved by adding it to the bundle. Both algorithms are shown to be lower bounded by 50% optimality, while convergence bounds are derived based on the network topology. Numerical results show that the bundle algorithm performs much better than the iterative approach while providing faster convergence times. It is also compared with the Prim Allocation (PA) auction algorithm where it is shown to exhibit much faster convergence times and give better assignments. The CBBA is also implemented in the CSAT simulation test-bed developed by Aurora Flight Sciences in conjunction with MIT, and shown to produce faster response times and better tracking performance than the currently used RDTA algorithm.

Thesis Supervisor: Jonathan P. How
Title: Professor

Acknowledgments

I would like to thank my advisor, Professor Jonathan How, for giving me the opportunity to work in this area of research and guiding me over the last 2 years. My colleague, Han-Lim Choi, whose contribution and advice, especially on the theoretical aspect of this work, was invaluable. My lab-mates, Brandon Luders, Frantisek Sobolic, Buddy Michini, Cameron Fraser, Karl Kulling, Ray He, Brett Bethke, Josh Redding, Jim McGrew and Spencer Ahrens, for always being available to work through new ideas, for periodic distractions, and for making the long hours in the lab an extremely enjoyable experience. I would also like to thank Jim Paduano at Aurora Flight Sciences for the opportunity to work on an exciting project and in particular, Olivier Toupet, whom I learned a great deal from and who showed a lot confidence in my work. It was a pleasure working with you.

I would like to thank Jaime Mateus and all of my friends here in Boston who have made the adventures outside of the lab so enjoyable. We gave'r!! My friends back in Canada, who have always managed to make it hard to come back after a visit. I would like to especially thank Jodie-Lee Primeau for supporting me through absolutely everything over the last few years. In many ways, I would not be the person I am today if it weren't for her. I would like to thank my brother Curtis and sister Kaitlin for all of their support, and for their multiple visits while I have been away. I love you both. My cousins Mark and Michael, Aunt Lynda and Uncle Rick for always showing interest and making an effort to visit when I was around. It is always great to see you guys. I would like to thank my Mom for her concern and help through every single one of the problems I have been faced with. Most of all, I would like to thank my Dad, who has inspired me my entire life and has helped me in more ways than I think he could ever know. I would have never made it this far without you.

This work was funded by AFOSR STTR Contract FA9550-06-C-0088

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Unmanned vehicles (UVs) have gained increased popularity over the years. Today, UVs are actively used in areas such as military operations [3–5], search and rescue [6, 7], perimeter security, underground mining [8], and hazardous environment exploration [9–11]. They are able to provide increased operation time while in many cases, minimizing the cost to complete a given task. With the lack of human occupancy onboard, UVs are well suited for a wide range of missions that a manned-vehicle simply could not perform, or would be considered too dangerous. As the demand for unmanned operation has increased, so too has the increase in autonomy of the individual vehicles, resulting in an increase in productivity and operational flexibility. This shift has allowed missions that once required many humans to operate a single vehicle, to be performed with only one human controlling many vehicles. With the success of the recent DARPA Grand Challenge [12], it is not a far stretch to imagine that most transportation systems in the future will have some autonomous capabilities.

Although individual autonomous vehicles have proven useful in many areas of operation, it is ultimately teams of such vehicles that will provide the greatest benefit. Combining vehicles with different capabilities into a heterogeneous fleet will allow for more flexible missions and ultimately increase the utility. For example, search missions would be able to cover more area as extra vehicles are added, surveillance missions would benefit from the increased coverage as well as provide redundancy in target localization. Groups of vehicles would also be able to complete missions

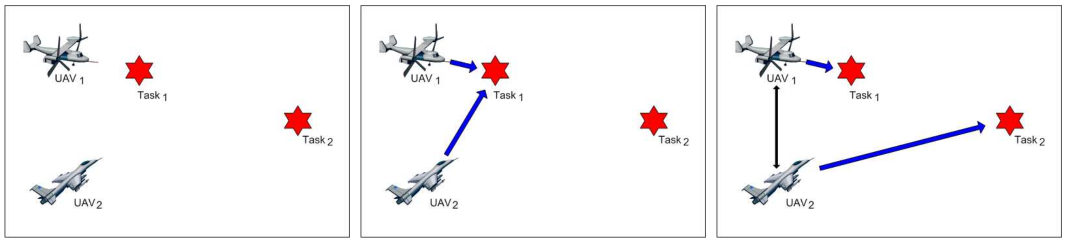


Figure 1-1: Two UAVs must complete the two tasks shown. Without cooperation, assignments are conflicted and a task is left unassigned. By cooperating, the fleet is able to achieve higher mission performances.

requiring many different payloads as well as give the system a degree of robustness to vehicle or payload failure [13]. Flexibility also exists in allowing vehicles to act independently, as a team, or by creating sub-teams with different mission objectives.

In any case, the value of having multiple vehicles is apparent and can increase the performance of almost any mission. The difficulty lies in coordinating the vehicles so that they improve performance and not hinder it. Consider a simple fleet of two Unmanned Aerial Vehicles (UAVs) that must each complete one of the two tasks shown in Figure 1-1. *Without cooperation*, they will each select the closest task to perform and complete it without considering what the other vehicle is doing. This leads to a conflicting assignment since both have selected the same task and the second task will go unassigned. *With cooperation* however, UAV 1 will realize that UAV 2 is much closer to the first task, and thus, better able to perform it. UAV 1 can then select the second task in order to maximize the value of the overall assignment.

The simple example described above illustrates the need for cooperation in missions with multiple vehicles. When the vehicles did not cooperate, they not only failed to obtain the best possible assignment, but they also performed the same task making one of the two vehicles unnecessary. This wastes resources and increases cost for no reason. Therefore, a strongly coordinated and cooperative fleet is necessary to fully achieve the benefits of a multi-vehicle platform.

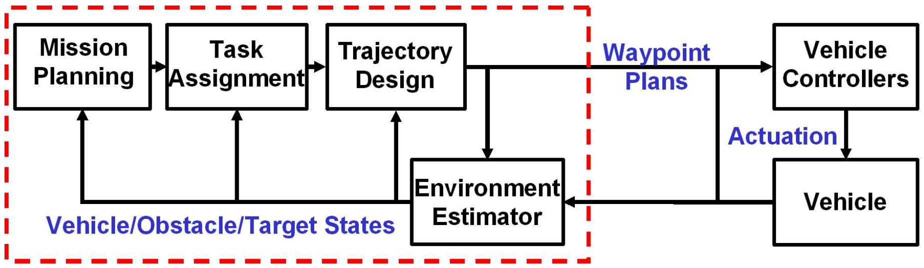


Figure 1-2: Cooperative Planning System Architecture - image taken from [1]

1.1 Motivation

Cooperation amongst a fleet of unmanned vehicles is important in order to improve the performance of a given mission. Without it, resources can be wasted and mission costs might increase without any significant improvement in performance. However, in a complex system with many vehicles and tasks, the coordination of vehicles is not easily achievable. Vehicles have different capabilities, their states are constantly changing, the states estimates of the tasks are dynamic and have a degree of uncertainty that is different for each member. The environment might also be dynamic or unknown, vehicle sensors can be very noisy, and many other factors exist that might hinder the coordination of vehicles in the fleet. It is thus important to develop algorithms that can efficiently produce plans in dynamic, noisy and uncertain environments.

A general cooperative planning architecture can be found in Figure 1-2. In these types of systems, a set of tasks is generated by a mission manager (MM), which are then divided between the members fleet using a task assignment algorithm. Once this is done, detailed trajectories can be generated for task execution to complete the mission. In cooperative systems, this task assignment process is extremely important since it is the mechanism for which members of the fleet will partition the assignment space amongst themselves. This can either be done off-line, before the mission is performed, or periodically as new information is received. Similar to the example listed in the previous section, it is important that the assignment be done cooperatively to improve performance and to ensure the mission objective is achieved. Difficulties lie in agreeing on the many different information sets required to make a decision.

One vehicle might think it has an accurate estimate of a task location, while another might think it is in an entirely different area of the map. If the vehicles have trouble agreeing on even this simple set of information, how then can they agree on a correct assignment?

This thesis addresses the task assignment problem with a focus on search and track missions. In these types of missions, a set of heterogenous vehicles might begin with some *a priori* information about the whereabouts of possible targets, and are tasked with searching the environment while keeping track of the targets that are found. Vehicles might vary in type and capability such as fixed wing high flying aircraft for searching, helicopters for tracking, or ground and water vehicles for tasks involving their respective environment types. Tracking a task in these types of missions involves acquiring a state estimate of the target, and using it to produce an estimate of the target's position at a later time in the mission for revisiting. The trade-off is then to increase the period between revisits so that the vehicles can perform other tasks, while ensuring (with sufficiently high confidence) that the target is where it was predicted to be when it is revisited. By optimizing the assignment, tasks will be tracked more efficiently and the vehicles will be able to handle more tasks and search the environment more effectively.

1.2 Literature Review

Many different methods exist that can be used by autonomous agents to distribute tasks amongst themselves from a known task list. Some involve centralized planning systems [14–21] in which vehicles communicate their situational awareness (SA) to a centralized server. With this information, the server can generate a plan for each vehicle and distribute it to the entire fleet. These types of systems are useful since they place much of the heavy processing requirements safely on the ground, making the vehicles smaller and cheaper to build. They also benefit from having a single SA in which the server can quickly generate plans for the entire fleet and react to new information as it arrives. On the other hand, this may force the vehicles to remain in constant communication with a specific area in the environment (to stay in contact with the central planner), reducing the possible mission ranges and creating a single point of failure. The assignment algorithm can also be computationally intensive for large fleets and may not scale well.

Decentralized approaches have thus been developed by instantiating the centralized planner on each vehicle in order to increase the mission range, and remove the single point of failure [22–26]. These distributed methods can reduce the computational costs and add increased flexibility, however, they often require perfect communication links with infinite bandwidth since each vehicle is assumed to have the same SA. If this is not the case, inconsistencies in the SA might cause conflicting assignments since each vehicle will be performing the centralized optimization with a different information set. Thus, decentralized algorithms generally make use of consensus algorithms [27–34] to converge on a consistent SA before calculating the assignment [35]. These consensus algorithms can guarantee convergence of the SA over many different dynamic network topologies [34, 36, 37], allowing the fleet to perform the assignment in highly dynamic and uncertain environments.

On the other hand, consensus algorithms can take a significant amount of time to converge on the SA and can often require transmitting large amounts of data to do so. This can cause severe latency in low bandwidth environments and can substantially increase the time it takes to find an assignment for the fleet. Various algorithms have been developed that attempt to reduce the communication required to ensure convergence to a conflict free solution. In [38], it is shown that it is possible to filter out unnecessary information while still maintaining an optimal solution. This both helps reduce computational load and the amount of communication that is required to produce an assignment. In [39], communication is reduced by maintaining a local and global (previously shared) SA. By doing this, each vehicle can compute an assignment based on each information set, and will communicate only if the assignments differ. Hierarchical approaches [40–42] can sometimes reduce the communication costs by forming sub-teams and replacing large networks with small dense communication areas instead.

Although the reduction in computation and communication is improved in the previous methods discussed, a conflicting assignment might not be avoided with an inconsistent SA. To account for this, the Robust Decentralized Task Assignment (RDTA) algorithm is proposed in [43, 44]. The algorithm is robust to inconsistencies in the SA by making use of a two stage optimization process. In the first stage, each agent creates several candidate plans based on their own SA, which are then communicated to the rest of the fleet. In the second stage, each agent then optimizes over the received plans to generate the final assignment. The algorithm is thus able to reduce the consensus time and communication overhead needed by not forcing the convergence to a consistent SA to ensure a conflict-free assignment. This algorithm might still however take a significant amount of time to produce a solution since each agent must wait to receive the plans generated by all of the members before performing the final optimization. It also restricts the network types from the algorithms developed using the consensus approach since each vehicle must be able to pass the plans in the first stage to every other vehicle at a specific time.

Auction algorithms [45–49] are another method for task assignment that have been shown to be efficient both in terms of communication and computation [50]. Generally, agents place bids on tasks and the one with the highest bid wins the assignment. The traditional way of computing the winner is to have a central system act as the auctioneer to receive and evaluate each bid in the fleet [51–53]. However, in many cases involving robotic agents, the central system is removed and one of the bidders acts as the auctioneer [47, 54–57]. Once all of the bids have been collected, a winner is selected based on a pre-defined scoring metric. In these types of algorithms, agents bid on tasks with values based solely on their own SA. It is known that each task will only be assigned to a single agent since only one agent is selected by the auctioneer as the winner. Because of this, most auction algorithms can naturally converge to a conflict-free solutions even with inconsistencies in their SA.

The downside of these approaches is that the bids from each agent must somehow be transmitted to the auctioneer. Similar to the RDTA algorithm, this limits the network topologies that can be used since a certain amount of connectivity is required between the agents in order to route all of the bid information. A common method to avoid this issue is to run the auction solely within the set of direct neighbors of the auctioneer [58, 59]. However, this can reduce the mission performance by not considering the rest of the fleet for tasking. Smith and Bullo [60] present an approach that removes the auctioneer altogether. Each agent calculates the optimal Euclidean Traveling Salesman Problem (ETSP) tour of the tasks and uses it to efficiently transmit the tasks that are available through the fleet. Although the algorithm can perform quite well in dynamic environments, it relies on perfect information in the task SA in order to ensure common ETSP tours throughout the fleet, which may not be possible in a realistic environment.

Various efforts have been made in the literature to extend the auction class of algorithms to the multi-assignment case. This is sometimes done by sequentially auctioning each target individually until there are no remaining tasks left to assign [53, 54, 59, 61, 62]. These methods can provide an easy way to implement a multi-assignment algorithm, but they can be slow to converge and, depending on the implementation, may provide poor assignments. Bundle approaches [63–66] have been developed that group common tasks into bundles and vehicles bid on groups rather than the individual tasks. By grouping similar tasks, these types of algorithms will converge faster than their iterative counterparts since a single conflict resolution will apply to multiple tasks. They will also have improved value in the assignment since they can logically group tasks that have commonalities. However, difficulties can arise in the computational cost of enumerating all possible bundle combinations, and winner determination has been shown to be $\mathcal{NP}$-complete [67], requiring the use of specialized winner determination algorithms [68–70].

The use of these techniques to perform search and track missions has been developed over recent years [13, 17, 71–77]. During the mission, a set of tasks will be identified that the agents must distribute amongst themselves. This list can include tasks such as tracking and classification of discovered targets, searching a specific area of the map, or even providing communication support for other agents in the fleet. In some cases the map is partitioned into separate search zones while vehicles are tasked to enter the area and execute some pre-defined search maneuver [78–81]. These algorithms can either store canned paths, or compute them online to optimally cover the area to be searched. The quality of the overall search however depends heavily on the partitioning of the map. A finer partition means that the value of searching a given area is more accurately known and the search is more efficient. Conversely, as the number of partitions is increased, the assignment can become increasingly difficult to perform. In other search methods, the map is partitioned into cells over which there is a probability distribution of targets [82–84]. A transition model of the targets is kept and used to model their movement between cells. The map can then be efficiently searched by maximizing the vehicle trajectories over the probability distributions. This however, causes coupling between the assignment and the path generation algorithm, making it difficult to find a proper solution [85]. To add to this difficulty, search regions can be very large and complex, causing intermittent and noisy communication with highly dynamic network topologies. Because of this, much of the literature for these types of missions has focused on enforcing strict network topologies such as a fully connected network [24, 64, 75, 86], a static connected network with routing capability [43, 53], or sometimes the assignment is simply done within a local sub-network [58, 59].

1.3 Objectives

The objective of this thesis is to develop a set of decentralized task assignment algorithms that are suitable for search and track missions in large environments with limited communication. Task assignment for these types of missions should be

1. Flexible to varying network structures and communication linkages
2. Robust to dynamic and uncertain environments
3. Guarantee convergence with an inconsistent SA
4. Provide fast convergence times with optimal or near-optimal solutions

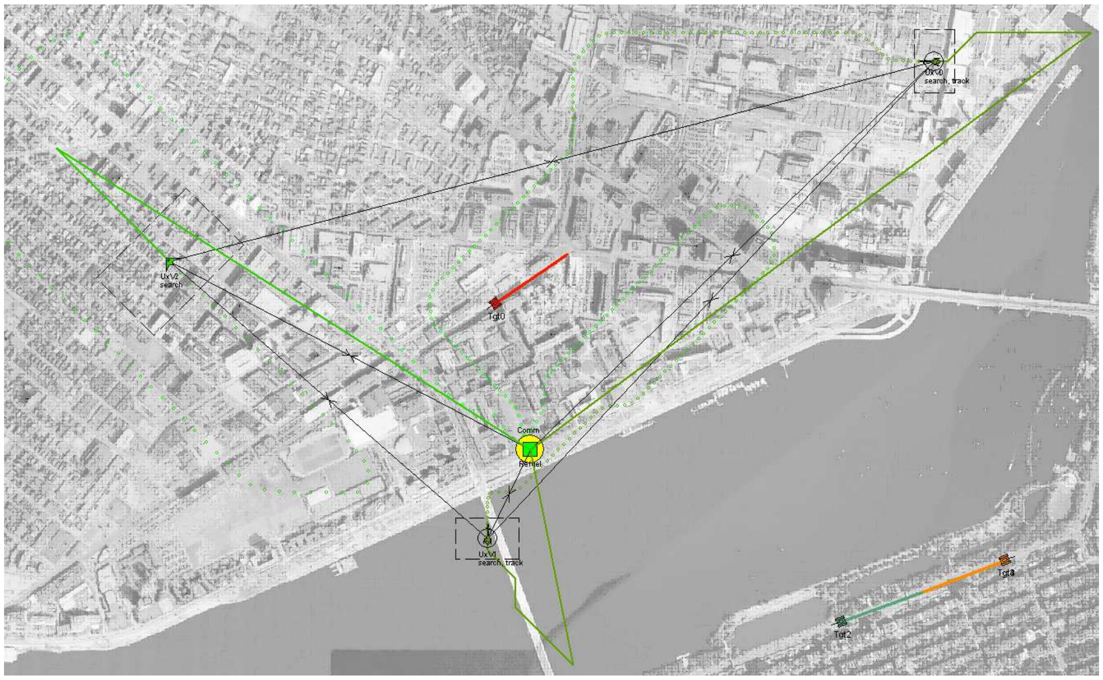


Figure 1-3: Three unmanned aerial vehicles search for four ground targets (tanks) in a known environment

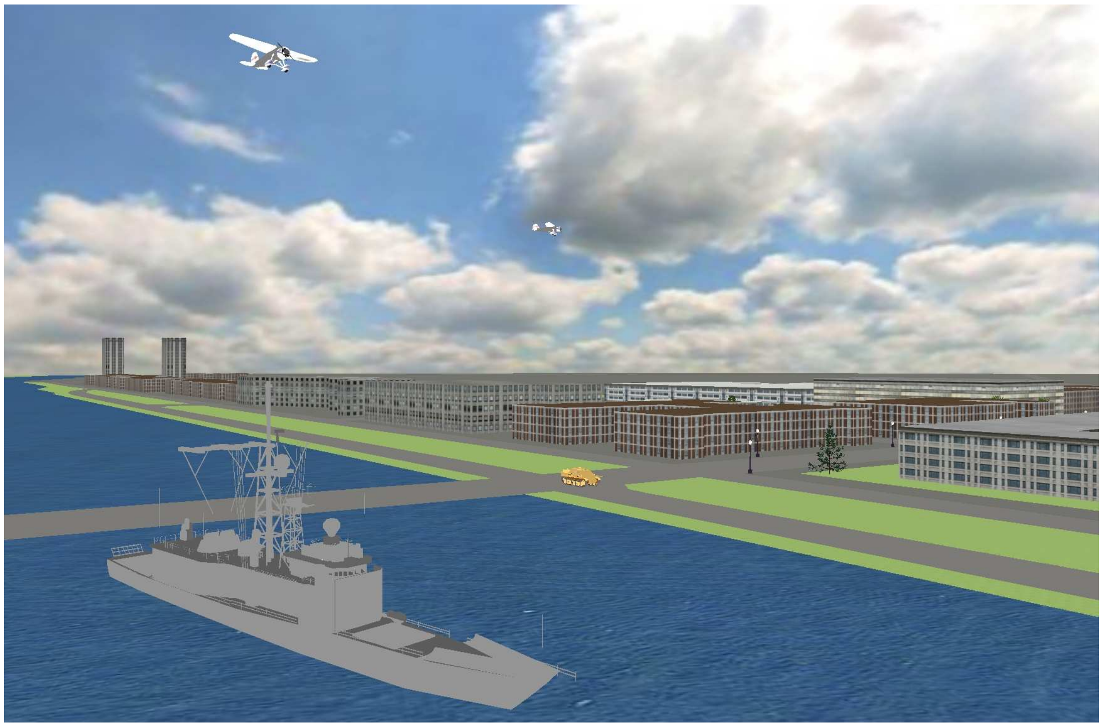


Figure 1-4: Vehicles fly in a 3D environment while searching for targets

The end-goal of this thesis is to implement the developed algorithms into the CSAT (Coordinated Search, Acquisition and Track) simulation environment (Figures 1-3 and 1-4) developed by Olivier Toupet at Aurora Flight Sciences in conjunction with MIT. This simulation environment is a multi-vehicle search and track platform in which any number of ground, water and air vehicles can coordinate to perform a search and track mission in many different environments. Algorithms can be implemented and executed in real-time to observe their behavior. The goal of the implementation is to provide a task assignment algorithm robust to varying network topologies and communication dropouts, that are common in search and track missions. The algorithm should be efficient in handling a task list much larger than the number of agents, while being quick to react as new information is inserted into the environment.

1.4 Overview

This thesis is structured as follows: Chapter 2 presents the consensus-based auction algorithm (CBAA). This is a single-assignment algorithm that can produce near-optimal assignments in dynamic uncertain environments. Chapter 3 will extend the CBAA to the multi-assignment case by first presenting the iterative single-assignment CBAA, and then developing the more general Consensus-Based Bundle Algorithm (CBBA). Various mechanisms will also be presented here to account for missions constraints such as vehicle capability, periodic ground station communication, refueling etc... Chapter 4 will present the implementation of the CBBA in the CSAT simulation environment, with some analysis of its performance in search and track missions. Finally, Chapter 5 will conclude the thesis with a summary of the work and contributions that have been made.

Chapter 2

Consensus-Based Auction Algorithm

Cooperation amongst a fleet of robotic agents is necessary in order to improve the overall performance of any mission. In the previous chapter, two main types of distributed tasking mechanisms were identified. The first was algorithms that make use of consensus approaches to converge on the SA, and then perform some type of optimization to arrive at an assignment [22–25, 35]. These approaches are flexible in network structure and can easily compute the optimal assignment. The second type of algorithms discussed are able to add robustness to inconsistencies in the SA, thus reducing consensus time and allowing for more realistic implementations [43, 54–57]. However, these generally enforced some strict network structure. The objective of this chapter is to develop a decentralized algorithm that combines properties from both types of assignment strategies. The algorithm is designed to produce near-optimal solutions in highly flexible network topologies, but can also quickly converge to a solution in uncertain and dynamic environments with inconsistencies in the SA.

This chapter will develop the Consensus-Based Auction Algorithm (CBAA), which was first proposed in [44]. This work extends the latter by further developing the CBAA as an algorithm for uncertain environments and provides significantly advanced analysis into its convergence and performance properties. The CBAA is a distributed and greedy auction strategy with a consensus-like step for conflict resolution. Instead of passing bids to a single source for evaluation, bids are made and conflicts are resolved through the network by running a information consensus routine on a winning bids list. Conflicts can then be resolved without requiring that the network be connected at a specific instant in time for bidding, and like many auction algorithms [47, 53, 59], converges regardless of inconsistencies in the SA. The CBAA is also shown to produce a near-optimal solution while maintaining fast convergence times.

2.1 Background

This section presents background information for the task assignment problem, as well as an outline of traditional auction and consensus methods.

2.1.1 Task Assignment Problem

The objective of the assignment problem in this thesis is, given a list of N_t tasks and N_u agents, to find a conflict-free assignment of tasks to agents that maximizes some global objective. An assignment is said to be free of conflicts if each task is assigned to no more than one agent. For each task j, agent i is awarded a score c_{ij} if it is assigned task j. Without loss of generality, this value is assumed to be nonnegative. The objective of the assignment is to maximize the overall score of the entire fleet. The problem can be formulated as follows:

$$\begin{aligned}
&\max \sum_{i=1}^{N_u} \sum_{j=1}^{N_t} c_{ij} x_{ij} \\
&\text{subject to} \quad \forall i = \{1, \ldots, N_u\} : \sum_{j=1}^{N_t} x_{ij} \leq 1 \\
&\qquad\qquad\quad \forall j = \{1, \ldots, N_t\} : \sum_{i=1}^{N_u} x_{ij} \leq 1 \\
&\qquad\qquad\quad \forall i = \{1, \ldots, N_u\},\ \forall j = \{1, \ldots, N_t\} : x_{ij} \in \{0, 1\}
\end{aligned} \tag{2.1}$$

Various approaches can be used to solve the above optimization, including Mixed Integer Linear Programming (MILP) [14, 85, 87], auction algorithms [46, 48], and network flow methods [21, 88]. An overview of various approaches for both centralized and decentralized architectures is given in [26].

2.1.2 Auction Algorithms

Auction algorithms are a well established method of addressing the task assignment problem. In centralized auction systems [45], the value of a task is given by $c_{ij} = a_{ij} - p_j$, where a_{ij} is the reward of assigning task j to agent i and p_j is the global price of task j. As the assignment progresses, the value of p_j is continuously updated to reflect the current bid for the task. The algorithm is started with any initial assignment (possibly randomly selected) and a set of initial task prices. Auctions are done in rounds and continue until all agents are satisfied with their assignment. An agent is said to be satisfied if it is assigned to the task giving it the maximum value ($\max_j c_{ij}$). If this is not the case, at the beginning of a round, some agent i which is not satisfied with its assignment is selected and the task that gives it a maximal reward is determined

$$j^\star = \operatorname*{argmax}_j a_{ij} - p_j. \tag{2.2}$$

If task $j^\star$ has already been assigned to another agent, the two agents swap tasks. Once this is done, the price of task $j^\star$ is increased such that the value $c_{ij^\star}$ is the same as the second highest valued task in agent i's list

$$p_{j^\star} = a_{ij^\star} - \max_{j \neq j^\star} (a_{ij} - p_j) + \epsilon \tag{2.3}$$

where $\epsilon > 0$ is some minimum price increment. This continues until the algorithm has converged to the final assignment and all agents are satisfied.

The centralized auction algorithm makes use of a global price list p_j that each agent can edit and has complete access to at all times. In decentralized systems, agents do not have global access to this information and thus, the price update in (2.3) is generally not performed. In these cases, the task scores are calculated using $c_{ij} = a_{ij} - p_{ij}$, where p_{ij} is the local price for agent i to complete task j, and bids for auctioned tasks are submitted to an auctioneer. In some cases the auctioneer is a central server [51], while in other cases the role is performed by the agents themselves [47, 56, 57]. The auctioneer collects all of the bids c_{ij} from each agent for a specific task j, and then selects the winner $i^\star$ based on the highest bid

$$i^\star = \underset{i}{\operatorname{argmax}}\, c_{ij}. \tag{2.4}$$

This process continues until each task has been assigned.

The purpose of the auctioneer in these decentralized methods is to avoid conflicts in the assignment. Tasks are sequentially put up for auction and only a single agent can win a task. However, if the task list is large, auctioning each task individually may be time-consuming, and furthermore, agents that are not in range at the auctioning time will never be considered for the assignment. Thus, other decentralized auction algorithms have been developed that remove the auctioneer in place of a different conflict resolution approach, and allow tasks to be bid on asynchronously. In the ETSP algorithm [60], each agent calculates a constant factor approximation of the ETSP tour of the set of tasks $\mathcal{Q} = \{q_1 \ldots q_{N_t}\}$, such that $tour(\mathcal{Q}) = \{q_{\sigma_1} \ldots q_{\sigma_{N_t}}\}$ is the ordered list of tasks along the tour, and σ_j is the j-th index along the tour. It is assumed in [60] that each vehicle knows the task locations precisely, such that each vehicle creates the same tour. During the mission, each agent i greedily selects the best task $curr^{[i]}$ and calculates the next available ($next^{[i]}$) and previous available ($prev^{[i]}$) tasks on the tour. Since each tour is the same, an agent k ($\neq i$) can observe upon receiving these values that all tasks in between $prev^{[i]}$ and $next^{[i]}$ along the tour have already been selected and can thus be removed from consideration. By doing this, agents can directly resolve conflicts and quickly prune their task lists to reduce the chance of future conflicts.

Although the auction algorithms listed above have been shown to efficiently produce near-optimal assignments, various limitations still exist. Centralized approaches, along with the auctioneer tasking methods, require that bids be submitted at a given instant to a specific location. This places a requirement for a fixed network structure that is connected at a specific time in the assignment. Other algorithms have removed the auctioneer to allow for flexible network structures, but require consistent task knowledge over the entire fleet in order to guarantee convergence. The auction approach developed herein uses a consensus algorithm for conflict resolution, which will be shown to allow flexible network structures without requiring consistent information over the fleet.

2.1.3 Consensus Algorithms

For decentralized systems, cooperating agents often require a globally consistent SA [22, 24]. In a dynamic environment with sensor noise and varying network topologies, maintaining a consistent SA throughout the fleet can be very difficult. Consensus algorithms are used in these cases to enable the fleet to converge on some specific information set before generating a plan [35]. Examples of typical information sets could be detected target positions, target classifications, agent states, and so on. These consensus approaches have been shown to guarantee convergence over many different dynamic network topologies [34, 36, 37].

Various methods have been proposed to accomplish this convergence among a fleet of autonomous agents. Some of them include Kalman-filtering approaches [27, 29] wherein agents communicate data asynchronously with their neighbors and update their information set using Kalman filtering techniques. These Kalman-filter-based approaches provide an adaptive mechanism to incorporate the varying uncertainty level of each agent's situational awareness. Agents communicate their information state ξ_i continuously until each vehicle converge to some nominal value $\xi^\star$. The information update equations presented in [29] are be written as

$$P_i[k+1] = [(P_i[k] + Q[k])^{-1} + \sum_{j=1}^{N_u} g_{ij}[k](P_j[k] + \Omega_{ij}[k])^{-1}]^{-1} \tag{2.5}$$

$$\xi_i[k+1] = \xi[k] + P_i[k+1] \cdot \sum_{j=1}^{N_u} [g_{ij}[k](\mu_j[k]P_j[k] + \Omega_{ij}[k])^{-1}(\xi_j[k] + \nu_{ij}[k+1] - \xi_i[k])]$$

where P_i is the covariance of the information state, $G[k]$ is the adjacency matrix ($g_{ij}[k] = 1$ if a link exists between agents i and j at instant k, and 0 otherwise), $\Omega_{ij}[k]$ is the expected value of the process noise $\nu_{ij}[k]$, and $\mu_j[k]$ is the outflow scaling factor, which is needed for unbiased consensus[29], such that

$$\mu_j[k] = \sum_{i=1, i\neq j}^{N_u} g_{ij}[k] \tag{2.6}$$

Other methods perform the information update using a weighted average of the current and received values [31, 34]

$$\xi_i[k+1] = W_{ii}[k+1]\xi_i[k] + \sum_{j=1}^{N_t} g_{ij}[k]W_{ij}[k+1]\xi_j[k] \tag{2.7}$$

where W_{ij} is the weighting factor for the information transmitted from agent j to i. This type of update allows for extensions to more generic types of consensus objectives. For instance, it is also possible to update the information state with the minimum or maximum value received, and so on.

In this thesis, the consensus idea is used to converge on the assignment value rather than the situational awareness. A maximum consensus strategy is implemented such that the current assignment will be overwritten if a higher value is received. By doing this, the network convergence properties found in the consensus algorithm literature can be exploited to converge on the assignment.

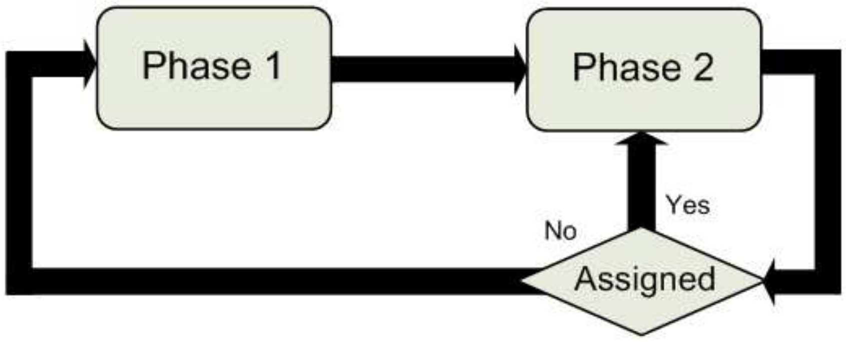


Figure 2-1: The CBAA iterates between two phases, the first is the auction process and the second is the consensus phase.

2.2 Algorithm Development

The Consensus-Based Auction Algorithm (CBAA) makes use of both auction and consensus algorithms to perform the optimization in (2.1). The algorithm consists of iterations between two phases (Figure 2-1). The first phase of the algorithm is the auction process, while the second is a consensus algorithm that is used to converge on a winning bids list. By iterating between the two, it will be shown that the CBAA can exploit the network flexibility and convergence rates of decentralized consensus algorithms, as well as the robustness and computational efficiency of the auction algorithms.

2.2.1 Phase 1: The Auction Process

The first phase of the algorithm is the auction process. Here, each agent places a bid on a task *asynchronously* with the rest of the fleet. By doing this, convergence is attainable much more quickly than in synchronous bidding strategies where the agents must wait for the rest of fleet to place a bid before moving to the next task. Let $c_{ij} > 0$ be the bid that agent i places for task j, and $H_i \in \{0,1\}^{N_t}$ the list of available tasks. Two vectors of length N_t that the agents will store and update throughout the assignment process will also defined. The first vector is $x_i(t)$, which is agent i's task list at time t, where $x_{ij}(t) = i$ if agent i has been assigned to task

j, and 0 if not. The second vector is the winning bids list $y_i(t)$, which is initialized as $y_{ij}(0) = 0$ for all j. This list will be further developed in section 2.2.2; but it can be assumed for now that y_{ij} is an up-to-date estimate of the highest bid made for each task thus far. A capability matrix $K \in \{0,1\}^{N_u \times N_t}$ will also be defined such that element $k_{ij} = 1$ if agent i is capable of performing task j, and 0 if not. Using the winning bids list and the capability matrix, the list of valid tasks $H_i(t)$ can be generated comparing the score achieved in completing a task with the associated value in the winning bids list:

$$H_i = (c_i > y_i) \wedge K_i \tag{2.8}$$

where $(a > b)$ returns a boolean vector whose j-th element is 1 if $a(j) > b(j)$ and zero otherwise, and $\wedge$ represents the element-wise boolean `and` operation.

The algorithm for the first phase is shown in Algorithm 1. At each iteration, an unassigned agent i selects a task J_i giving it the maximum score based on their current list of winning bids

$$J_i = \operatorname*{argmax}_j H_{ij} \cdot c_{ij} \tag{2.9}$$

If the agent has already been assigned a task, this selection process is skipped and the agent moves to phase 2. It is important to note that once an agent selects a task, that task is assigned for the remainder of the assignment period. Therefore, if equation (2.8) returns the zero vector ($H_{ij} = 0, \ \forall j$), then either the agent is incapable of performing the remaining tasks ($K_i = 0$) or all of the tasks have been assigned and cannot be outbid ($c_{ij} \leq y_{ij}, \ \forall j$); thus, the agent is not needed for this assignment. Otherwise, the agent will select a task and update its x_i and y_i vectors. If a tie occurs in finding J_i, an agent can select one of them either randomly or lexicographically based upon the task identifier.

2.2.2 Phase 2: The Consensus Process

The second phase of the CBAA is the consensus section of the algorithm. In general, auction algorithms compare task bids head-to-head and the agent with the highest

Algorithm 1 CBAA Phase 1 at time t:

```
1: $\forall i \in \{1, \dots, N_u\}$
2: procedure SELECT TASK($c_i, y_i(t-1), x_i(t-1)$)
3:     if $\sum_j x_{ij}(t-1) = 0$ then
4:         $H_i = (c_i > y_i(t-1)) \wedge K_i$
5:         $J_i = \text{argmax}_j \, H_{ij} c_{ij}$
6:         $x_{iJ_i}(t) = i$
7:         $y_{iJ_i}(t) = c_{iJ_i}$
8:     end if
9: end procedure
```

value is the winner. In order to perform the conflict resolution, auction approaches generally require a fully connected network or a connected network with routing in order to transmit bids to the auctioneer. In the CBAA, however, agents make use of a consensus strategy to converge on the list of winning bids, and use that list to determine the winner. This allows asynchronous bidding and conflict resolution over all tasks while not limiting the network to a specific structure.

Let $\mathbb{G}(t)$ be the undirected communication network at time t with symmetric adjacency matrix $G(t)$. The adjacency matrix is defined such that $g_{ik}(t) = 1$ if a link exists between agents i and k at time t, and 0 otherwise. Agents i and k are said to be *neighbors* if such a link exists. It is assumed that every node has a self-connected edge; in other words, $g_{ii}(t) = 1, \ \forall i$. This section also assumes that the channels are noiseless and transmitted messages are received one time step after they are sent. A time step in this case will be defined as a single unit of time in the simulation. Thus, if a message is passed over link g_{ik}, agent k will receive the message at the very next iteration. An iteration of the algorithm can also be defined as the execution of both phase 1 and phase 2 of the CBAA and takes one time step to perform. At each iteration of phase 2 of the algorithm, agent i receives the list of winning bids y_i from each of its neighbors. The consensus update for each task j in the list is then

$$y_{ij}(t) = \max_k g_{ik}(t) \cdot y_{kj}(t-1), \tag{2.10}$$

and the assignment for agent i can be updated using

$$z_{ij} = \underset{k}{\operatorname{argmax}}\, g_{ik}(t) \cdot y_{kj}(t-1)$$
$$x_{ij} = \begin{cases} 0 & \text{if } z_{ij} \neq i \\ x_{ij}(t) & \text{otherwise.} \end{cases} \tag{2.11}$$

/noindent Note that agent i's own list is included in the ones received ($g_{ii}(t) = 1$), and thus, the consensus phase will replace the information state y_{ij} with the largest value between agent i and its neighbors.

If an agent is outbid, it releases that task and goes back to the auction phase of the algorithm. Ties in determining z_{ij} cannot be resolved by random selection, since tie-breaking should be conducted coherently over the fleet. Two possible ways of breaking this tie are suggested: 1) intentionally inserting a small random number to the bid, or 2) tagging the transmission packet with the agent's identification number (AID) that indicates who sent the corresponding element y_{ij} values and breaking the tie with it.

Proposition 1. *The CBAA will provide the same assignment as the Sequential Greedy Selection Algorithm. This centralized algorithm recursively finds the highest score in the cost matrix and removes the associated row and column (agent and task) from the list of possible selections*

$$\begin{aligned} (i_n^\star, j_n^\star) &= \underset{(i,j)\in\mathcal{I}_n\times\mathcal{J}_n}{\operatorname{argmax}}\ c_{ij} \\ \mathcal{I}_{n+1} &= \mathcal{I}_n \setminus \{i_n^\star\} \\ \mathcal{J}_{n+1} &= \mathcal{J}_n \setminus \{j_n^\star\} \\ C_{n+1} &= C_n \circ E_{i_n^\star, j_n^\star} \end{aligned} \tag{2.12}$$

for $n \leq \min\{N_u, N_t\}$. The index sets and the cost matrix are initialized as $\mathcal{I}_1 = \{1, \ldots, N_u\}$, $\mathcal{J}_1 = \{1, \ldots, N_t\}$, and $C_1 = [c_{ij}] \in \mathbb{R}^{N_u \times N_t}$. $E_{i_n^\star, j_n^\star} \in \{0,1\}^{N_u \times N_t}$ has zeros in the entries of the i-th row and j-th column, and ones otherwise, while $\circ$

denotes the entry-wise product.

Proof. When the algorithm is initialized, each agent calculates the scores for each task. Of all the scores, there will be one agent $i_1^\star$ that has the highest score over the entire fleet for some task $j_1^\star$ (i.e. $(i_1^\star, j_1^\star) = \text{argmax}_{(i,j)}\, c_{ij}$). Since task selection of CBAA in (2.9) is greedy, the task $j_1^\star$ will be selected by agent $i_1^\star$. Moroever, agent $i_1^\star$ will never be outbid since its bid was the highest value in the fleet and conflict resolution between agents in (2.10) is a direct comparison. Thus, agent $i_1^\star$ will have its assignment from the first iteration of (2.12). Similarly, the next highest score in the matrix for an agent $i_2^\star \neq i_1^\star$ and task $j_2^\star \neq j_1^\star$, will be selected and won, since it is the next highest score available in the cost matrix. This process continues until each agent has an assignment which will provide the same results as the Centralized Sequential Greedy Selection Algorithm in (2.12). $\square$

Proposition 2. *At the termination of the CBAA assignment, the winning bid lists for all agents will have converged to the list of scores* $\{c_{i_1^\star j_1^\star}, \ldots, c_{i_m^\star j_m^\star}\}$ *with* $m = \min\{N_u, N_t\}$*, which can be generated by the centralized greedy recursion in (2.12), with appropriate re-indexing.*

Proof. Without loss of generality, assume that the agent index is such that $i_n^\star = n$. After the first iteration in (2.12), $i_1^\star$ will have selected task $j_1^\star$ and $y_{i_1^\star j_1^\star} = c_{i_1^\star j_1^\star}$. From Proposition 1, it is known that this value will never change since agent $i_1^\star$ cannot be outbid. After this value has been transmitted to each vehicle, every $y_{j_1^\star}$ will have reached a steady-state value of $c_{i_1^\star j_1^\star}$. Similarly, the second value from the recursion will set $y_{i_2^\star j_2^\star} = c_{i_2^\star j_2^\star}$ and eventually the winning bids list for each vehicle in the fleet will converge to $y = \{c_{i_1^\star j_1^\star},\ \ldots\ c_{i_m^\star j_m^\star}\}$, where $m = \min\{N_u, N_t\}$. For the case where agents are arbitrarily indexed, y will converge to an appropriately reordered list. $\square$

The above propositions give insight into the convergence of the algorithm which will be discussed in the next section. Note that if the network is fully connected (each agent can communicate directly with every other agent), then the order in which tasks will be assigned in Propositions 1 and 2 will be the same. As the distance between agents increases in the network, the conflict resolution time will increase, and the assignment order might change, however, the final solution will remain the same.

2.3 Convergence

The CBAA is considered to have converged to a solution when $m \triangleq \min\{N_u, N_t\}$ tasks have been assigned to an agent. This section will show that the proposed algorithm converges for static networks by presenting a finite upper-bound of its convergence time. It will also suggest the condition under which the algorithm will converge for dynamic networks. The convergence analysis in this section was performed jointly with Choi [89].

2.3.1 Worst-Case Convergence Bound

Suppose that a static network is connected; thus, there exists a (undirected) shortest path length $d_{ik} < \infty$ for every pair of agents i and k. Assuming every edge length is unity, the network diameter D then becomes

$$D = \max_{i,k} d_{ik}. \tag{2.13}$$

The convergence time T_C is defined as

$$T_C \triangleq \min t \in \left\{ t \in \mathbb{Z}_+ : \sum_{i=1}^{N_u} x_{ij}(t) = 1, \quad \sum_{j=1}^{N_t} \sum_{i=1}^{N_u} x_{ij}(t) = m \right\}. \tag{2.14}$$

Proposition 3. *The convergence time of the consensus-based auction algorithm over a connected fixed network with diameter D is upper-bounded by:*

$$T_C \leq \bar{T}_1 \triangleq D \cdot m. \tag{2.15}$$

Proof. From the recursion for the centralized assignment (2.12), define a list $L^\star$ with components $\left\{(i_1^\star, j_1^\star), \cdots, (i_m^\star, j_m^\star)\right\}$ where $m = \min\{N_t, N_t\}$ such that

$$(i_n^\star, j_n^\star) = \operatorname*{argmax}_{(i,j)\in\mathcal{I}_n\times\mathcal{J}_n} c_{ij} \tag{2.16}$$

which is the list of highest valued tasks not previously assigned. After at most D iterations, information from each agent's first selection will have been received by every other agent and at least $(i_1^\star, j_1^\star)$ will be assigned. Another maximum of D iterations will produce the $(i_2^\star, j_2^\star)$ assignment. This process is repeated m times until the entire list is complete. Since the assigned agent and task at each iteration are removed from the selection lists, either the available agent set or the task set becomes empty, which means completion of the full assignment. Thus, the convergence time of the CBAA, T_C, for a fixed network is upper bounded by $D \cdot m$. $\square$

Proposition 3 ensures finite-time convergence of the proposed CBAA over any static connected network regardless of the scoring matrix C. Note that $\bar{T}_1$ is not an attainable bound in general, since in the worst case analysis, once a conflict has passed the length of the diameter, one of those agents will win the bid and that path will never be used again for conflict resolution. In other words, a conflict in the worst case can only cross each d_{ik} a maximum of one time. Thus, a tighter bound can be found using the m longest d_{ik} values:

$$\begin{aligned} \bar{T}_2 \triangleq \max & \sum_{i<N_u}\sum_{k>i} d_{ik} z_{ik} \\ \text{subject to } & \sum_{i<N_u}\sum_{k>i} z_{ik} = m = \min\{N_u, N_t\} \\ & \mathbf{z} \in \{0,1\}^{N_u(N_u-1)/2}. \end{aligned} \tag{2.17}$$

It is obvious that $\bar{T}_2 \leq \bar{T}_1$ since $d_{ij} \leq D$. $\bar{T}_2$ is a tighter bound than $\bar{T}_1$ and easily computed once a network topology is given. Like $\bar{T}_1$, this bound might still not be attainable since the conflicts must happen in order along the network for the worst case convergence to happen. The attainable worst-case convergence time for the case $N_t \leq N_u$ can be computed from the following proposition:

Proposition 4. *For a given static graph with N_u nodes, the worst-case convergence time of the CBAA to reach a conflict-free assignment can be solved with the following 0-1 Linear Program*

$$\bar{T}_3 = \max_{z_{ij}^k} \sum_{k=1}^{N_t} d_{ij} z_{ij}^k \tag{2.18}$$

subject to

$$\sum_{k=1}^{N_t} z_{ij}^k \leq 1, \quad \forall i \neq j \in [1, N_u] \cap \mathbb{Z} \tag{2.19}$$

$$\sum_{ij:i\neq j} z_{ij}^k = 1, \quad \forall k \in [1, N_t] \cap \mathbb{Z} \tag{2.20}$$

$$\sum_{i\neq j} z_{ij}^k - \sum_{m\neq j} z_{jm}^{k+1} = 0, \quad \forall j < N_u, \ \forall k < N_u \tag{2.21}$$

$$z_{ij}^k \in \{0, 1\}. \tag{2.22}$$

Proof. First, note that if the constraints in (2.21) are relaxed, the optimization (2.18) - (2.22) will give the same solution value as $\bar{T}_2$, since it selects the m longest edges while avoiding multiple selections. Therefore, $\bar{T}_3 \leq \bar{T}_2$. In the worst case, a maximum of m conflicts should be resolved sequentially, and this happens when agent i_2^*, in recursion (2.16), realizes that it is outbid by i_1^*. This means that i_2^* should be one of the agents that is separated from i_1^* by D. Likewise, in the worst case situation, i_{n+1}^* should be one of the agents that were outbid by i_n^* on task j_n^*. This relation corresponds to the set of constraints in (2.21). □

Remark 1. *For dynamic networks in which $G(t)$ varies with time, convergence of CBAA procedure is guaranteed if there exists $\tau < \infty$ such that*

$$\mathbb{W}(t) = \mathbb{G}(t) \cup \mathbb{G}(t+1) \cup \cdots \cup \mathbb{G}(t+\tau-1) \tag{2.23}$$

is connected $\forall t$. Moreover, the convergence time is upper bounded by $\tau \cdot \min\{N_u, N_t\}$, since any information about confliction is transmitted within τ. □

2.3.2 Probabilistic Bound for Static Networks

Although the upper bound on convergence shows that the algorithm is guaranteed to converge in finite time, the actual worst case scenarios are very unlikely to happen. For instance, the conflicts must arise in such a way that they never happen at the same time, are always at least located at the end of the longest remaining path, and after each conflict resolution, another conflict begins afterward with one of the previously conflicted agents. In practice, it would make more sense to look at the expected value and the probabilistic deviation of the convergence time instead of the worst-case value. The probabilistic bound of the convergence time will be defined as

$$\bar{T}_P = \widehat{T}_C + \kappa\sigma(T_C). \tag{2.24}$$

where $\widehat{T}_C$ and $\sigma(T_C)$ denote the estimates of the expected value and the standard deviation of the convergence time respectively. $\kappa > 0$ indicates the confidence level; for instance, if the convergence time is normally distributed and $\widehat{T}_C$ is the true expected convergence time, $\kappa = 2.0$ ensures that the convergence time will be less than $\bar{T}_P$ with probability 97.5%.

This work assumes that the average convergence time can be well approximated with the following form:

$$\widehat{T}_C = \widehat{D} \cdot \widehat{m}_e, \tag{2.25}$$

inspired by the expression of $\bar{T}_1 = D \cdot m$. The degree of validity of this assumption depends on the actual cost map, but it will be verified that this approach provides a sufficient estimate of the actual convergence time.

One reasonable choice of $\widehat{D}$ is the average distance between two agents D_{avg}, computed by

$$D_{avg} = \frac{1}{N_u(N_u - 1)} \sum_{i=1}^{N_u} \sum_{j=1}^{N_u} d_{ij}. \tag{2.26}$$

Suppose that there exists a total of $\widehat{m}_e$ tasks in conflict at time 0, and also that one conflict is resolved on average every D_{avg} time steps; then, it will take $D_{avg} \cdot \widehat{m}_e$ time steps until all the conflicts are resolved. In this reasoning, the average number of initial conflicts is the same as the number of conflicts that is resolved sequentially. Thus, this thesis estimates $\widehat{m}_e$ by quantifying the average number of initial conflicts over the fleet. Assuming no prior information about the cost map, the average number of conflicts over the fleet with size N_u can be expressed as

$$m_{avg} = \sum_{k=2}^{N_u} \eta_k \binom{N_u}{k} \frac{1}{N_t^{k-1}} \left(1 - \frac{1}{N_t}\right)^{N_u - k}. \tag{2.27}$$

where $\eta_k = 1 + 1/N_t^{k-2}$. The index k denotes the number of agents in conflict on a single task, while η_k represents the number of equivalent pair-wise conflicts for a single k-tuple-wise conflict. For instance, if 3 agents are in conflict on a single task, on average $1/N_t$ additional conflicts should be resolved after the initial conflict has been cleared. The binomial expression $\binom{N_u}{k}$ represents the total number of k-tuples, and the remainder of the summation shows the probability that a given k-tuple can be in conflict over N_t tasks. With D_{avg} and m_{avg}, the estimate of the expected convergence time for a static graph is written as

$$\widehat{T}_C = D_{avg} \cdot m_{avg}. \tag{2.28}$$

The standard deviation of the convergence time can be derived by figuring out the standard deviation of the distance between agents and of the number of conflicts to resolve. The standard deviation of the distance can be empirically computed as

$$\sigma_d = \sqrt{\frac{\sum_{ij:i \neq j} d_{ij}^2}{N_u(N_u - 1)} - D_{avg}^2}. \tag{2.29}$$

Regarding the variance of the number of conflicts to resolve, it is exploited that the number of effective conflicts m_e is bounded below by zero and bounded above by N_t. Considering the possible skewness of the distribution of m_e, one estimate of the standard deviation is

$$\sigma_m = \frac{\min\{m_{avg}, N_t - m_{avg}\}}{2} \tag{2.30}$$

where the factor 2 represents the 95% confidence interval for a normal distribution.

Assuming that the distance between agents is independent of the number of conflicts to resolve, the variance of the convergence time becomes

$$\sigma(T_C) = \sqrt{\sigma_d^2\sigma_m^2 + \sigma_d^2 m_{avg}^2 + \sigma_m^2 D_{avg}^2}. \tag{2.31}$$

2.3.3 Convergence with Inconsistent Information

For simplicity, assume that the agent states are perfectly known and that the only sources of error are in the task states. Assume p_{ij} is some parameter of task j (*e.g. location of target*) estimated by agent i that is used to calculate $\bar{c}_{ih}$ such that $\bar{c}_{ij} = f(p_{ij})$. Let $\bar{C}$ be the scoring matrix containing the local scores

$$\bar{C} = \begin{bmatrix} \bar{c}_{11} & \bar{c}_{12} & \cdots \\ \bar{c}_{21} & \bar{c}_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{2.32}$$

Using proposition 1, and interchanging C with $\bar{C}$, the same analysis can be done to arrive at the final CBAA assignment. Thus, even with a scoring matrix based on inconsistent data, the algorithm is still guaranteed to converge to a conflict free solution. Inconsistencies do however affect the optimality of the final solution and the trade-off becomes initial consensus on the situation awareness versus the overall value of the assignment. The same argument extends to the case where the agent states are also unknown as well.

2.4 Performance

This section will analyze the performance of the CBAA solution against the optimal solution. Due to Proposition 1, the solution to CBAA is the same as that to Sequential Greedy Selection Algorithm. It will be shown that the objective value for the sequential greedy solution, and thus the CBAA solution, is at least 50% of the optimal objective value. A more practical expected performance level of CBAA will also be derived for some important abstractions. Performance bounds in this section were contributed by Choi [89].

2.4.1 Worst-Case Bound

The actual performance of the CBAA compared to the optimal solution depends on what the scoring matrix looks like. The worst-case bound considers the worst possible performance of CBAA over all score matrix $C \in \mathbb{R}_{+}^{N_u \times N_t}$.

Proposition 5. *CBAA guarantees 50% optimality:*

$$\mathbf{OPT} \leq 2 \cdot \mathbf{CBAA} \tag{2.33}$$

where **OPT** *and* **CBAA** *are the objective values from the optimal solution and the CBAA solution, respectively.*

Proof. First, note that the solution to the CBAA is the same as the centralized greedy assignment as in Proposition 1. Thus, if a performance bound can be found with the the centralized greedy assignment algorithm, then the same bound can be used for the CBAA. Without loss of generality, assume that the centralized greedy method assigns tasks in such a way that:

$$x_{ij} = \begin{cases} \delta_{ij}, & \text{if } \max\{i,j\} \leq m \\ 0, & \text{if } \max\{i,j\} > m \end{cases} \tag{2.34}$$

where $m \triangleq \min\{N_u, N_t\}$ and δ_{ij} is the Kronecker delta, and

$$c_{ii} > c_{jj}, \text{ if } i > j. \tag{2.35}$$

In other words, agent i is assigned task i that provides greater local reward than agent j who is assigned task j. This assumption does not reduce generality, since the agent and task indexing can always be reordered to satisfy the above conditions. With this, the objective value for the centralized greedy, or equivalently CBAA, solution is

$$\mathbf{CBAA} = \sum_i c_{ii}. \tag{2.36}$$

Because each agent selects its task in a greedy way given the selections of its precedents, the following inequalities should be satisfied for the greedy solution:

$$\begin{aligned} c_{ii} &> c_{ij}, \ \forall i, \ \forall j > i \\ c_{ii} &> c_{ji}, \ \forall i, \ \forall j > i. \end{aligned} \tag{2.37}$$

Notice that the case in which the greedy selection is the farthest from the optimal solution, the scoring matrix will look like

$$\begin{aligned} c_{ij} &= c_{ii} - \epsilon, \quad \forall i, \ \forall j > i \\ c_{ji} &= c_{ii} - \epsilon, \quad \forall i, \ \forall j > i, \end{aligned} \tag{2.38}$$

with a very small ϵ that satisfies

$$\epsilon < \min_{i<m} c_{ii} - c_{i+1,i+1}. \tag{2.39}$$

Now let each agent try to improve its reward by selecting another task. Because the assignment should be conflict-free, if one agent selects a task, then the agent which originally took that task should select another one. Because of (2.37), an agent can improve its individual reward by choosing a task with a lower index, while (2.38) leads to an ϵ decrease of the individual reward for the agent who selects the higher-indexed task. Note that the greatest performance enhancement is accomplished by the following strategy:

$$j_i^\star = \begin{cases} m - i + 1, & \text{if } i \in [1, m] \\ \emptyset, & \text{otherwise}, \end{cases} \tag{2.40}$$

which leads agent i with $i < \lfloor m/2 \rfloor$ to select a higher indexed task, and agent i with

$i \in [\lceil m/2 \rceil + 1, m]$ to select a lower indexed task. This gives the objective value:

$$\begin{aligned} \mathbf{OPT} &= \sum_{i=1}^{\lfloor m/2 \rfloor} (c_{ii} - \epsilon) + \sum_{i=\lfloor m/2 \rfloor+1}^{\lceil m/2 \rceil} c_{ii} + \sum_{i=\lceil m/2 \rceil+1}^{m} (c_{(m-i+1),(m-i+1)} - \epsilon) \\ &= 2 \times \sum_{i=1}^{\lfloor m/2 \rfloor} (c_{ii} - \epsilon) + \sum_{i=\lfloor m/2 \rfloor+1}^{\lceil m/2 \rceil} c_{ii} \\ &\leq 2 \times \sum_{i=1}^{m} c_{ii} = 2 \cdot \mathbf{CBAA}. \end{aligned} \tag{2.41}$$

Thus, 50% optimality is guaranteed for the CBAA. □

Remark 2. *The 50% performance lower bound is not a weak bound since it can be achieved in the following example:*

$$C = \begin{bmatrix} 1 & 1-\epsilon \\ 1-\epsilon & \epsilon \end{bmatrix}. \tag{2.42}$$

with small ϵ. The CBAA solution for which agent k selects task k for $k = 1, 2$ results in the objective value of $1+\epsilon$, while the optimal solution for which agent k selects task $3-k$ gives the objective value of $2-\epsilon$. For an infinitesimal ϵ, $\mathbf{OPT} = 2 \cdot \mathbf{CBAA}$.

2.4.2 Expected Performance

Proposition 5 suggests that the CBAA solution guarantees at least 50% optimality regardless of the scoring matrix, and this bound is the tightest bound unless further information about the scoring matrix is provided. In spite of the importance of this worst-case bound, a more practical value might be the average performance of the CBAA. This section analytically derives the upper bound of the performance gap, ($\mathbf{OPT}$ – $\mathbf{CBAA}$)/$\mathbf{OPT}$, for two abstract settings. The first considers the case in which each element of the scoring matrix is i.i.d. with uniform distribution, while the second deals with a more practical situation where agents and targets are randomly distributed over a two-dimensional space with the objective of the assignment to maximize the sum of some time-discounted reward.

The key principle in deriving the expected optimality gap is the following Proposition:

Proposition 6. *If each element of the scoring matrix is distributed i.i.d,*

$$\mathbb{E}[\textbf{Sequence}] \leq \mathbb{E}[\textbf{CBAA}] \leq \mathbb{E}[\textbf{OPT}] \leq \mathbf{E}[\textbf{InfOpt}]. \tag{2.43}$$

where **Sequence** *and* **InfOpt** *denote the solution based on the following strategies:*
Sequence: agent precedence is defined first; the lower-indexed agent (not reordered index) greedily selects a task and then the next highest indexed agent greedily selects while taking conflicts into account.
InfOpt: each agent selects greedily without considering conflict resolution.

In case every score value is i.i.d, the performance of Sequence *cannot be better than that of CBAA on average.* **Sequence** *should not depend on specific precedence order, and thus it represents the performance of a randomly-ordered greedy selection process, which has no reason to be better than CBAA. Also, since* InfOPT *considers a relaxation of the original problem, it should provide an upper bound of the optimal solution.*

Thus, an upperbound of the optimality gap can be obtained as follows:

$$\mathcal{E} \triangleq \frac{\mathbb{E}[\textbf{OPT}] - \mathbb{E}[\textbf{CBAA}]}{\mathbb{E}[\textbf{OPT}]} \leq \frac{\mathbb{E}[\textbf{InfOpt}] - \mathbb{E}[\textbf{Sequence}]}{\mathbb{E}[\textbf{InfOpt}]}, \tag{2.44}$$

once $\mathbb{E}[\textbf{InfOpt}]$ and $\mathbb{E}[\textbf{Sequence}]$ are computed.

Uniform scoring matrix

In the case that $c_{ij} \sim \mathcal{U}[0, c_{\max}],\ \forall(i,j)$, the lower and upper bound of the expected performance of the CBAA solution can be obtained in a closed form. The upper and lower bounding expectation values can be obtained by using the order statistics. First, consider the **InfOpt** case. Each agent selects the largest entry from N_t realizations of i.i.d samples. Thus, the distribution of each agent's selection corresponds to N_t-th order statistics of $\mathcal{U}[0, c_{\max}]$. It is known that the k-th order statistics (k-th smallest one) from n samples taken from $\mathcal{U}[0,1]$ has the Beta distribution[90]:

$$U_{(k)} \sim \text{Beta}(k, n+1-k) \tag{2.45}$$

and its mean is

$$\mathbb{E}[U_{(k)}] = \frac{k}{n+1}. \tag{2.46}$$

In the **InfOpt** case, each agent independently selects the N_t-th order statistics from its list of rewards. Thus, the expected performance is simply the sum of each agent's expected performance:

$$\mathbb{E}[\textbf{InfOpt}] = m\mathbb{E}[c_{\max} U_{(N_t)}] = mc_{\max}\frac{N_t}{N_t+1} = mc_{\max}\left[1 - \frac{1}{N_t+1}\right] \tag{2.47}$$

where $m = \min\{N_u, N_t\}$.

On the other hand, in the **Sequence** case, the expected performance of i-th agent is the expected value of the largest order statistics out of $(N_t - i + 1)$ samples. Therefore,

$$\begin{aligned}\mathbb{E}[\textbf{Sequence}] &= c_{\max}\sum_{i=1}^{m}\frac{N_t - i + 1}{N_t - i + 2}\\ &= c_{\max}\sum_{i=1}^{m}\left(1 - \frac{1}{N_t - i + 2}\right)\\ &= mc_{\max} - c_{\max}\sum_{i=1}^{m}(N_t - i + 2)^{-1}.\end{aligned} \tag{2.48}$$

Thus,

$$1 - \frac{1}{m}\sum_{i=1}^{m}(N_t - i + 2)^{-1} \leq \frac{\mathbb{E}[\textbf{CBAA}]}{mc_{\max}} \leq \frac{\mathbb{E}[\textbf{OPT}]}{mc_{\max}} \leq 1 - \frac{1}{N_t+1}, \tag{2.49}$$

which leads to

$$\begin{aligned}
\frac{\mathbb{E}[\mathbf{OPT}] - \mathbb{E}[\mathbf{CBAA}]}{\mathbb{E}[\mathbf{OPT}]} &\leq 1 - \frac{1 - \frac{1}{m}\sum_{i=1}^{m}(N_t - i + 2)^{-1}}{1 - \frac{1}{N_t+1}} \\
&= \frac{1}{N_t} + \frac{1}{m}\left(1 + \frac{1}{N_t}\right)\sum_{i=1}^{m}(N_t - i + 2)^{-1} \qquad (2.50) \\
&\triangleq \mathcal{E}_u. \qquad (2.51)
\end{aligned}$$

Figure (2-2) plots $\mathcal{E}_u$ when $N_t = m$ for different N_t values. It is found that the expected optimality gap of the CBAA will never exceed 15%. Also, in the case that $N_t \gg 1$, $\mathcal{E}_u$ can be approximated as

$$\mathcal{E}_u \approx \frac{1 + \log m}{m}. \qquad (2.52)$$

Numerical experiments in section 2.6.1 will validate the use of $\mathcal{E}_u$ as an indication of the average performance of the CBAA.

Uniformly distributed agents and targets

Consider the situation where N_u agents and N_t targets are uniformly distributed over a two-dimensional space $[0, L] \times [0, L]$, and the goal is to assign agents to target to maximize the sum of each agent's time-discounted reward. In this case, the distribution of each element in the scoring matrix is i.i.d, thus, a similar analysis as the previous section can be done to derive the performance bound of the CBAA in this setup. The first step is to derive the probability density of each entry of the scoring matrix:

$$c_{ij} = c_0 \exp(-r_{ij}/r_0) \qquad (2.53)$$

where C_0 and r_0 are constants.

If the probability density function (pdf) of r_{ij} is known, the pdf of c_{ij} can be

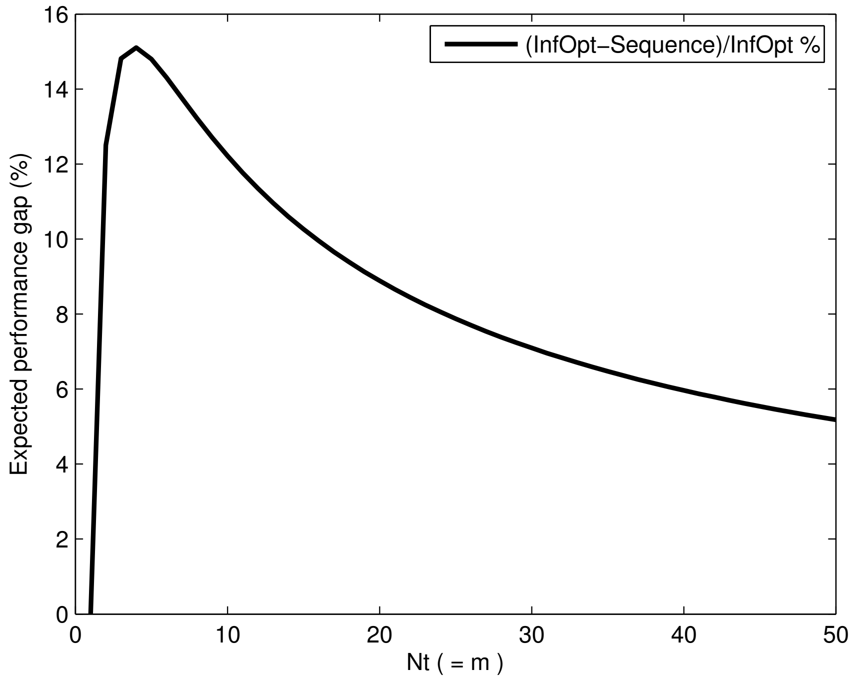


Figure 2-2: Optimality Gap for the uniformly distributed c_{ij}.

expressed as

$$f_C(c) = \begin{cases} \frac{r_0}{c} f_R\left(-r_0 \log(c/c_0)\right), & \text{if } c \in [c_0 e^{-\sqrt{2}L/r_0}, c_0], \\ 0, & \text{otherwise,} \end{cases} \tag{2.54}$$

where $f_R(r)$ is the pdf of the distance between an agent and target (indices are omitted to avoid confusion). Since the positions of an agent and target have uniform distribution, the coordinate difference of x and y have the triangular distribution:

$$f_X(x) = \begin{cases} \frac{1}{L}\left(1 + \frac{x}{L}\right), & \text{if } x \in [-L, 0] \\ \frac{1}{L}\left(1 - \frac{x}{L}\right), & \text{if } x \in [0, L] \\ 0, & \text{otherwise,} \end{cases} \tag{2.55}$$

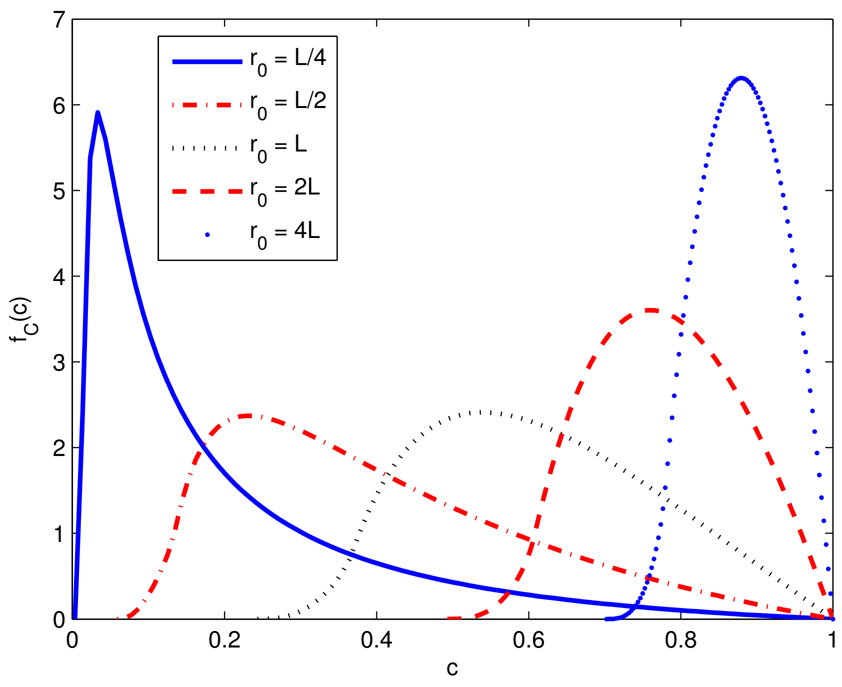


Figure 2-3: Probability density for the score value for the case with uniformly deployed agents and targets

and $f_Y(y)$ has the same form. The pdf of r is related to the pdfs of x and y as follows:

$$f_R(r) = \int_0^{2\pi} f_X(r\cos\theta) f_Y(r\sin\theta) d\theta. \tag{2.56}$$

It can be shown that $f_R(r)$ can be derived as a closed form:

$$f_R(r) = \begin{cases} \frac{r}{L^2}\left[2\pi - \frac{8r}{L} + \frac{2r^2}{L^2}\right], & \text{if } r \in [0, L] \\ \frac{r}{L^2}\left[4(\sin^{-1}\frac{L}{r} - \cos^{-1}\frac{L}{r}) + 8(\sqrt{\frac{r^2}{L^2} - 1} - 1) + (4 - 2\frac{r^2}{L^2})\right], & \text{if } r \in [L, \sqrt{2}L] \\ 0, & \text{otherwise.} \end{cases} \tag{2.57}$$

Figure 2-3 depicts $f_C(c)$ with $c_0 = 1$ for five different values of r_0: $L/4, L/2, L, 2L, 4L$.

Given $f_C(c)$, the k-th order statistics out of i.i.d samples of size N_t is represented

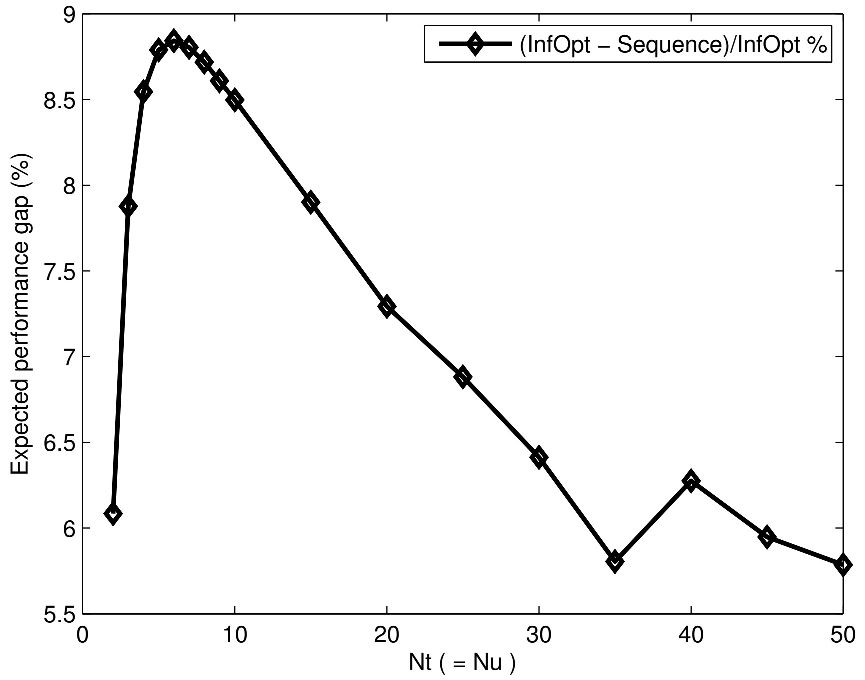


Figure 2-4: Upper bound of the optimality gap for the case with uniformly distributed agents and targets in two-dimensional space

as

$$f_{C(k)}(c) = N_t \binom{N_t - 1}{k-1} F_C(c)^{k-1}(1 - F_C(c))^{N_t - k} f_C(c). \tag{2.58}$$

Figure 2-4 illustrates the upper bound of the optimality gap, $1 - \mathbb{E}[\textbf{Sequence}]/\mathbb{E}[\textbf{InfOpt}]$, in the case that $r_0 = L$ for different values of $n = N_t = N_u$. It can be seen that the performance of the CBAA provides less than a 9% optimality gap, and that the gap becomes smaller as the problem size increases. Numerical simulations in Appendix A will confirm that the actual optimality gap $(\textbf{OPT} - \textbf{CBAA})/\textbf{OPT}$ is within the predicted upper-bound.

2.5 Convergence Criteria for Dynamic Scoring

The CBAA was presented using a static scoring system. This means that a snapshot of the environment is taken at the start of the assignment period, and scores are calculated under the assumption that they will remain fixed for its duration. In a dynamic environment, however, by the time a conflict has been resolved the actual task scores might be much different than the ones calculated at the beginning of the assignment period. In this case, assuming a fixed scoring matrix might cause the agent to select a task that is no longer the best choice. By ignoring these effects, the performance of the mission will be subject to increasing degradation as task values become outdated in the mission.

Consider the impact of updating the scoring matrix C with new information during the assignment. From (2.12) we know that the agent with the highest possible bid (the maximum $c_{ij} \in C$) will be assigned its desired assignment. However, if the scoring matrix is not fixed, no guarantee can be made that the current highest value $c_{ij}(t_0)$ will still be the highest value at some time $t_1 > t_0$. This might cause churning as the agents continuously outbid each other as fluctuations occur in the scores, significantly extending the convergence time. In some extreme cases, the algorithm may not converge at all. There is thus a need to incorporate a scoring mechanism that can handle this dynamic update.

Proposition 7. *The CBAA will converge to a conflict free assignment while utilizing a dynamically updated scoring matrix provided that following criteria in the scoring mechanism are met:*

$$\textit{i)}\ c_{ij}(t_1) > c_{kj}(t_1) \Longrightarrow c_{ij}(t_2) > c_{kj}(t_2), \quad \forall\, t_2 > t_1, \quad \forall k, \tag{2.59}$$

$$\textit{ii)}\ c_{ij}(t_1) \geq c_{ij}(t_2), \quad \forall\, t_2 > t_1 \tag{2.60}$$

Proof. The first criterion states that if an agent can outbid another at any time t_1, then it can always do so, while the second states that an agent's bid must be monotonically non-increasing. At some time t_1, the highest possible single assignment score $\max_{(i,j)} c_{ij}(t_1)$ and corresponding assignment pair $(i^\star, j^\star)$ can be found. With (2.60) satisfied, the bid's score is the highest it will ever be and it follows from (2.59) that the bidding agent will not be outbid at any time $t_2 > t_1$. Thus, that agent and task are now assigned and the row and column $(i^\star, j^\star)$ can be removed from the scoring matrix. This process can be continued as in (2.12) to create a list of highest valued assignments. From here, the proof of convergence in proposition 3 follows. It should be noted that if (2.1) was changed to a minimization problem, (2.59) and (2.60) would need to be reversed accordingly. □

With these clear criteria for convergence, the problem is now shifted to finding a scoring scheme that can fit this framework. In the case where there is no such scoring formulation, the static algorithm can still be used and a conflict-free assignment will still be obtained. It should be noted that the effects of a dynamic environment will only begin to affect the assignment over long convergence times. This is generally found in areas with sparse communication where agents have to travel large distance to resolve any conflicts. In most situations the expected convergence time is fast (about 2–3 time steps) and the static approximation will be valid.

2.6 Results

Numerical results are obtained to demonstrate the many different properties of the CBAA algorithm. For comparison, three different algorithms are implemented to act as benchmarks. The first is the implicit coordination algorithm [43], which given perfect SA over the fleet, solves the optimal assignment problem (2.1). Although this approach can obtain an optimal solution, the amount of communication required to converge on the SA can be very large, and the computational load used to solve the optimization for the entire fleet can be very large as the number of tasks and agents increases. Thus, the implicit coordination algorithm is often impractical for implementation. The second algorithm, which will be called the Greedy Based Auction Algorithm (GBAA), is similar to the CBAA but does not make use of the consensus phase. Agents greedily select tasks and resolve conflicts with their direct neighbors only. This algorithm requires slightly less communication resource per time step than the CBAA but will converge much more slowly under communication range constraints. The third algorithm is the ETSP algorithm developed in [60]. It removes the auctioneer similarly to CBAA and has a novel message passing system to diminish the number of conflicts. However, it assumes perfect knowledge of the task locations for all agents.

2.6.1 Optimality and Convergence

Monte Carlo (MC) simulations were used to analyze the performance of the CBAA static assignment algorithm. The number of tasks and agents were each incremented from 1 to 40, while 500 randomized simulations were performed in each case. For each simulation, random networks were created by forming a random spanning tree (RST) over the fleet of agents using [91], and then adding varying amounts of random links to the network. It was assumed that a very large and randomized simulation set would provide a reasonable distribution over all possible networks. Values of c_{ij} were randomly sampled with uniform distribution over $[0, 100]$. The benchmark used for comparison was the implicit coordination algorithm [20].

Figure 2-5 shows the results of the simulations with the x and y axis defined by N_t and N_u, respectively. The z axis indicates the percent deviation from the optimal solution. These results show that the deviation is maximum when the number of agents and tasks are the same ($N_t = N_u$), and in the worst case it deviates less than 6% from the optimal solution. Figure 2-6 shows the diagonal of this plot indicating the worst-case performance. Notice that the actual optimality gap computed by simulations is upper bounded by the analytical prediction in Figure 2-2. As the fleet grows past approximately 10 agents, the performance deviation slowly decreases until it flattens out near the 35 agent mark. Convergence results were obtained for the same simulations and can be found in Figure 2-7. The worst-case values are once again located along the diagonal, and flatten out as the number of tasks and agents became large.

2.6.2 Probabilistic Bounds

Monte-Carlo simulations were used to validate the probabilistic upper bound of the convergence time, $\bar{T}_P \equiv \widehat{T}_C + \sigma(T_C)$. Figure 2-8 shows how the probability of the actual convergence time exceeding $\bar{T}_P$ changes as the network size (increasing number

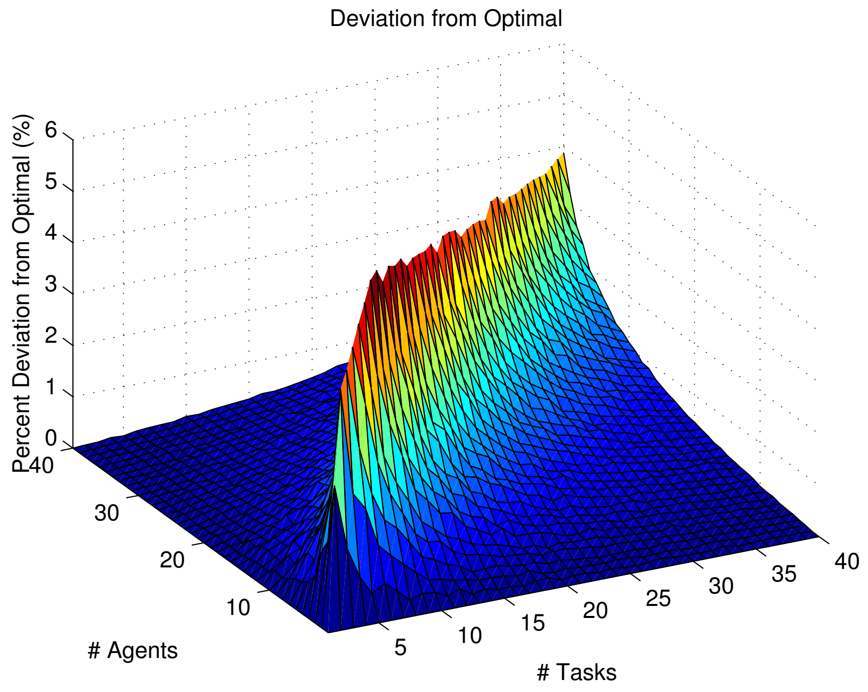


Figure 2-5: The deviation of the CBAA algorithm from the optimal solution shows a bound of less than 6%

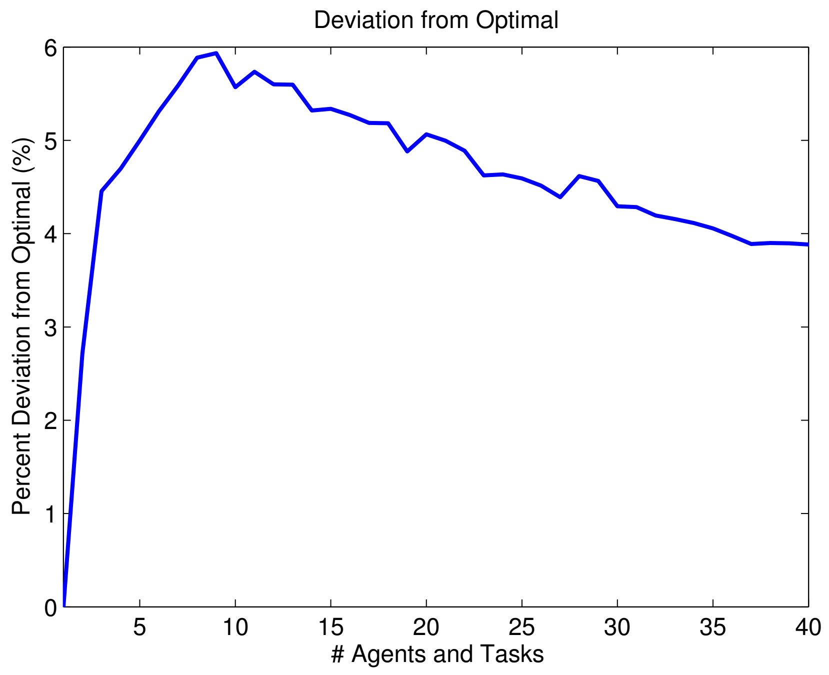


Figure 2-6: Diagonal slice of the CBAA performance graph representing the worst-cases of the values

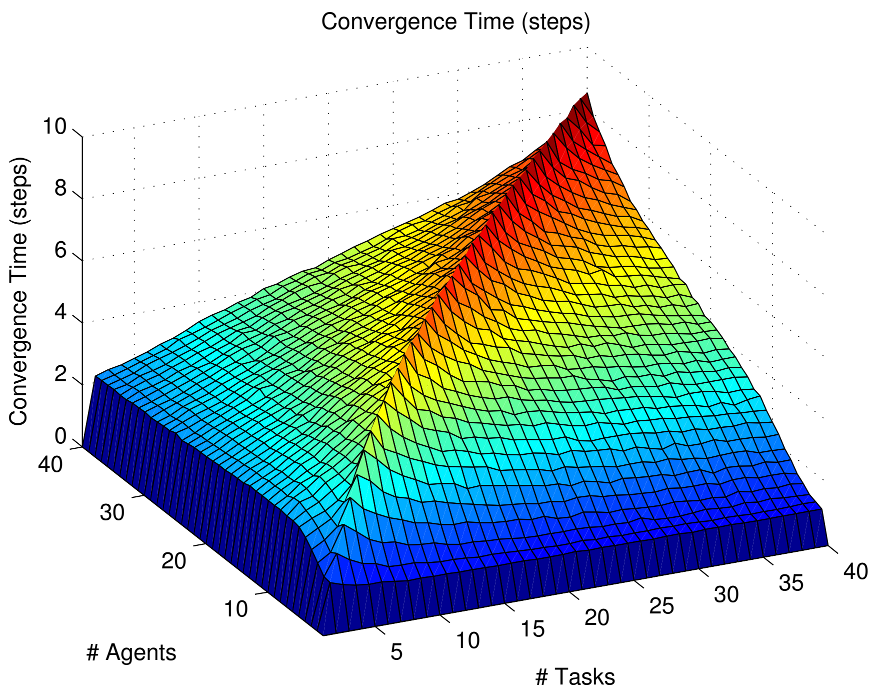


Figure 2-7: The convergence of the CBAA algorithm shows that the worst case values are located along the diagonal and flattens as the number of agents and tasks is increased

of agents). It is found that this probability decreases as the size of the network grows; in the case where there are more than 10 agents, $\bar{T}_P$ can be regarded as an effective upper-bound of the convergence time. Figure 2-9 depicts the average values of the deterministic bound $\bar{T}_2$ and the probabilistic bound $\bar{T}_P$ ($\kappa = 2$) with respect to the size of the network. It should be noted that the probabilistic bound is much less conservative than the deterministic worst-case bound, although in the simulations it is effectively an upper-bound on the actual convergence time for $N_u > 10$.

2.6.3 Dynamic Environments

To simulate a dynamic environment, tasks and agents were randomly placed in a gridded area 2000m × 2000m in size. Tasks were held stationary while agents were able to move with a predefined constant velocity. The task positions were known to the agents *a priori* and their scores were based on the estimated time of arrival. The communication range of the vehicles was incrementally increased while the mission

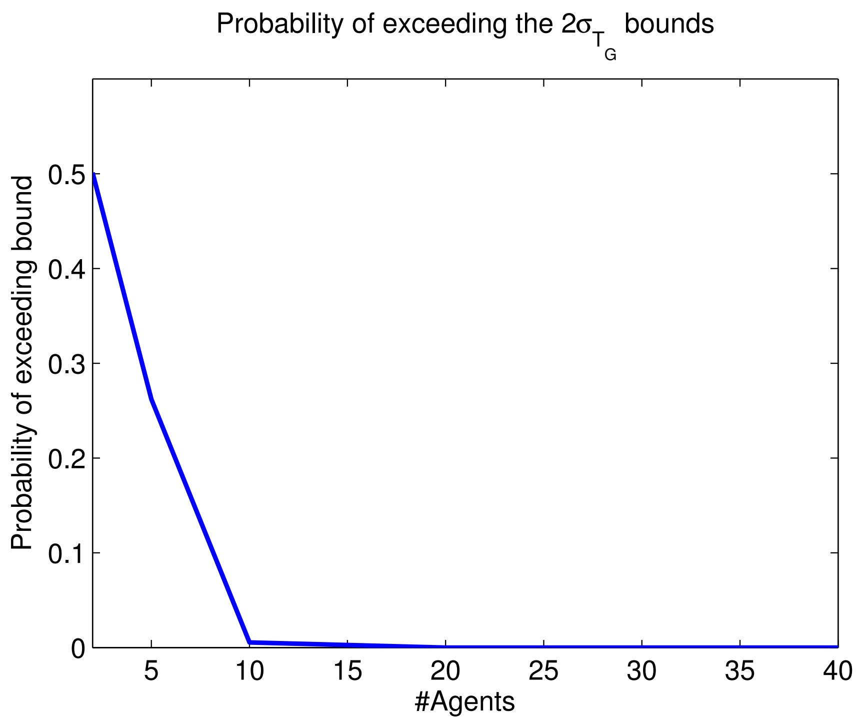


Figure 2-8: Probability of the actual convergence time exceeding $\bar{T}_P$

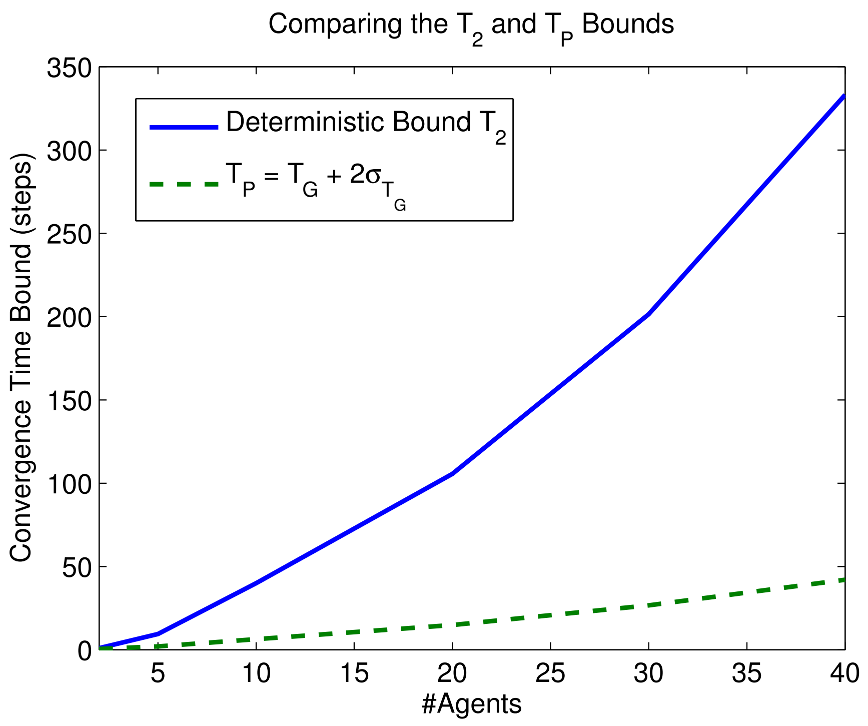


Figure 2-9: Comparison of $\bar{T}_P$ and $\bar{T}_2$

objective in (2.61) was used, where T_{ij} is the arrival time for agent i at task j. For these simulations, $N_u = N_t$ and 100 MC simulations were performed for each data point.

$$\begin{aligned}
&\min \sum_{i=1}^{N_u} \sum_{j=1}^{N_t} T_{ij} x_{ij} \\
&\text{subject to} \quad \forall i = \{1, \ldots, N_u\} : \sum_{j=1}^{N_t} x_{ij} \leq 1 \\
&\qquad\qquad\quad \forall j = \{1, \ldots, N_t\} : \sum_{i=1}^{N_u} x_{ij} \leq 1 \\
&\qquad\qquad\quad x_{ij} \in \{0, 1\}
\end{aligned} \tag{2.61}$$

Figure 2-10 shows the effect of using the static and the dynamic assignment strategies. The x axis is the ratio of the agents' communication range to the maximum allowable distance in the grid $(\frac{R}{\sqrt{2}L})$, where L is the width of the environment. The y axis indicates the average mission completion time. Results show that if the communication range is greater than some nominal value (normalized at 0.2 or 500m in this figure), the advantage of updating the scores dynamically during the assignment is removed. Large communication ranges will diminish the network diameter and decrease the convergence time; therefore, the changes in the scoring matrix throughout the assignment process are minimal, and the static approximation is "good enough." In a low communication environment, this is not the case, and the dynamic nature of the scoring should be incorporated.

Figures 2-11 and 2-12 show the plots for 5 and 20 agents respectively with the dynamic assignment structure. In each case, results indicate that in low communication environments, the consensus-based auction algorithm outperforms both the ETSP and GBAA strategies. As the communication range of the agents is increased, the network connectivity increases to the point where direct conflict resolution can be done at each time step between the agents. With perfect communication, the three

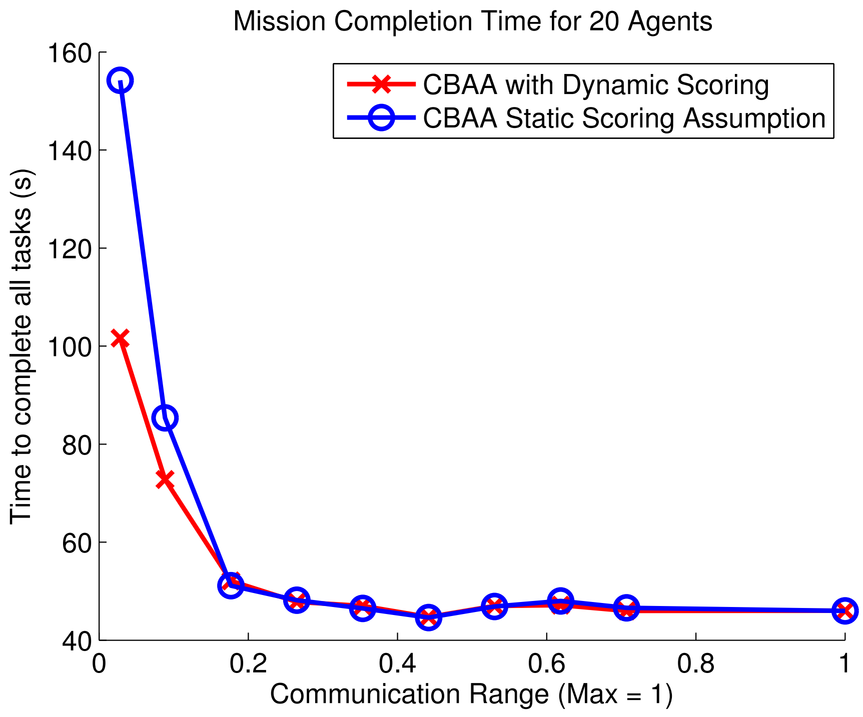


Figure 2-10: The effect of incorporating dynamic tasking to the assignment is beneficial when communication is low and the algorithm takes longer to converge

algorithms converge to approximately the same result. Furthermore, as the number of agents is increased (from Figure 2-11 to 2-12), the deviation between CBAA and both ETSP and GBAA strategies is increased for low communication range values. In this case, CBAA makes use of the added network connectivity as a result of the increased communication range to resolve conflicts more quickly. This indicates the benefits of resolving conflicts through the consensus phase instead of through direct communication only, or by only using the network connectivity to prune unselected tasks as in the ETSP algorithm.

It should be noted that the deviation from optimal in these simulations were quite large. This is because time was used as the objective function, so not only was the final solution sub-optimal, but the time it took to find the solution further degraded the results. The total time was measured from the time the algorithm was started until the time at which the last agent arrived at its destination.

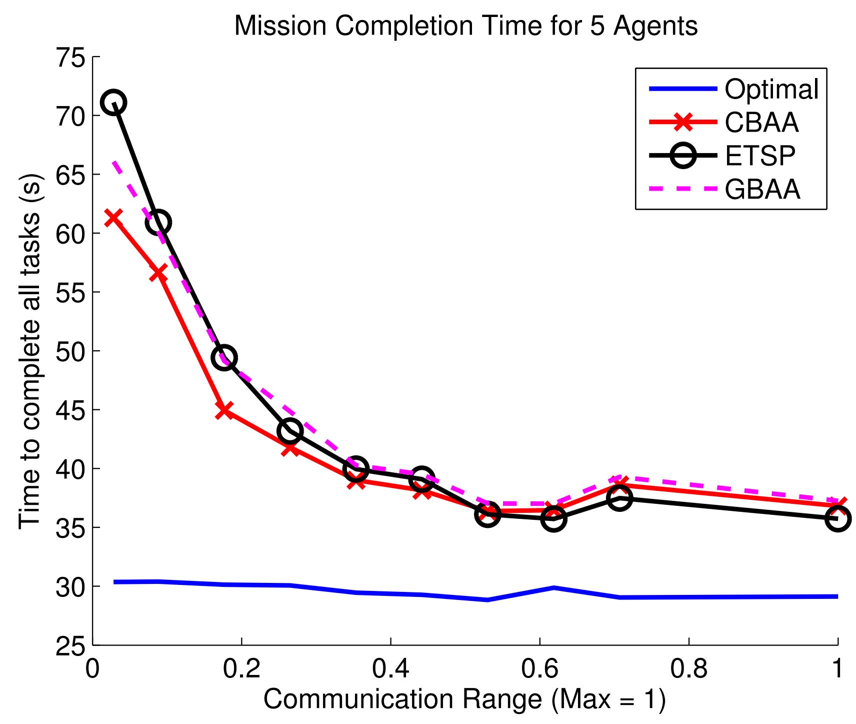


Figure 2-11: By comparing the CBAA to the GBAA and ETSP algorithms, it is shown to have better performance as the communication range is decreased

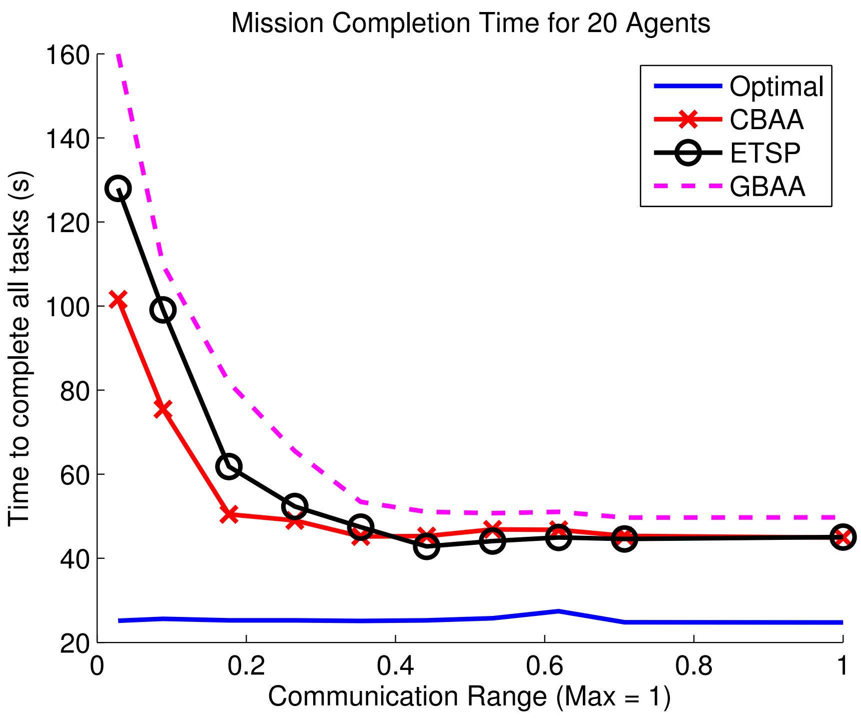


Figure 2-12: The addition of multiple agents to the fleet allows the CBAA to makes use of the extra edges for faster conflict resolution resulting in closer-to-optimal solutions than those found using the GBAA and ETSP cases

2.6.4 Environmental Uncertainty

One of the main features of the CBAA algorithm is that it is guaranteed to converge to a conflict-free assignment, in low communication and highly uncertain environments. Simulations were performed by placing agents and tasks randomly with a continuous distribution over a 2000m $\times$ 2000m environment. Each agent's knowledge of the task locations were perturbed by a random variable (Gaussian distribution with standard deviations incremented from $0.01L$ to $0.2L$) and the SA error (E_{SA}) was calculated using the following

$$E_{SA} = \sqrt{\sum_{i,k:i\neq k}^{N_u} \sum_{j=1}^{N_t} (p_{ij} - p_{kj})^2} \tag{2.62}$$

where p_{ij} and p_{kj} is the estimated position of task j by agent i and k, respectively. Scores for each simulation were calculated based on an agent's estimated distance to a task and static communication networks were created randomly. 1000 MC simulations were performed for each data point.

Figures 2-13 and 2-14 show the optimality and average convergence time as a function of N_t (assuming that $N_t = N_u$) and the normalized error ($\frac{E_{SA}}{\sqrt{2}L}$). The largest simulated E_{SA} value of 0.5 would then indicate that the average position error of a task was equivalent to the distance from the center of the grid, to one of the far corners ($\sim 1414m$). The results show that the optimality of the final solution degrades as the level of inconsistency in the SA grows. However, even with a large amount of error ($\sim$0.5), the average convergence time of the algorithm remains relatively constant at $3 - 3.5$ time steps. This confirms that even with large uncertainty in the environment, the CBAA will converge to a conflict free assignment without changing any of the convergence performance guarantees and the derived bounds will still hold. By comparison, both the implicit coordination and the ETSP algorithms would need to run many consensus iterations to converge on the SA in order to accommodate their underlying perfect knowledge assumptions.

It should be noted that it Figure 2-14, the convergence time is actually slightly improving as the situational awareness error is increased. Note that in the case with

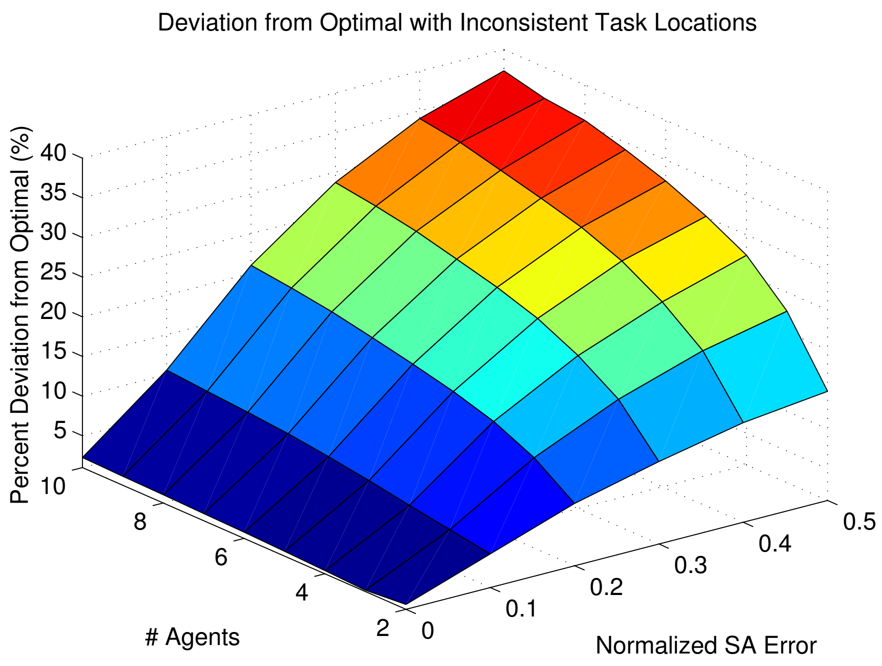


Figure 2-13: Uncertainty in the environment an significantly affect the optimality of the CBAA algorithm

no error, if each agent has a clear distinct best task from the rest of the fleet (i.e. each agent is right next to one of the tasks), the algorithm will converge on the first iteration since each agent will have selected a different task. With large SA error, each agent's estimate of a target position is much different and because of this, the likelihood of a task being the best choice for multiple agents is diminished. This slightly reduces the number of conflicts on average and the convergence time is slightly increased.

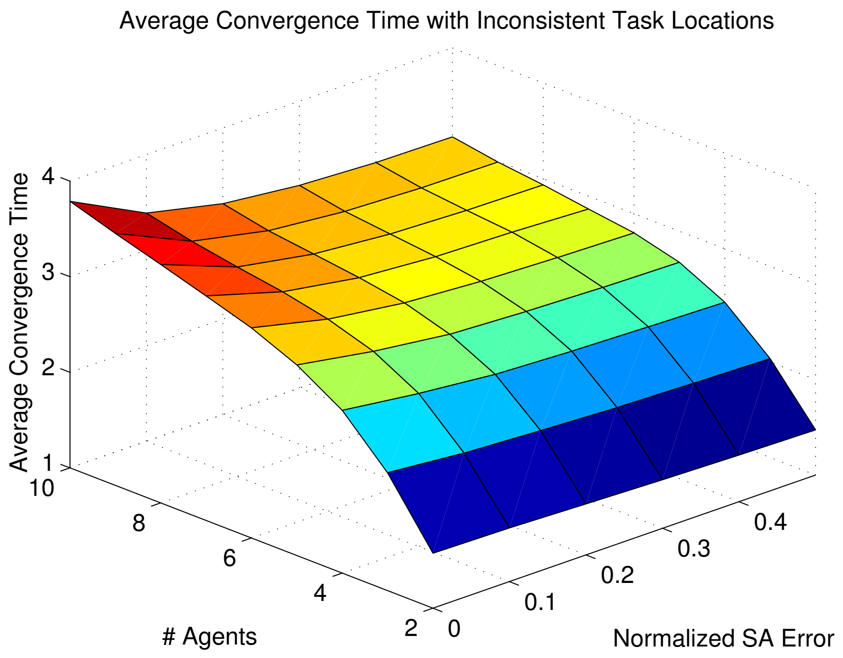


Figure 2-14: The convergence CBAA algorithm is unaffected by large uncertainty in the environment

Chapter 3

Consensus-Based Auction Algorithms for Multiple Assignments

The single assignment algorithm developed in the previous chapter was shown to have some nice convergence and performance properties. This method will work well when $N_t \leq N_u$ since all of the tasks can be completed after a single assignment, but when $N_t > N_u$, a number of tasks are left unassigned and might result in an incomplete mission. Also, it is important that agents have the ability to group common tasks and perform them together rather than selecting a single task to perform at a time. This will not only improve efficiency, but reduce cost as well. In this chapter multi-assignment strategies are developed to enable a small group of vehicles to assign themselves to a large number of tasks.

Multi-assignment auction methods are presented in [53, 61, 62] that sequentially place each task up for auction. At each round, a single-task auction is performed and every agent is able to place a bid. Once the winner is determined, the auctioned task is removed from the list, and the algorithm continues with the next task. [59] proposes a similar algorithm, but in this case, the agents themselves act as the auctioneers and only direct neighbors are able to place bids. This reduces the connectivity requirement on the network, but may provide poor assignments by not considering the entire fleet in the optimization. For large task lists, these sequential auctioning methods may also take a long time to converge, since each agent can only place a bid for one task at a time. The solution can further provide poor performance results depending on the order in which the tasks are auctioned, although in [53], tasks are auctioned in the order of increasing cost. [54] presents a task exchange mechanism to improve the performance; however, slow convergence times can still be problematic.

Combinatorial auction algorithms (often named bundle algorithms) have been proposed in which agents can group common tasks and bid on them as a package. In general, like the single assignment case, each agent is connected to an auctioneer who accepts bids and declares a winner. In [64], agents select multiple combinations of bundles and submit them to an auctioneer for bidding. Bids can either be placed on every possible combination, or heuristics can be used to prune the number of combinations if the computational load is too great. Difficulties arise in selecting the winning bids since agents may not be grouping the same tasks and different bids may have tasks that overlap. Winner determination in these situations have been shown to be $\mathcal{NP}$-complete [67]. However, many approximation algorithms have been developed to solve it [69, 70]. [63] presents an iterative approach in which each agent submits a single bid to the auctioneer at each round. The auctioneer keeps track of the winning bids and increases the task prices for the next round. This process is continued until each agent has submitted a non-conflicting winning bid. The winner determination problem in this case is much easier with only a single bid from each agent, but is done multiple times before the algorithm terminates. Although this method can reduce the computation, many iterations might be required which can make the algorithm slow to converge to a final assignment.

In this chapter, the single-assignment CBAA is extended to the multi-assignment case in which all of N_t tasks are assigned to N_u agents. The objective is to maintain a decentralized approach without forcing a specific network structure as is typical for most combinatorial auctions. Two algorithms will be developed: The first is the Iterative Consensus-Based Auction Algorithm. In this approach, agents perform iterations of the single assignment algorithm. After each round, the set of tasks that was assigned is removed from the task list, the task scores are updated, and the algorithm continues with another assignment iteration until all of the tasks have been assigned. This algorithm, will be shown to provide faster convergence times than sequential auction strategies. However, the assignment can be poor by restricting the size of each agent's assignment to be the same. The second algorithm that will be developed is the Consensus-Based Bundle Algorithm (CBBA). This algorithm groups complimentary tasks into bundles and agents bid on them in tandem. Unlike other bundle algorithms, however, only a single bundle is created and updated throughout the assignment. An extension of the consensus phase for the single assignment algorithm is used to ensure convergence to a conflict-free solution. It will be shown that the CBBA converges much faster than not only sequential auction algorithms, but the iterative CBAA algorithm as well. It will also be shown to guarantee 50% worst-case performance, similar to the CBAA for single assignment. Numerical experiments compared to Prim Allocation (PA) [53], a centralized sequential auction algorithm, will validate the fast convergence and good performance of the CBBA.

3.1 Iterative Consensus-Based Auction Algorithm

Sequential auctions are a common way of assigning multiple tasks to a group of vehicles. In these approaches, a single task is put up for auction by an auctioneer, and each agent submits a bid for it. The agent submitting the highest bid wins the task and the auctioneer proceeds to the next task in the list. However, if the task list is large, auctioning each task individually may be time consuming. Furthermore, agents not in direct communication may either have to route their bids to the auctioneer, or be excluded from the auction altogether. This might significantly decrease the performance of the algorithm.

The iterative approach developed in this section involves iteratively solving the single assignment problem from the previous chapter. Using this approach allows agents to bid on single tasks, but asynchronously with the rest of the fleet. This means that they can bid on any task they choose, which will allow faster convergence times than the sequential approaches. Furthermore, using the consensus phase, the algorithm can guarantee convergence with highly variant network topologies.

3.1.1 Algorithm Development

Let $c_{ij}^n \geq 0$ be the value that agent i receives for placing task j at the n-th iteration of the CBAA, and $H_i \in \{0,1\}^{N_t}$ is the list of available tasks. The iterative approach in this section does not allow reordering of tasks; thus, c_{ij}^n is the marginal score that the i-th agent will receive by adding the j-th task to the end of its current assignment A_i. The assignment A_i is an ordered list consisting of $n-1$ tasks and is initialized as $A_i = \emptyset$. The definition of the winning bids list $y_i(t)$ in this chapter will remain unchanged from the single assignment case, but the assignment list $x_i(t)$ is refined such that $x_{ij}(t)$ is the agent perceived by agent i to have made the current winning bid for j. For instance, $x_{12} = 4$ would indicate that agent 1 believes that task 2 has been won by agent 4.

Thus, in the auction process in this iterative algorithms, if agent i bids on task j, then $x_{ij} = i$ and $y_{ij} = c_{ij}^n$. Consensus in this algorithm will be done on both y_i and x_i vectors with update equations:

$$\begin{aligned} y_{ij}(t) &= \max_k g_{ik}(t) \cdot y_{kj}(t-1) \\ x_{ij}(t) &= \operatorname*{argmax}_k g_{ik}(t) \cdot y_{kj}(t-1) \end{aligned} \tag{3.1}$$

In order to ensure termination of n-th iteration, the agents constantly check the following condition:

$$|X_{ik}| = n, \quad \forall k \tag{3.2}$$

where $X_{ik} \triangleq \{j \in \{1 \ldots N_t\} | x_{ij} = k\}$. In other words, it is the set of all tasks that agent i thinks are assigned to agent k, and $|X_{ik}| = n$ indicates that each agent has been assigned exactly one task for each iteration. If this is true for all $k \in \{1 \ldots N_u\}$, then the assignment is known to have converged. Once this is done, the assigned tasks are removed from the list and added to each agent's assignment:

$$A_i = X_{ii}, \tag{3.3}$$

while the scores are recalculated to reflect the current assignment. Note that

$$c_{ij}^{n+1} = 0, \quad \forall j \in A_i \tag{3.4}$$

with A_i being the updated assignment list of length n, because a task already included in the assignment list will not produce any marginal reward.

3.1.2 Convergence

Section 2.3 in Chapter 2 outlines the convergence of the single assignment algorithm. Taking the result from proposition 3, it is shown that the upper bound on a single iteration of the algorithm is $D_n \cdot m$, where D_n is the network diameter for iteration n of the algorithm, and $m = \min\{N_u, N_t\}$. The number of single-assignment iterations N_i required for convergence can be calculated as

$$N_i = \left\lceil \frac{N_t}{N_u} \right\rceil \tag{3.5}$$

Thus the upper-bound on convergence for the iterative CBAA can be calculated as

$$T_{IT} \leq \sum_{n=1}^{N_i} D_n \cdot m \tag{3.6}$$

Although a tighter bound can be obtained by exploiting the 0-1 program in proposition 4, the above will be sufficient to show convergence.

Comparing this bound with those of the decentralized sequential auction algorithms is straightforward. For the assignment of the n-th task, the bid from each agent must be routed through the network to the auctioneer taking at most D_n time steps to arrive. Since the auctioneer must receive all of the bids before making an assignment, each task will take exactly D_n time steps to be assigned. Thus, the convergence time can be calculated as

$$T_s = \sum_{n=1}^{N_t} D_n. \tag{3.7}$$

In case $D_n = D$ is constant through the assignment, the convergence time for the sequential auction approach will be

$$\bar{T}_s = D \cdot N_t. \tag{3.8}$$

With the same network assumption for the iterative CBAA, and assuming $N_t > N_u$ (thus, $m = N_u$), it can be shown that

$$\begin{aligned}
\bar{T}_{IT} &\leq (N_i - 1) \cdot D \cdot N_u + D \cdot (N_t - (N_i - 1) \cdot N_u) \\
&\leq N_i \cdot D \cdot N_u - D \cdot N_u + D \cdot N_t - N_i \cdot D \cdot N_u + D \cdot N_u \\
&\leq D \cdot N_t \\
&\leq \bar{T}_s.
\end{aligned} \tag{3.9}$$

where the first inequality is from the worst-case convergence bound in proposition 3. If $N_u > N_t$ (i.e. $m = N_t$), the algorithm reduces to a single iteration of the single assignment algorithm and the same inequality holds. Thus, for a given network, the convergence time for the CBAA is always upper-bounded by the sequential auction approach. Moreover, as presented in the previous chapter, the actual convergence time of a single iteration of the CBAA is much faster than the worst-case bound $D \cdot m$; thus, the iterative CBAA will converge much faster than decentralized sequential auction algorithms. Numerical simulations in section 3.4.2 will verify this.

3.2 Consensus-Based Bundle Algorithm

Many different bundle algorithms have been explored in the literature [63–66]. In many cases, agents bid on multiple combinations of tasks and participants submit bids to an auctioneer. The auctioneer can then run an optimization algorithm to determine which of the agents have won assignments. Various winner determination algorithms can be used in order to assign winning bids to the bidders. However, this process can be computationally intensive for a large number of tasks and agents [67]. Furthermore, each agent must be able to submit bids to the auctioneer, thus, the network must either be fully connected or connected with sufficient density so that each agent can route bids through the network to the auctioneer.

This section will develop the Consensus-Based Bundle Algorithm (CBBA). The algorithm maintains only a *single bundle* that is updated as new tasks or conflicts arise. By doing this, computational requirements are kept low and the algorithm iterations can be executed at much higher rates to improve convergence time. Each task in the bundle has a separate bid that is based on the improvement in score the bundle received by adding it. Conflicts in task selection are resolved greedily based on which bid is the highest for a given task. This is a much simpler winner determination system than in conventional bundle approaches in which the winner is determined by a complex optimization [69, 70]. Even with this simplified winner determination system, the CBBA will still be shown to provide near-optimal assignments. Furthermore, like the iterative algorithm previously discussed, it makes use of the consensus phase from the single assignment algorithm for conflict resolution, allowing it to converge under variable network types. Once again the algorithm is a two phase process. In the first phase, each agents lists the available tasks and sequentially selects a task to include into a candidate bundle. The second phase invokes a consensus algorithm in order to resolve conflicts in the bundle with the rest of the fleet. The algorithm will be shown to produce faster convergence times and better assignments than the iterative approach discussed in section 3.1. The algorithm will also be compared against the Prim Allocation (PA) algorithm [53], and it will be shown to exhibit faster convergence times and better assignments.

3.2.1 Phase 1: Building a Bundle

Let an agent i's bundle B_i be defined as the set of tasks that have been added to its assignment, and path L_i be the ordered list of tasks it has been assigned. It should be pointed out that tasks in the bundle are ordered based on which ones were added first, while in the path they are ordered based their location in the assignment. The marginal score improvement of task j performed by agent i is then

$$c_{ij} = \begin{cases} 0, & \text{if } j \in L_i \\ \max_{n \leq N_{L_i}+1} S_i^{L_i \oplus_n \{j\}} - S_i^{L_i}, & \text{otherwise.} \end{cases} \tag{3.10}$$

where $\oplus_n$ denotes the operation that inserts task j into the n-th position of a given path, and N_{L_i} is the cardinality of L_i. It is assumed that the addition of any task to a bundle will result in a non-negative improvement in score; in other words, $c_{ij} \geq 0$.

It will also be assumed that the scoring function is submodular (3.11) [92], meaning that the value of a task does not increase as other elements are added to the set before it. In other words, a task is either worth more or the same the earlier it is added to the bundle: for $B_- \subset B_+ \subset \mathcal{J} = \{1, 2, \ldots, N_t\}$, it is satisfied for all $j \in \mathcal{J} \setminus B_+$ that

$$f(B_- \cup \{j\}) - f(B_-) \geq f(B_+ + \{j\}) - f(B_+) \tag{3.11}$$

with a slight abuse of notation in treating a bundle as an unordered set. This does not lose any generality, because the scoring function $f(\cdot)$ can be defined as a two-step process that generates an appropriate path, and computes the score associated with that path. This submodularity will be required for convergence of the algorithm, and detailed convergence properties will be discussed in section 3.2.4. Also, a simple modification of the bidding scheme that ensures convergence of the algorithm with a non-submodular scoring function will be presented as well.

A bundle of tasks B_i is created by adding tasks sequentially until some maximum path length N_m has been reached, or until $H_i = 0$, where H_i is defined in (2.8) as the set of valid tasks, and K_i is the agent's capability vector. Algorithm 3 shows the first phase of the assignment process where y_i is the winning bids list, x_{ij} is the agent perceived by i to have made the current winning bid for j, and L_i is the current path for agent i.

The function *GetUpdatedScores* on line 7 of the algorithm takes the current path

Algorithm 2 Bundle Algorithm Phase 1 at time t:

```
 1: $\forall i \in \{1, \ldots, N_u\}$
 2: procedure BUILD BUNDLE($x_i(t-1), y_i(t-1)$)
 3:     $y_i(t) = y_i(t-1)$
 4:     $x_i(t) = x_i(t-1)$
 5:     while $length(L_i) < N_t$ do
 6:         $[c_i,\ n_i] = GetUpdatedScores(L_i)$
 7:         $H_i = (c_{ij} > y_{ij}(t)) \wedge K_i, \quad \forall j \in \{1, \ldots, N_t\}$
 8:         $J_i = \text{argmax}_j\, H_i c_{ij}$
 9:         if $c_{iJ_i} > 0$ then
10:             $B_i = B_i \oplus_{|B_i|} \{J_i\}$
11:             $L_i = L_i \oplus_{n_{iJ_i}} \{J_i\}$
12:             $y_{iJ_i}(t) = c_{iJ_i}$
13:             $x_{iJ_i}(t) = i$
14:         else
15:             break
16:         end if
17:     end while
18: end procedure
```

L_i, and returns for each task j, the increase in value c_{ij} the bundle will gain by including it, and the index at which the task should be inserted into the current assignment n_{ij}. The location at which the task will be inserted into the path will depend on the scoring function. Any function that satisfies non-negativity and submodularity can be used as a scorning function. For instance, this section uses the following scoring function:

$$S_i^L = \sum_{k=1}^{|L|} V_{ij}^k = \sum \lambda^{T_{jk}(L)} C_j \tag{3.12}$$

where V_{ij}^k be the time discounted score for agent i along the path L for task j at index k. $\lambda \in [0, 1]$ is the time discount factor, $T_{jk}(L)$ is the estimated time agent i will take to arrive at task j along the path L, and C_j is the static score associated with performing task j. Since the path is determined to produce maximum score for a given bundle, the function *GetUpdatedScores* returns the scoring vector c_i whose j-th element is

$$c_{ij} = \max_n \left\{ \sum_{k=1}^{n-1} V_{ij}^k + V_{ij}^n + \sum_{k=n}^{N_{L_i}} V_{ij}^{k+1} \right\}. \tag{3.13}$$

This is similar to the cheapest insertion heuristics [93] found in traveling salesman problems (TSP). Note that the task is inserted into the bundle B_i at the back end of the list, while the task is inserted into the path L_i at the argmax of the right-hand-side of (3.13). Details of the cheapest insertion heuristic can be found in Appendix B.

3.2.2 Phase 2: Conflict Resolution

In the single assignment algorithm, agents bid on a single task and release it upon receiving a higher value in the winning bids list. In the CBBA, however, agents add tasks to their bundle based on their currently assigned task set. Thus, if an agent is outbid for a task, it should also release the tasks that were added to the bundle after it, since they might no longer be the best ones to select. By allowing agents to outbid earlier selected tasks, this algorithm will be able to achieve higher valued assignments than the iterative approach previously developed. Releasing the tasks in this manner can however cause further complexity in the algorithm. If an agent is able to release tasks without another member selecting it, the consensus update equations from (2.10) and (2.11) will no longer converge to the appropriate values since the maximum bid observed might no longer be valid. Therefore, the consensus phase of the algorithm will need to be enhanced in order to ensure that these updates are accurate.

In the multi assignment consensus stage, three vectors will be used for consensus. The first two vectors will be identical to the iterative assignment algorithm developed in the previous section: 1) the winning bids list y_i and 2) the winning agent x_i. The third vector, τ_i of length N_u, is the time of the last information update from each of the other agents. Each time a message is passed, the time vector is populated with

$$\begin{aligned} &\tau_{ik} = t_r, && \forall k \in \{k : g_{ik} = 1\}, \\ &\tau_{ij} = \max_{k\in\{k:g_{ik}=1\}} \tau_{kj}, && \forall j \in \{j : g_{ij} = 0\} \end{aligned} \qquad (3.14)$$

where k and i are the sender and receiver agents, and t_r is the message reception

Table 3.1: Actions to take based on incoming message parameters

Sender (x_i)	**Receiver** (x_j)	**Action for** j
i	j	if $y_i > y_j \rightarrow$ update
	i	update

	k	if $t_{ik} > t_{jk} \rightarrow$ update
		else if $y_i > y_j \rightarrow$ update

	none	update
j	j	nothing
	i	reset
	k	if $t_{ik} > t_{jk} \rightarrow$ reset
	none	nothing
k	j	if $t_{ik} > t_{jk}$ and $y_i > y_j \rightarrow$ update

	i	if $t_{ik} > t_{jk} \rightarrow$ update
		else $\rightarrow$ reset

	k	if $k_i == k_j$ and $t_{ik} > t_{jk} \rightarrow$ update
		else if $t_{ik_j} > t_{jk_j}$ and $t_{jk_i} > t_{ik_i} \rightarrow$ reset
		else if $t_{ik_j} > t_{jk_j} \rightarrow$ update
		else if $t_{ik_i} > t_{jk_i}$ and $y_i > y_j \rightarrow$ update

	none	if $t_{ik} > t_{jk} \rightarrow$ update
none	j	nothing
	i	update
	k	if $t_{ik} > t_{jk} \rightarrow$ update
	none	nothing

time.

When agent j receives a message from another agent i, the x and t lists are used to determine which agent's information is the most up-to-date for each task. Once this is done, a decision is made to either update the current values in the list, reset them to the default values ($x_{ij} = y_{ij} = 0$), or leave them alone. Table 3.1 outlines all of the possible cases.

The first two columns of the table indicate the agent that each of the sender i and receiver j believes to be the current winner for a given task. The third column indicates the action the receiver should take. The agents, upon receiving a message, verify which of the information states are correct using the table and then take the appropriate actions. In the table there are four different cases of x_i values to consider:

1. it is believed the sender (i) has won the bid

2. it is believed the receiver (j) has won the bid

3. it is believed some other agent (k) has won the bid

4. it is unknown (none) who has won the bid

If a bid is changed during the second stage, each agent checks to see if any of the updated tasks were in their bundle, and if so, those tasks along with all of the tasks that were added to the bundle after them, are released:

$$\begin{aligned} X_i &= \{j \in B_i | x_{ij} \neq i\} \\ k_i &= \min k \in \{k | B_{ik} \in X_i\} \\ B_i &= B_i \setminus \{B_{i,k_i}, \ldots, B_{i,|B_i|}\}. \end{aligned} \tag{3.15}$$

From here, the algorithm returns to the first stage and new tasks are added from the task list.

3.2.3 Performance

This section will show that the performance guarantee for the Consensus-Based Bundle Algorithm is within 50% of optimal. To do this, it will first be shown that the calculation of the optimal solution can be formed as a Traveling Salesperson Problem (TSP) [94]. It will then be shown that the CBBA developed herein is equivalent to the Cheapest Insertion (CI) heuristic for solving TSPs [93, 95], which has a well known bound of 50% of the optimal solution. Details of both the TSP and the CI heuristic can be found in Appendix B.

Proposition 8. *The optimal task allocation of N_t tasks to N_u agents is equivalent to solving a TSP with edge weights $e_{kj} = c^k_{ij}$ if the agent prior to task j on the tour ($k-1$ nodes separate them) is agent i, and 0 otherwise.*

Proof. First, note that if there is only a single agent, the maximum scoring arrangement of tasks is equivalent to a standard maximum TSP with an edge weight

$$e_{kj} = \begin{cases} c^k_{ij}, & \text{if } j \in A_i \\ 0, & j = i \end{cases} en \tag{3.16}$$

where edges from tasks to agents have weights of 0. The individual problem can be written as a general TSP using

$$\begin{aligned}
\max \sum_{j=1}^{N_t+1} \sum_{k=1}^{N_t+1} & c^i_{kj} e_{kj} \\
\text{subject to } & \sum_{k=1}^{N_t+1} e_{kj} = 2 \\
& \sum_{\delta(S)} e_{kj} \geq 2, \quad \forall S \subset \mathcal{V}, S \neq \emptyset \\
& e_{jk} \in \{0, 1\}
\end{aligned} \tag{3.17}$$

where $\delta(S)$ denotes the set of all nonempty subsets of nodes, and $\mathcal{V} = A_i \cup \{i\}$ is the list of vertices for agent i and its tasks. $e_{kj} = 1$ if there is an edge between vertices k and j.

For multiple vehicles, the optimal assignment is the sum of each of the individual assignments (3.17) with an extra constraint prohibiting the same task from appearing in multiple assignments. The assignment can then be written as follows:

$$\begin{aligned}
\max \sum_{i=1}^{N_u} & \left\{ \sum_{j=1}^{N_{L_i}+1} \sum_{k=1}^{N_{L_i}+1} c^i_{kj} e^i_{kj} \right\} \\
\text{subject to } & \sum_{k=1}^{N_{L_i}+1} e^i_{kj} = 2 \\
& \sum_{\delta(S)} e^i_{kj} \geq 2, \quad \forall S \subset \mathcal{V}_i, S \neq \emptyset \\
& \sum_{k \notin A_i, j \in A_i}^{N_t} e^i_{kj} = 0 \\
& e^i_{jk} \in \{0, 1\}
\end{aligned} \tag{3.18}$$

where $\mathcal{V}_i$ is the list of vertices in agent i's assignment tour. Notice that if the second constraint in (3.18) is changed such that it spans the entire vertices set over the fleet $\mathcal{V}$, then the resulting formulation has the form of a TSP.

When performing the optimization in (3.18) with $\mathcal{V}$, edges need to be added in between the assignments in order to satisfy the first two constraints. However, the

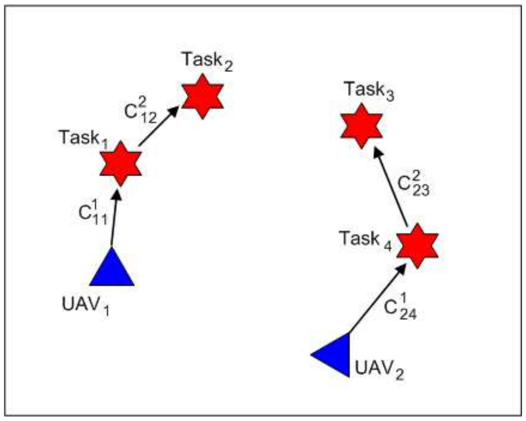


Figure 3-1: Optimal Assignment

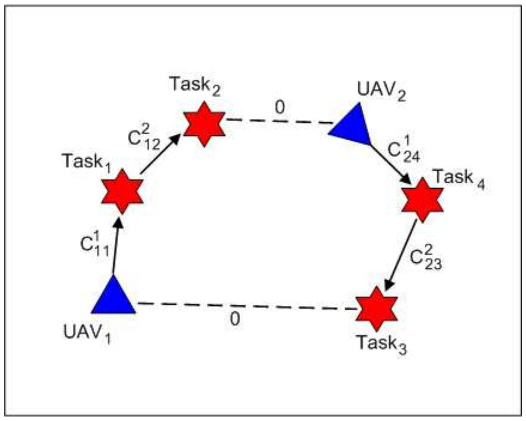


Figure 3-2: Assignment Reordered as a TSP

third constraint prohibits edges between tasks belonging to different assignments. The only way to satisfy this is to add edges in between a task from one assignment, to the agent of another. An example of this is shown in Figures 3-1 and 3-2. From (3.16), it is known that edges from tasks to agents have zero weight, and thus, the value of the assignment will remain unchanged. Therefore, solving (3.18) with the edge constraints listed in (3.16) will provide the optimal assignment.

□

Proposition 9. *The Consensus-Based Bundle Algorithm performs a cheapest insertion heuristic on the assignment; therefore, it guarantees 50 % optimality.*

Proof. Let the initial tour be defined by zero-weighted edges between all agents (Figure 3-3). Let C_1 be the initial scoring matrix before any tasks are added to an agent's bundle such that c^1_{ij} is the score for adding task j to the first index:

$$C_1 = \begin{bmatrix} c^1_{11} & c^1_{12} & \cdots \\ c^1_{21} & c^1_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{3.19}$$

Like in 2.12, there will be some value (i^*_1, j^*_1) for which $(i^*_1, j^*_1) = \operatorname{argmax}_{(i,j)} c^1_{ij}$. It is also known that $B_{i^*_1 1} = j^*_1$ since the bundles are built greedily. With a submodular scoring function, the task value that other agents have will never be higher than

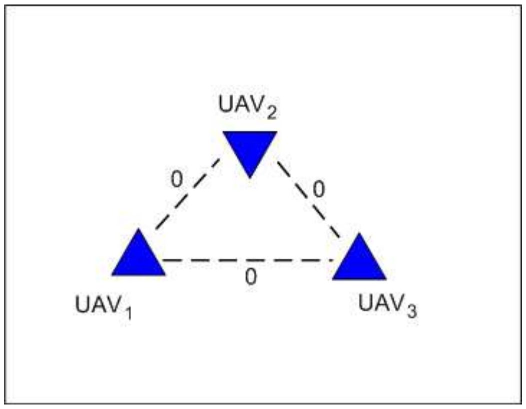


Figure 3-3: Initial Tour

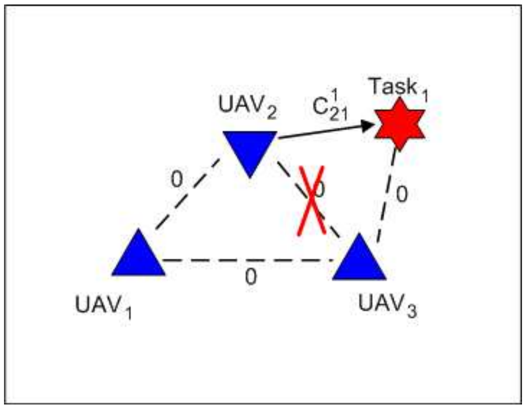


Figure 3-4: First Insertion

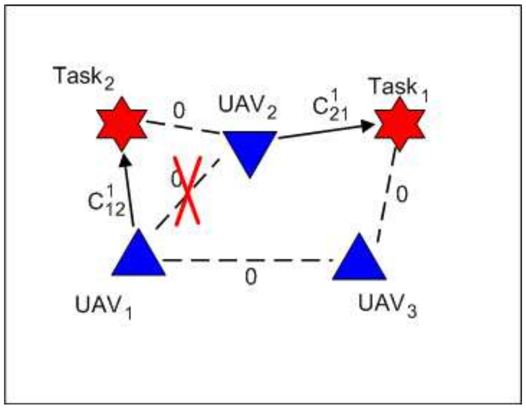


Figure 3-5: Second Insertion

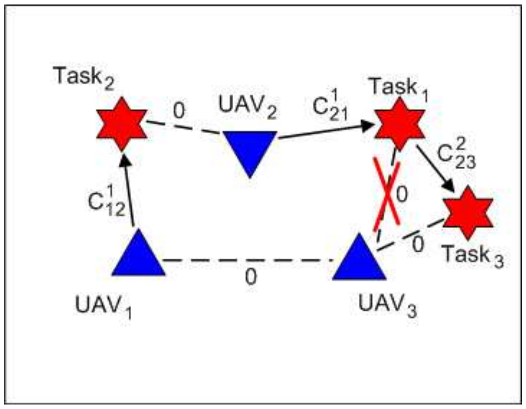


Figure 3-6: Third Insertion Insertion

the value in the initial matrix, so it is known that (i_1^*, j_1^*) can never be outbid and the assignment will always hold. Thus, it can be inserted after i_1^* as a part of the assignment (See the link from UAV$_2$ to Task$_1$ in Figure 3-4). Similarly, after task j_1^* is removed, the scoring matrix will be updated to reflect the assignment as C_2. The same process can be followed to find the next highest value and produce the assignment (i_2^*, j_2^*), which can be inserted into the network next and so on (Figures 3-5 and 3-6). This iterative process is precisely the cheapest insertion algorithm outlined in [93], which has a proven lower bound of 50% of the optimal tour. Proposition 8 showed that the TSP tour was equivalent to the optimal assignment, thus it can be reasoned that the CBBA will give an assignment no worse than 50% of the optimal one. □

Remark 3. *While the process in the proof of proposition 9 adds a task one at a* *time, the actual CBBA algorithm may assign many tasks at the same time. However, the performance guarantee still holds, because there always exists a sequential process equivalent to simultaneous insertion of multiples tasks.*

3.2.4 Convergence

As mentioned in section 3.2.1, it is assumed that a task's scoring function is submodular. This means that its value does not increase the later it is added to the plan.

Proposition 10. *With a submodular scoring function, the CBBA converges within $\bar{T}_B \leq D \cdot N_t$ time steps, where D is the network diameter.*

Proof. In the worst case, task bundles are built but only one task is won in the fleet at a time. This is outlined in Figure 3-11. In it, two agents i and k are separated by the network diameter D, and are incrementally outbidding each other for successive tasks. This will cause the most conflicts and force the resolution time to take the most number of iterations since it must traverse the longest path on the network. The first conflict will be resolved in at most D iterations (Fig. 3-7). Once this is done, agent k will release each task in its list and rebuild the bundle (Fig. 3-8). After another D iterations the newest conflict is resolved an the process is continued for $D \cdot N_t$ iterations until the assignment is complete (Fig. 3-10). Thus, the algorithm is bounded by a $\bar{T}_B \leq D \cdot N_t$ convergence time. $\square$

Without submodularity, this convergence bound could not be guaranteed. Consider the two agents with the initial scoring function for two tasks shown in Figure 3-12. They each select the highest valued task first ($A_1 = \{2\}$ and $A_2 = \{1\}$), and then recalculate the scoring function (Figure 3-13). Without the submodularity assumption, the value for the remaining target(s) might be higher than their original scores. Once the recalculation is done, each agent again adds the next best task to their bundle. Thus the assignments are $A_1 = \{2, 1\}$ and $A_2 = \{1, 2\}$ while the bids are $y_1 = \{50, 100\}$ and $y_2 = \{100, 50\}$. Each agent will win one of the tasks, however, they will also both lose the first task that they selected in the bundle. Using

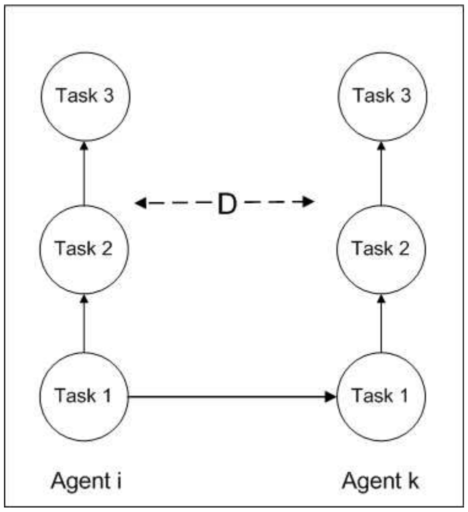


Figure 3-7: Iteration 1

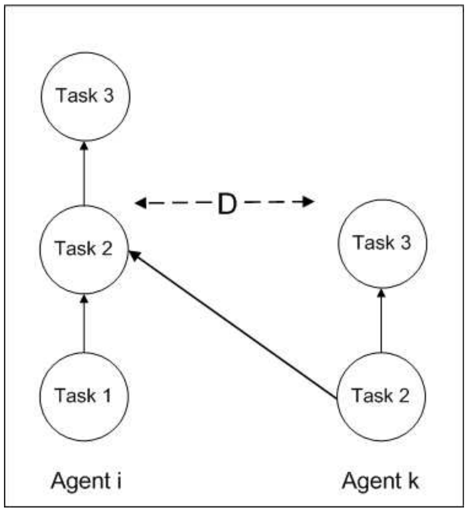


Figure 3-8: Iteration $2 \times D$

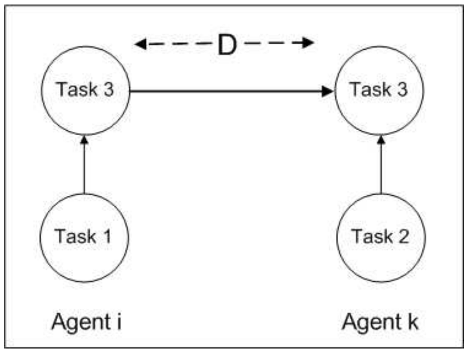


Figure 3-9: Iteration $3 \times D$

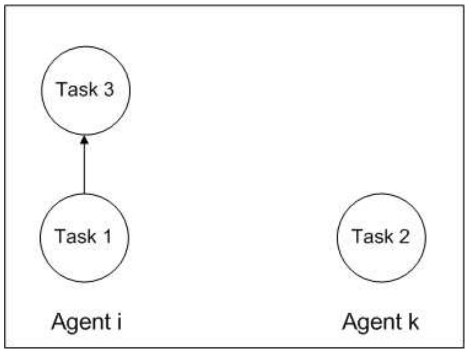


Figure 3-10: Final Assignment

Figure 3-11: Progression of the worst case convergence time of the CBBA across the network diameter D

the conflict resolution rules developed in section 3.2.2, the agents, having both been outbid for their first task, will both give up tasks 1 and 2. This will create a cycle in the bidding and conflict resolution process which can be very complicated and hard to detect for large fleets. By ensuring that the scoring function is submodular, the first task selected is always the greatest value and the same cycle will not occur.

Similarly to the iterative CBAA approach, the same reasoning can be used to show that for a given network, the convergence time of the CBBA is upper-bounded

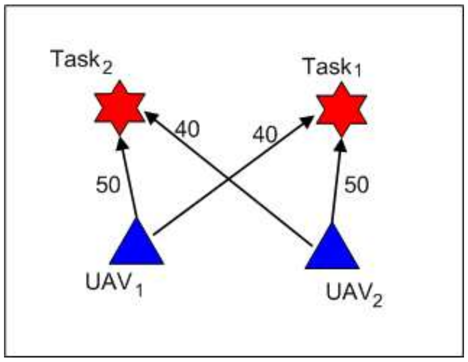


Figure 3-12: Initial Scoring

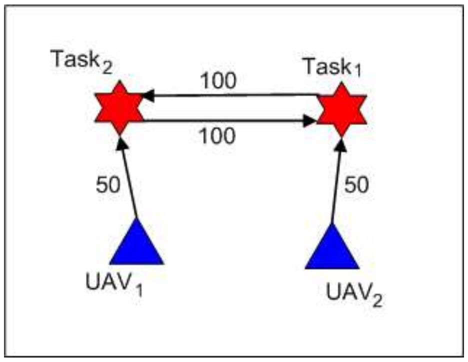


Figure 3-13: Scores after first selection

by that of the sequential auctioning approach:

$$\bar{T}_B \leq \bar{T}_s \tag{3.20}$$

This bound, however, is very unlikely to be attained. For it to occur, two agents on opposite ends of the network would need select the same bundle, and each one win alternating tasks in the list as in Figures 3-11. For fleets of even a small size, the likelihood of 1) no other agents winning a bid, and 2) only one task being allocated at a time, is very small. In practise, the expected convergence bound will be much smaller than this since conflicts can be resolved in parallel and agents will select tasks asynchronously.

Remark 4. *Even in the case that a scoring function is not submodular, the following simple modification of the algorithm that ensures a monotonically decreasing property of a bid over time will lead to convergence*

$$c_{ij}(t) = \min\left\{c_{ij}(t), c_{ij}(t-1)\right\}. \tag{3.21}$$

3.3 Robustness

At the beginning of this chapter, it was stated that in any multi-assignment extension of the CBAA, the convergence properties of the algorithm should not change. More specifically, the multi-assignment extension should still guarantee convergence with inconsistencies in the SA, and converge to a conflict free solution in dynamic networks. This section will show that for both the iterative and bundle versions of the CBAA, these properties still hold.

Proposition 11. *If the maximum assignment size N_m is set to 1, then both iterative CBAA and the CBBA reduces to the single assignment CBAA.*

Proof. For the iterative CBAA, this is a trivial result since it uses iterations of the single assignment algorithm. Each iteration causes the assignment size of every agent to increase by one. Thus, running the single assignment algorithm once will finish the assignment with $N_m = 1$.

In the CBBA, limiting the assignment size will cause the agents to select only a single task. Task scoring is based on the improvement in score the agent receives, but if $N_m = 1$ this value will simply be the task score c_{ij} since there will never be any tasks in the list. Thus, bidding is the same as the single assignment algorithm. In the consensus phase, an additional vector τ_i was needed to determine when tasks were released without being directly outbid (Eq. 3.15). However, this can only occur for task sizes greater than one, since agents release tasks that were added after a task that was outbid. Thus, the winning bids list is only updated if the received winning bid is greater then the current one. This indicates that the consensus phase is the same as the single assignment case as well, and the algorithms are said to be the same. □

With proposition 11, it is known that for $N_m = 1$, both multi-assignment extensions satisfy the CBAA robustness properties. For the iterative approach, it is trivial to show that this will always be the case for $N_m > 1$ since it can be argued that each successive iteration is a new assignment using the single assignment algorithm with a slightly different task list. For the CBBA, proposition 10 showed that it will converge in finite time. With $N_m > 1$, conflict resolution happens in parallel for each task that is bid across the network diameter. If the diameter is changing, the assignment will not change provided that the network is averagely connected as stated in (2.23). Similarly with inconsistent information, bids are based on the local data set. If the algorithm can converge with $N_m = 1$, increasing N_m will simply add more tasks to the parallel conflict resolution process, but the same arguments can be made for each successive task added to the bundle as was made for $N_m = 1$. Thus, the multi-assignment algorithms will both maintain the same robustness to varying networks and inconsistent SA the single assignment CBAA.

3.4 Results

Monte-Carlo (MC) simulations were performed to quantify the performance, convergence, and computation time of the multi-assignment algorithms. For each simulation, tasks and agents were randomly placed on a $2000m \times 2000m$ grid. Each task was given a fixed score of 1000 and the assignment scores for each agent were based on a time discounted value as in (3.12). Networks were created the same way as in the previous section, by generating a random spanning tree [91], and then adding varying amounts of random links to the network. For comparisons, the optimal solution was calculated by using the implicit coordination algorithm [43]. The algorithm assumes perfect SA over the fleet and solves the assignment by listing all of the possible task combinations and solving for the optimal using a MILP. For both the iterative and bundle versions of the CBAA, numerical results were used to verify their performance, convergence times, robustness to uncertainty and computation time. Further simulations were done to compare the CBBA with the Prim Allocation (PA) algorithm [53].

3.4.1 Performance

Figures 3-14 and 3-15 show the performance for the iterative and bundle approaches respectively. Due to computational limitations with the optimal solution, a receding horizon approach had to be taken in which the assignment sizes were limited to $N_m = 3$. For the iterative approach, the worst case average performance was found along the main $N_t = N_u$ diagonal. This is the same result from the single assignment case in the previous chapter. There is also a second ridge along the $N_t = 2 \cdot N_u$ line.

This is from the final iteration having $N_t = N_u$ tasks remaining. The assignment value reaches a maximum of an approximate 10% deviation from optimal. For the CBBA, the worst case performance was found when $N_u \ll N_t$, and had a maximum average deviation from optimal of 4%.

Simulations were done with $N_u = N_t$ in order to see the value in each algorithm while changing the maximum assignment size N_m. With $N_m = 1$ (Figure 3-16), as was shown in section 3.3, the two algorithms converge to the same solution. Here, the assignment is better than the 6% bound found in Chapter 2, as the scoring isn't based on a random distribution. With the scoring based on a physical distance, less conflicts will arise on average, leading to more vehicles winning their preferred assignments. Figure 3-17 shows the same simulations with a horizon size $N_m = 5$. In all cases, the bundle algorithm was shown to outperform the iterative approach. This was expected since the iterative approach essentially forces each assignment to have the same length.

Finally, the performance of both algorithms were compared to the 50% performance bound. Figure 3-18 shows the results from simulations performed in the iterative assignment worst-case configuration ($N_u = N_t$), while figure 3-19 shows it in the CBBA worst-case configuration $N_u \ll N_t$. In both cases the solutions were well above the bound.

3.4.2 Convergence

To measure the convergence of the respective algorithms, simulations were first performed with a fully connected network. Doing this provided insight into the agent and task ratios providing the most conflicts. Figure 3-20 shows the convergence for the iterative approach. It shows that the algorithm converges much slower when the number of tasks is a multiple of the number of agents ($N_t = n \cdot N_u$). This can be seen in Figure 3-20 as the second ridge of the diagonal has $n = 2$. This matches well with result from the single assignment case, since the last iteration of the algorithm is the case where $N_t = N_u$. Figure 3-21 shows the equivalent simulations for the bundle algorithm, and once again shows the worst case to be when $N_u \ll N_t$. Simulations

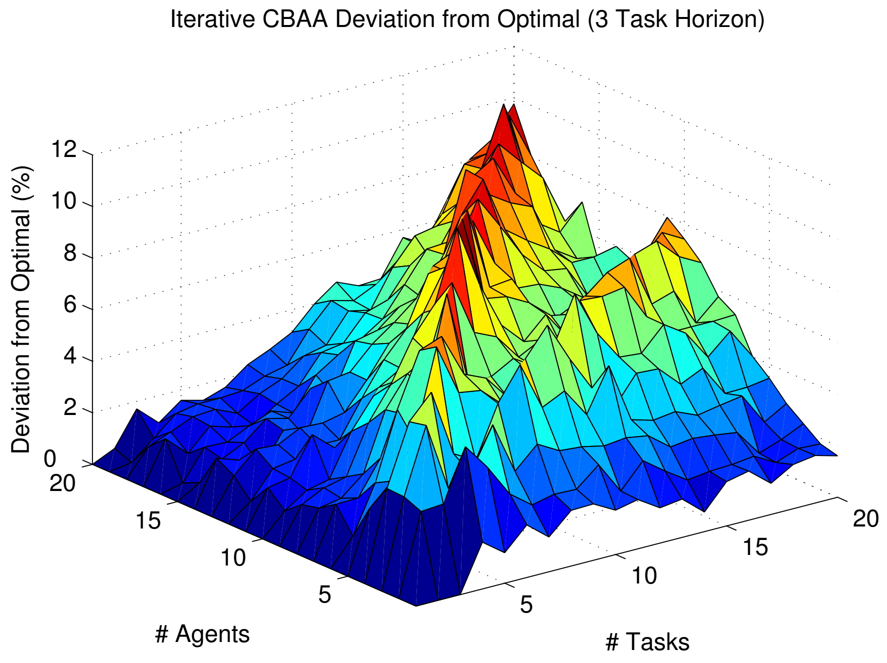


Figure 3-14: Deviation from Optimal for Iterative CBAA

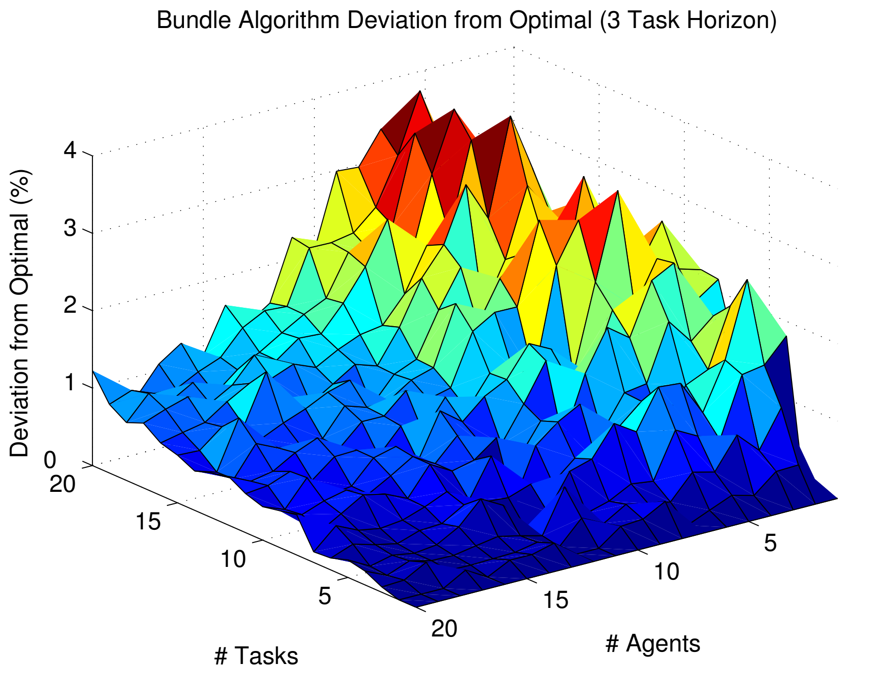


Figure 3-15: Deviation from Optimal in CBBA

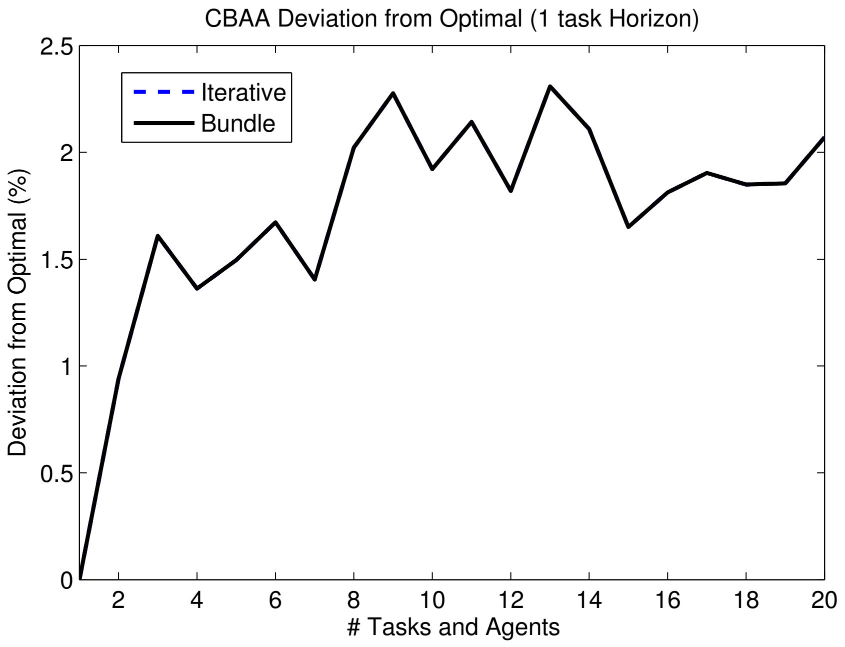


Figure 3-16: Deviation from Optimal with $N_m = 1$

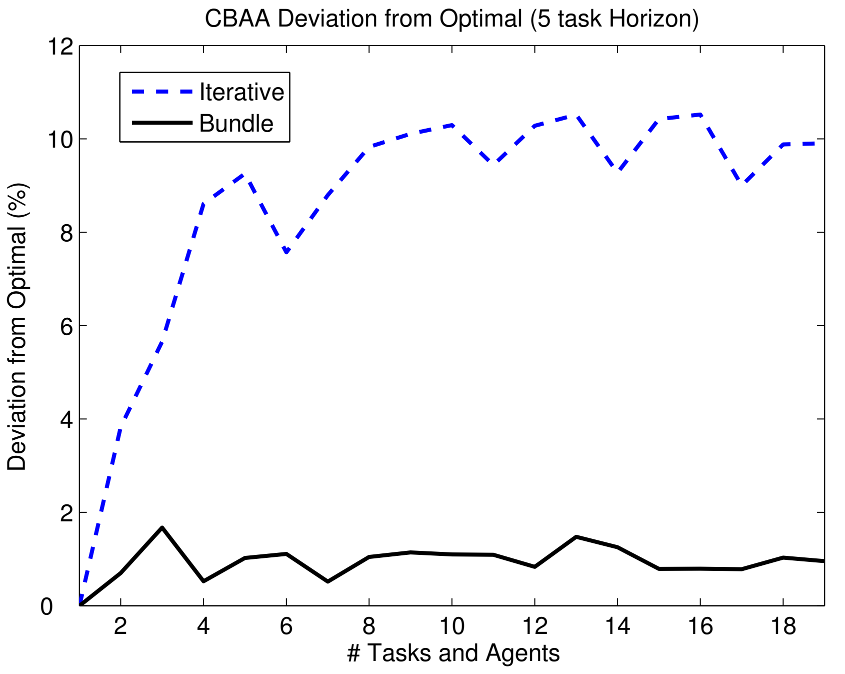


Figure 3-17: Deviation from Optimal with $N_m = 5$

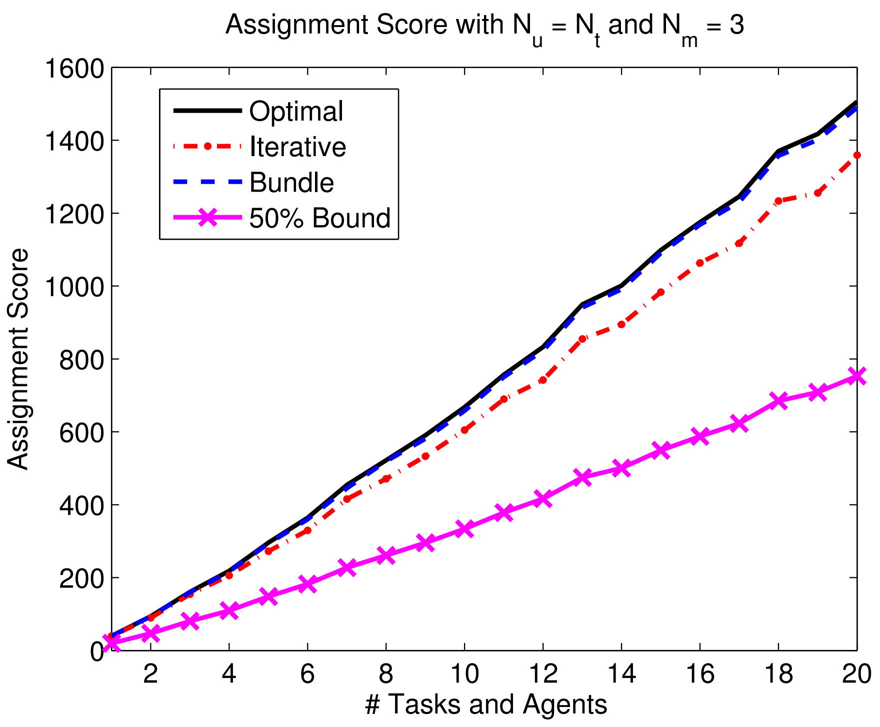


Figure 3-18: Performance Bound in Iterative CBAA Worst Case Configuration ($N_u = N_t$)

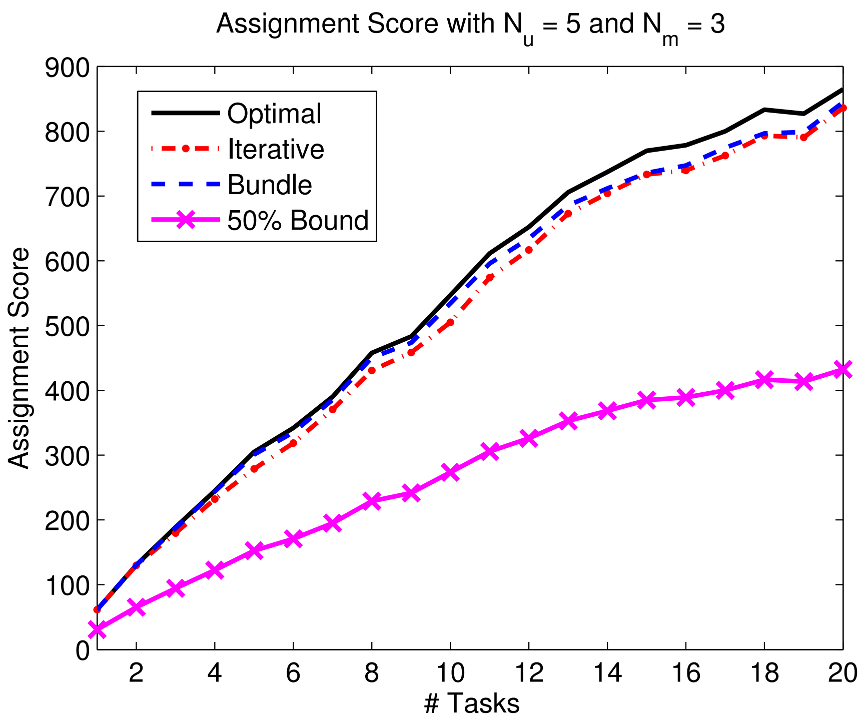


Figure 3-19: Performance Bound in CBBA Worst Case Configuration ($N_u \ll N_t$)

Table 3.2: Computation Time (s) with $N_u = 5$ and $N_t = 20$

N_m	**1**	**3**	**5**
iterative	0.007	0.0119	0.014
bundle	0.017	0.0271	0.0549
implicit	0.0122	0.5793	57.719

Table 3.3: Computation Time (s) with $N_u = 5$ and $N_t = 5$

N_m	**1**	**3**	**5**
iterative	0.0068	0.0068	0.0068
bundle	0.0164	0.0217	0.231
implicit	0.0093	0.0181	0.0244

Table 3.4: Computation Time (s) with $N_u = 20$ and $N_t = 5$

N_m	**1**	**3**	**5**
iterative	0.0316	0.0317	0.0318
bundle	0.0487	0.0579	0.0646
implicit	0.0119	0.0359	0.0586

were thus performed with $N_u = 10$ and $N_t = 30$ using network topologies of varying diameter (Figure 3-22). The CBBA was shown to have a much faster convergence times than the iterative approach, mostly due to its ability to resolve conflicting tasks in parallel.

3.4.3 Scalability

One of the main advantages of most auction algorithms over optimal planners is the efficiency in which assignments can be made. To calculate the optimal solution, the implicit coordination algorithm must first list all permutations (with some pruning), evaluate them, and then perform the optimization. Tables 3.2 through 3.4 show the computation times used to execute the different algorithms. For low horizons, or a small number of tasks, the tables show that the optimal solution can easily be found. However, when the task list is large, the optimal solution scales very poorly with the number of tasks and the benefit of using an auction strategy to minimize computation becomes apparent. Figures 3-23 and 3-24 plot the convergence time as a function of the number of tasks for $N_m = 3$ and $N_m = 6$ respectively. Note that the plots are on a log scale.

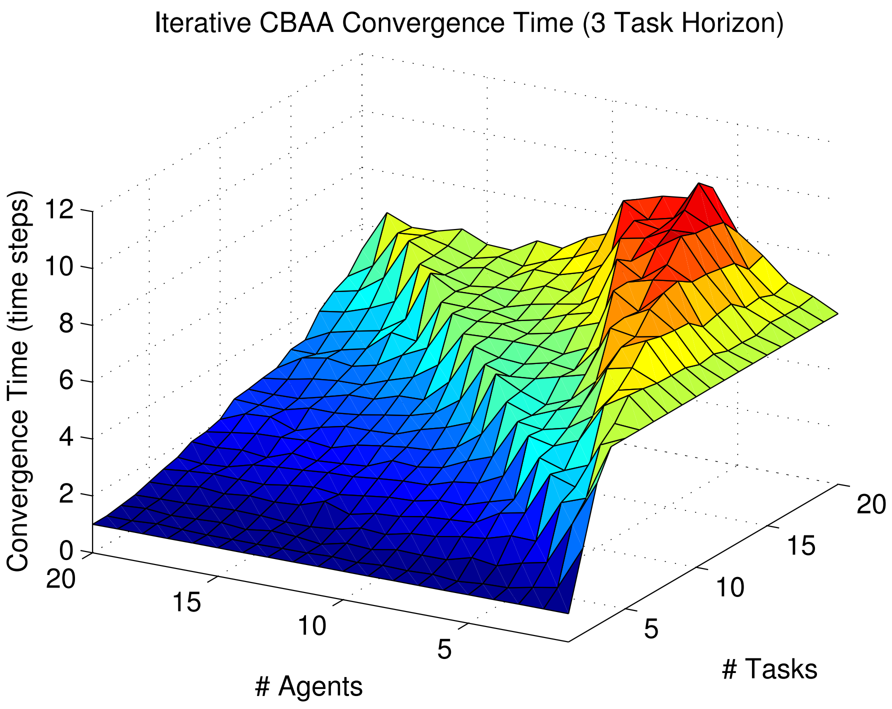


Figure 3-20: Convergence Time for Iterative CBAA $D = 1$

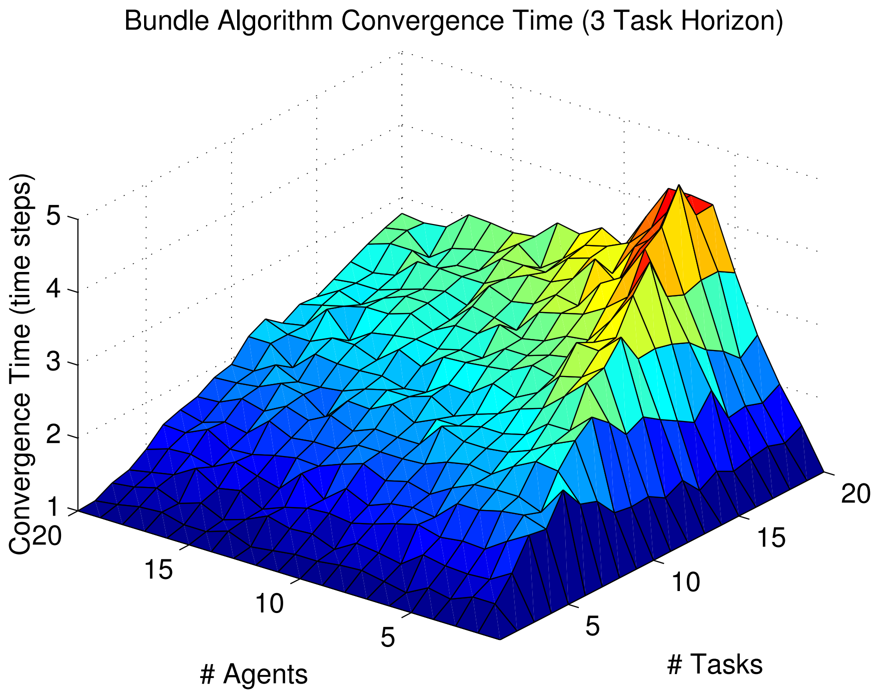


Figure 3-21: Convergence Time for CBBA with $D = 1$

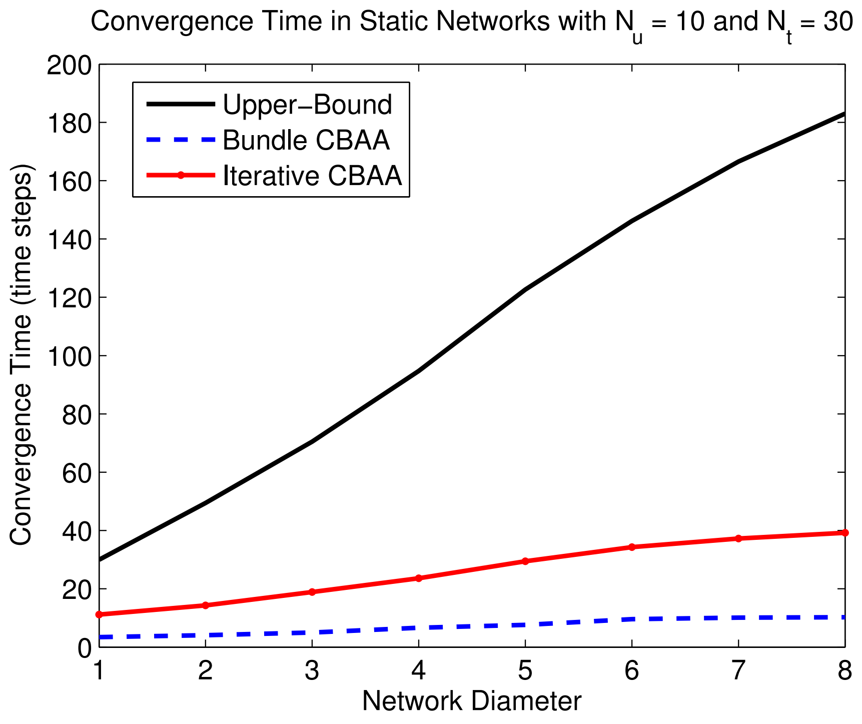


Figure 3-22: Comparison of the Convergence Times with the Upper Bound as a Function of the Network Diameter for $N_u = 10$ and $N_t = 30$

3.4.4 Comparisons with the Prim Allocation Algorithm

The Prim Allocation (PA) algorithm [53], is an sequential centralized auction strategy. Each agent creates a minimum spanning tree (MST) and bids on the task that is closest to any of the nodes in the assignment. This process continues until all of the tasks have been assigned. Tasks are then ordered through the tree by performing a depth-first search (DFS) [96]. The algorithm is designed to minimize the total distance traveled by the fleet to accomplish the tasks, however, other heuristics have been developed in [62, 97] that can be used as well. As previously stated, the PA algorithm is sequential, thus the convergence of both the iterative and bundle strategies presented here are upper-bounded by the PA.

Simulations were done to compare the total distance traveled by the vehicles using the PA algorithm as well as the CBBA. For the latter, the distance wasn't minimized outright. However, the scoring function in (3.12) is based on the agent's distance to a task. This is a good example of how using generic scores when developing algorithms

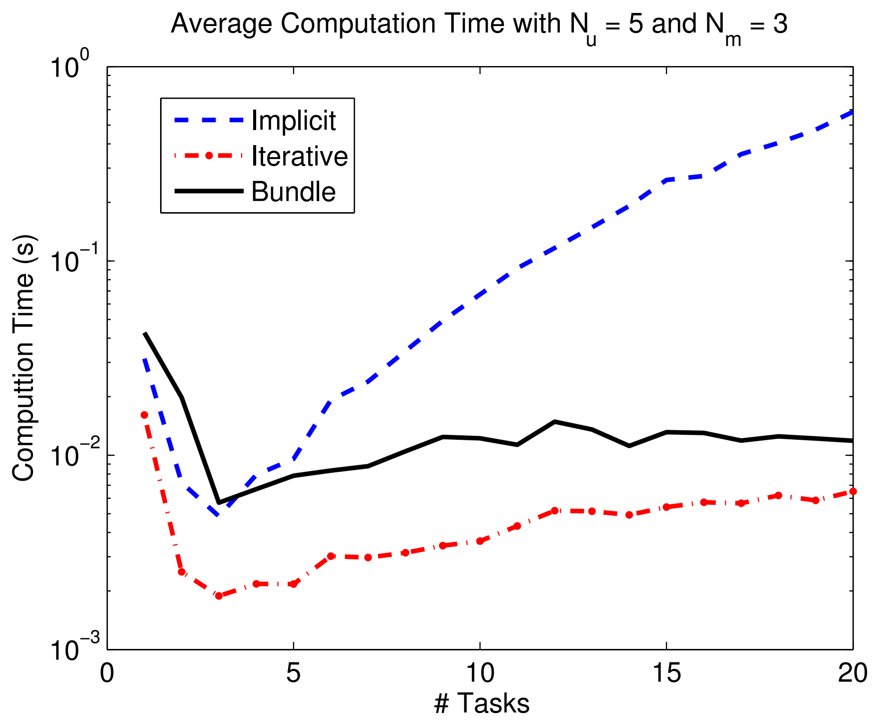


Figure 3-23: Computation Time with a Maximum Assignment Size of 3

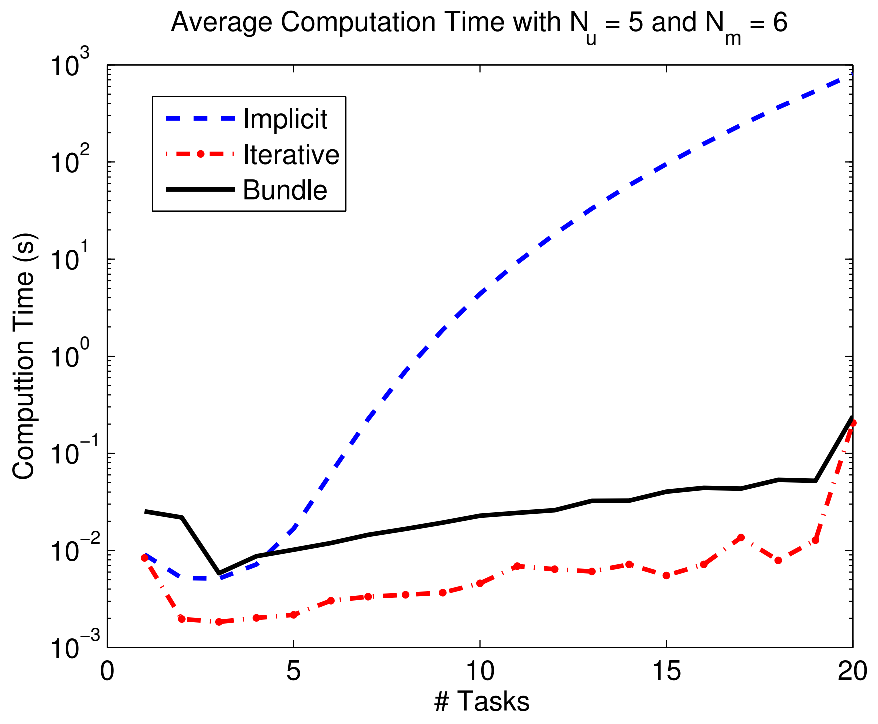


Figure 3-24: Computation Time with a Maximum Assignment Size of 6

can provide more flexibility in terms of objective functions. Figures 3-25 and 3-26 show the total distance traveled with $N_u = N_t$ and $N_u = 5$, respectively. In both cases, the CBBA provides a solution in which the fleet travels less distance than the PA method. Furthermore, convergence time for the PA algorithm was compared against the CBBA using a fully connected graph in Figure 3-27. Since the PA algorithm assigns each task one at a time, the algorithm will be slow for a large number of tasks. This demonstrates the benefit to resolving conflicts in parallel and allowing agents to bid on tasks asynchronously.

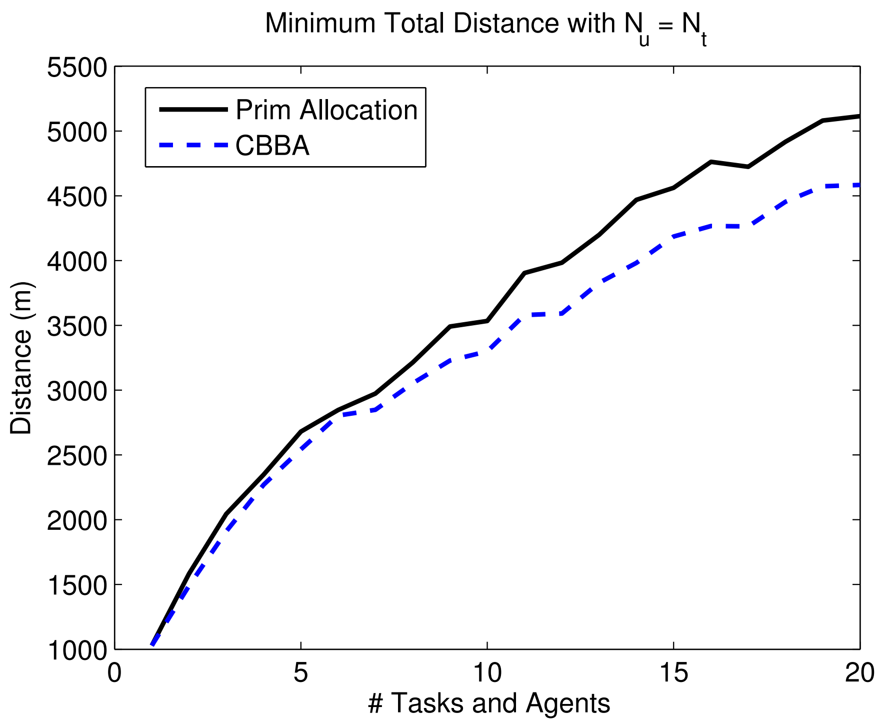


Figure 3-25: Distance Needed for Assignment with $N_u = N_t$

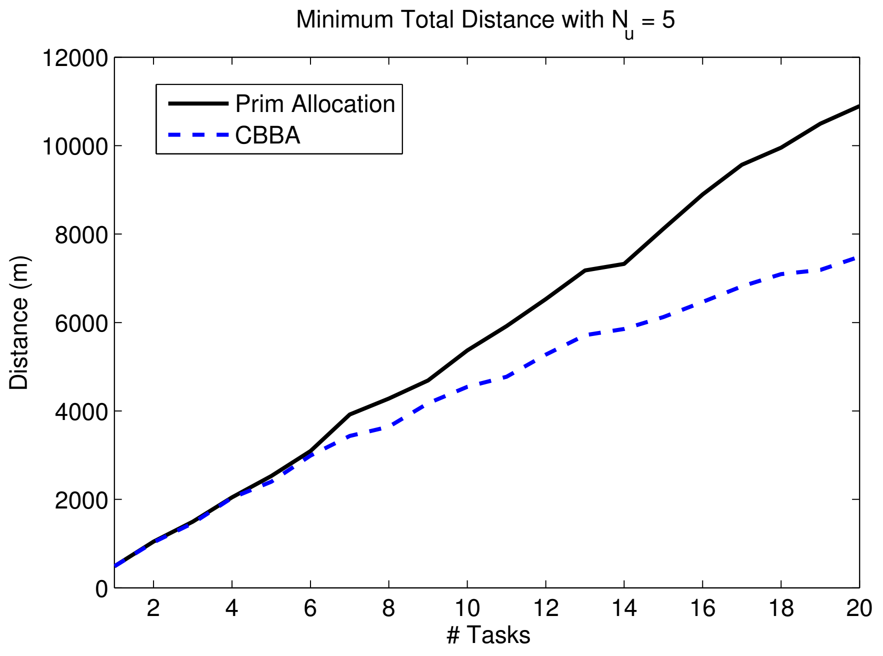


Figure 3-26: Distance Needed for Assignment with $N_u = 5$

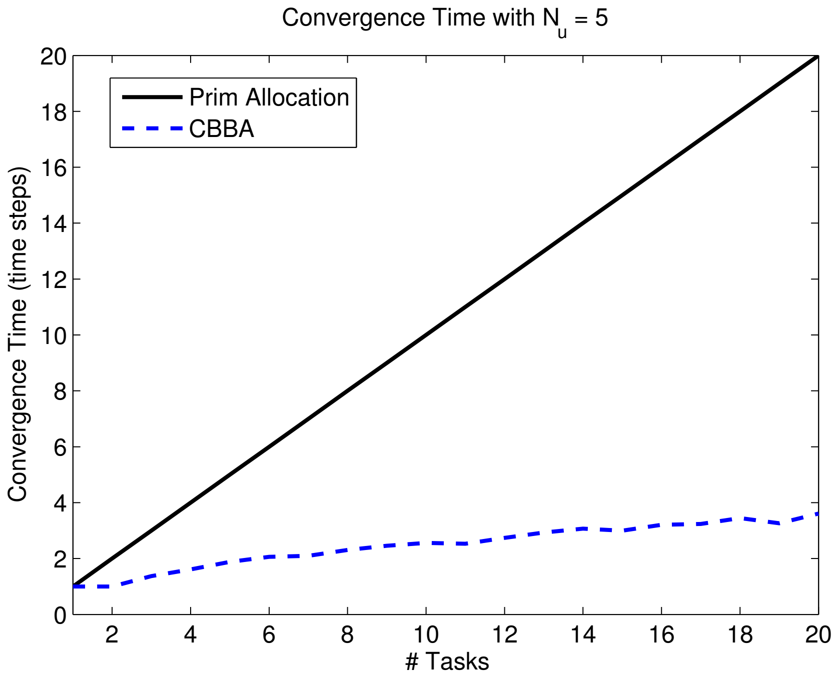


Figure 3-27: Convergence Comparison with $N_u = 5$

Chapter 4

Implementation for Search and Track Missions

Search and track missions have been studied extensively in recent years [13, 17]. In these types of missions, a fleet of autonomous vehicles is given a known or unknown environment that it must search for a specific set of targets. Upon discovery of a target, the vehicles must continue to search the remaining environment while keeping track of the targets that have already been found. There is thus an inherent trade-off between how much search is done and how well the discovered targets are tracked. Vehicles might also be subject to limited fuel, limited capability, and limited communication resources, which can further increase complexity in the problem formulation.

Various methods exist to enable a fleet of autonomous agents to search a given environment. These include pre-generated search paths [78–81], area coverage algorithms [98–100], formation flying [77], and cooperative methods [17, 76]. In many cases, the environment is discretized into cells over which there is a probability distribution of targets [82–84]. Task assignment algorithms are used to allocate track tasks and, in some cases, divide search regions among the vehicles. However, search regions are generally very large and complex. Communication is generally intermittent and noisy, and the network structure can be highly dynamic. At any given instant, the network might even be split into many disconnected sub-networks. Because of this, much of the literature for these types of missions has focused on enforcing strict network topologies, such as a fully connected network [24, 64, 75, 86], or a static connected network with routing capability [43, 53]; other times the assignment is simply done within a local sub-network [58, 59].

The objective of this chapter is to detail the implementation of a task assignment algorithm that will provide the network flexibility that is needed for complex search and track missions. The algorithm is implemented on the CSAT (Coordinated Search, Acquisition and Track) Simulation test-bed that was developed by Aurora Flight Sciences (AFS) in conjunction with MIT. This sophisticated simulation environment allows developers the opportunity to implement real-time algorithms in a controlled environment that can emulate many of the real-world complexities that may exist. Currently, the simulation uses the Robust Decentralized Task Assignment (RDTA) algorithm to coordinate vehicles in the environment. In this chapter, a continuous Consensus-Based Bundle Algorithm (CBBA) will be described and compared against RDTA. Simulations show that the CBBA provides quicker response times to newly discovered tasks, as well as the ability to successfully keep track of more discovered targets. It will also be shown that the CBBA has a better mission performance for combined search and track, and is able to make efficient use of increased communication to improve this value.

4.1 CSAT Mission

The CSAT mission involves three main tasks: search, acquisition, and track. A group of unmanned vehicles cooperate and periodically perform revisit tasks for discovered targets, broad-area searching that can include map building if the environment is unknown, and close-in viewing of specific targets for classification. In these situations, communication can also be of concern since the environments that are being searched can be highly cluttered and may span very large distances causing intermittent communication links. Bandwidth will also be limited which will further reduce the ability for agents to communicate. Thus, it is necessary that the algorithms that

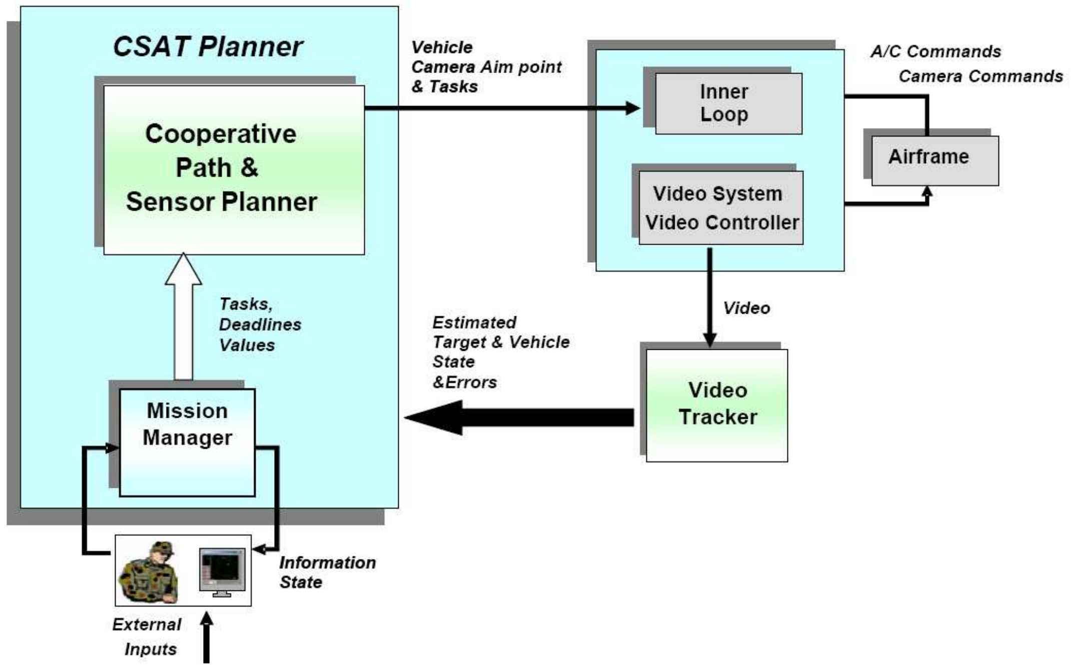


Figure 4-1: CSAT Mission Architecture - image taken from [2]

are developed be robust to these highly unpredictable scenarios. Furthermore, the fleet should perform the mission in a decentralized fashion to improve the coverage area and to avoid single points of failure.

Figure 4-1 shows the CSAT onboard architecture that is used by the vehicles. A mission manager takes in the mission information state and, possibly, information from a human interface, and generates a list of tasks with associated completion times. The cooperative planner takes this list and cooperates with the other vehicles to agree on an assignment, and creates feasible trajectories to ensure maximal performance. The planner can take into account high valued search areas, obstacles, vehicle dynamics, and refueling constraints when generating the assignment and trajectories. A list of way-points and arrival times is then sent to vehicle controllers to execute the plan. A vision module is used as the vehicle's main sensor and is located on a separate processor. It relays target estimates back to the mission manager to complete the loop.

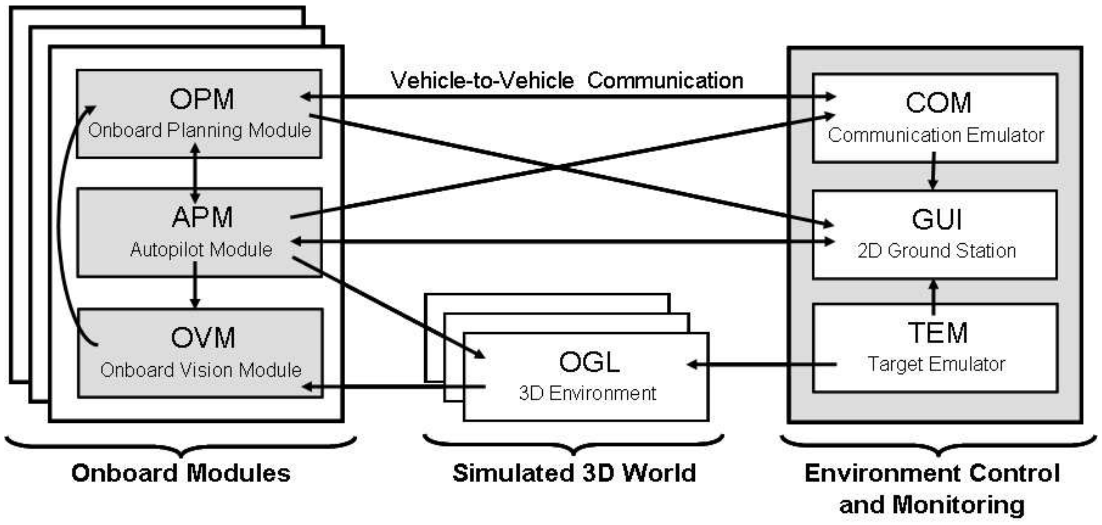


Figure 4-2: CSAT Simulation Architecture

4.1.1 Simulation Environment

The full CSAT simulation architecture is shown in Figure 4-2. It is divided into three separate areas: the onboard modules, the 3-D environment, and environment monitoring and control modules.

Onboard the individual vehicles there are three separate modules, the first of which is the onboard planning module (OPM). The OPM provides high level autonomy to each vehicle. It communicates with the other vehicles to coordinate assignments and share information, while passing its state information to the ground station for monitoring. It should be noted that the ground station is completely passive and is not used for anything other than viewing the mission progression. Trajectories are created in the OPM and passed to the autopilot module (APM) for execution. The autopilot is based on guidance and navigation algorithms developed in [101], with dynamics based on a simple double integrator with rate limits. The APM can also act as an interface to commercial autopilots for flight testing or hardware in-the-loop simulations (HILSim). Finally, the OPM uses a vision module (OVM) as its main sensor for target detection and tracking. The OVM module has two modes of operation. In the emulation mode, the module receives target ground truth from the target emulator and passes state estimates to the OPM if the target is within the field of view. This allows for a controlled environment to test the effects of specific levels of noise and error in the estimates. In the second mode, sensor images are read in from the 3D environment and vision software is used to extract target estimates.

The second section of the CSAT simulation environment is the OpenGL 3D environment that the agents and targets have the option of plugging into (Figures 4-3 and 4-4). The environment reads state information from the APM and TEM modules, and generates the visualizations of each in 3D. Multiple viewing modes exist, including a sensor footprint view which can be used to extract images as input to the OVM. If hardware testing is needed, it is also possible to use a real camera pointed at the screen for image extraction.

Finally, the last area in the simulation is for environmental control and monitoring. Here, three modules are presented that emulate the environment for the onboard algorithms. The first is the communication emulator (COM). All messages between agents are routed through this module. Based on the location of the agents, the emulator adds a specific amount of delay to each message that is calculated using the vehicle transmission power, range and channel bandwidth. Noise and packet loss can also be added, however for these simulations, it was assumed that upon transmission, messages are received without error. If there is no direct connection between agents, the emulator tries to route the message using an $A^{\star}$ search algorithm [102] through the fleet to simulate broadcasting. If no route is available, the message is not sent. Targets in the simulation are controlled from the target emulator (TEM). They can either start at random or predefined locations, but then travel along predefined routes in the environment, choosing random directions at each route intersection. The position of each target is sent to the OVM and OGL modules when active. It is possible to alter target speeds and behavior, although for these simulations, the targets moved at a constant velocity and were non-evading.

The final module in this section is the passive ground station (GUI). This module displays the ground truth of the entire mission and allows viewers the chance to see, at a high level, how the cooperative algorithms are behaving (Figure 4-5). It can also show the probability distribution of the targets that the agents use for searching

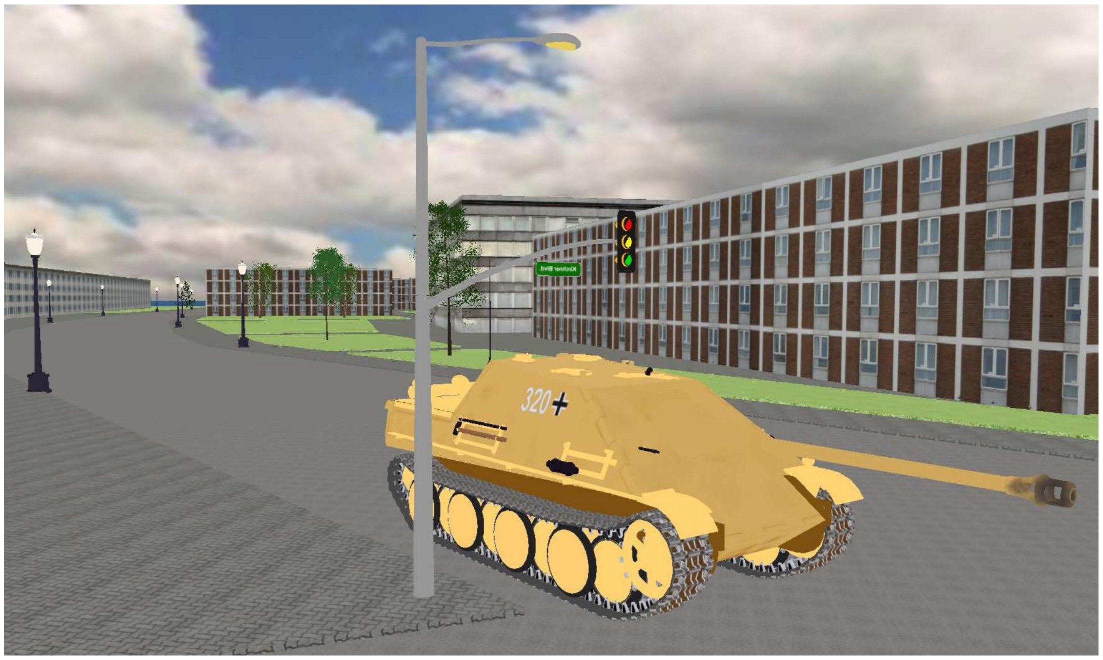


Figure 4-3: CSAT Simulation OpenGL 3D Environment

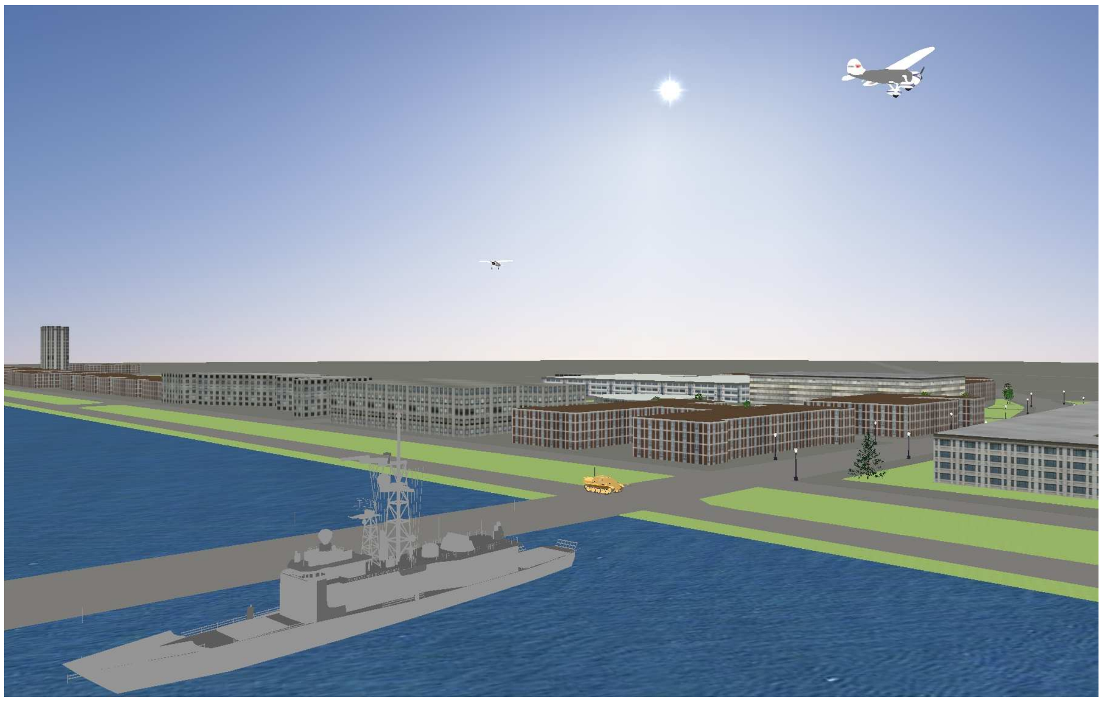


Figure 4-4: Vehicles plug into the environment and track targets using the vision module

the environment (Figure 4-6), and monitor the agents' health status, plans, and communication statistics for each vehicle.

4.2 Onboard Planning Module

The Onboard Planning Module is the main engine in the simulation for the cooperative search and track framework. This section will outline some of the important features it has that will affect how the CBAA will be implemented.

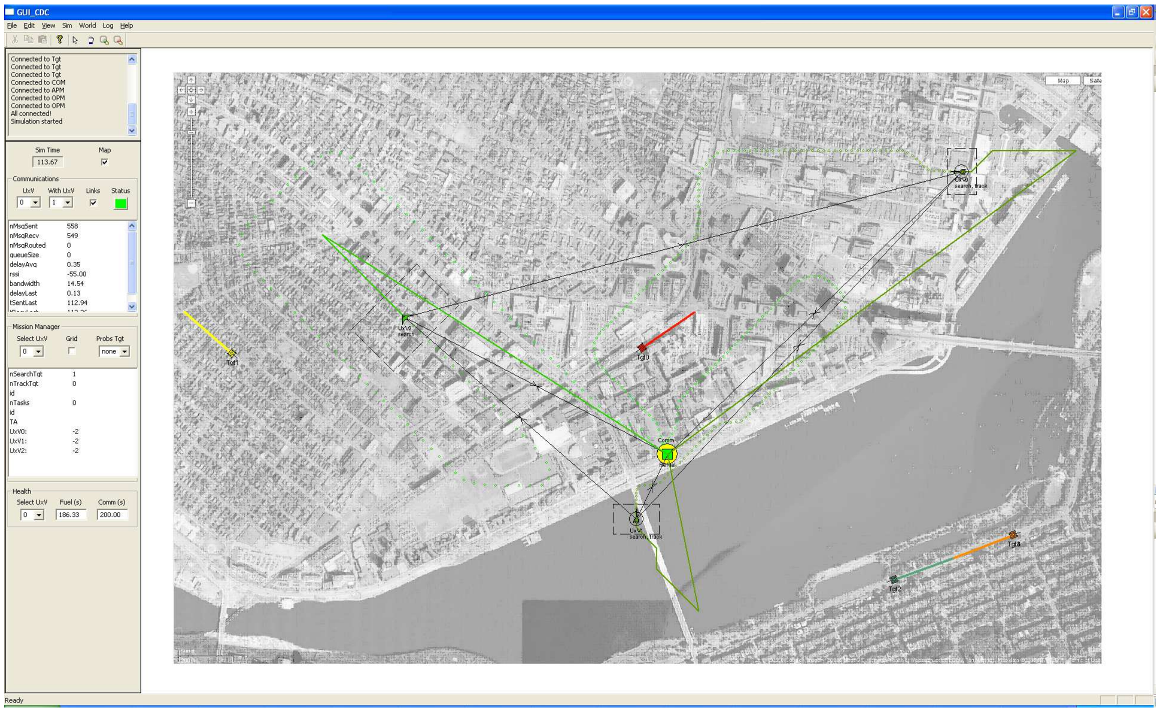
4.2.1 Searching the Environment

Environments in the CSAT simulator are created offline and discretized into an $N_x \times N_y$ grid of $N_c = N_x \cdot N_y$ identically sized cells. The cells are used to discretize the probability distribution of the targets, as well as for path planning and obstacle avoidance. Two types of grid maps exist: search maps and an environment map. The environment map is initialized to some distribution of targets that is known *a priori*, where P_i^0 is the probability of finding a previously unknown target in cell $i \in \{1, \ldots, N_c\}$. Search maps represent the probability distributions $P_i^k,\ \ k \in \{1, \ldots, N_s\}$ for targets that are known or thought to exist, where N_s is the current number of search maps. Search maps can be created from targets that were found but have since been lost, or from a target has been found but an accurate state estimate has yet to be received. There are thus a total of $N_s + 1$ sets of distributions. At each time step, the probabilities are updated as follows:

$$P_i[n+1|n] = \begin{cases} 0 & \text{if } i \rightarrow \textit{visited at } n \\ AP_i[n|n] & \text{otherwise} \end{cases} \tag{4.1}$$

and normalized to

$$P_i[n+1|n+1] = \frac{P_i[n+1|n]}{\displaystyle\sum_{i=1}^{N_c} P_i[n+1|n]} \tag{4.2}$$



```
Connected to Tgt
Connected to Tgt
Connected to Tgt
Connected to COM
Connected to APM
Connected to OPM
Connected to OPM
All connected!
Simulation started
```

Sim Time: 113.67 Map: ☑

Communications — UxV: 0 With UxV: 1 Links: ☑ Status: 🟩

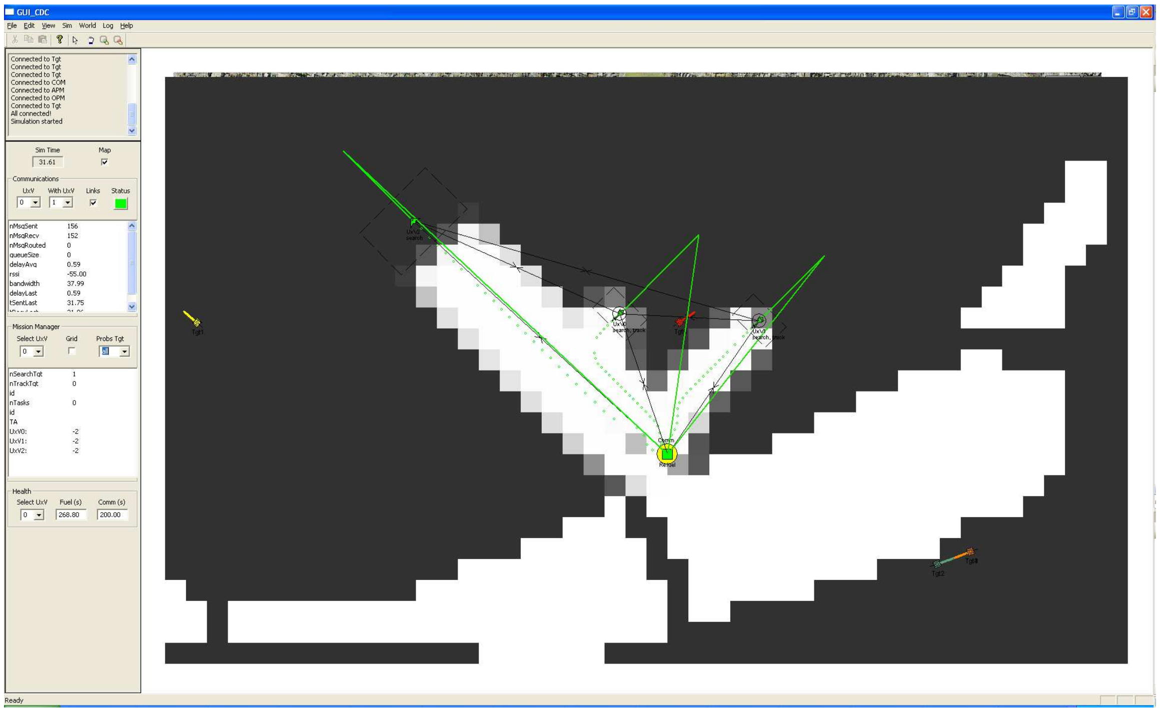
```
nMsgSent     558
nMsgRecv     549
nMsgRouted   0
queueSize    0
delayAvg     0.35
rssi         -55.00
bandwidth    14.54
delayLast    0.13
tSentLast    112.94
```

Mission Manager — Select UxV: 0 Grid: ☐ Probs Tgt: none

```
nSearchTgt   1
nTrackTgt    0
id
nTasks       0
id
TA
UxV0:        -2
UxV1:        -2
UxV2:        -2
```

Health — Select UxV: 0 Fuel (s): 186.33 Comm (s): 200.00

Figure 4-5: CSAT Simulation Ground Monitoring Station



```
Connected to Tgt
Connected to Tgt
Connected to Tgt
Connected to COM
Connected to APM
Connected to OPM
Connected to Tgt
All connected!
Simulation started
```

Sim Time: 31.61 Map: ☑

Communications — UxV: 0 With UxV: 1 Links: ☑ Status: 🟩

```
nMsgSent     156
nMsgRecv     152
nMsgRouted   0
queueSize    0
delayAvg     0.59
rssi         -55.00
bandwidth    37.99
delayLast    0.59
tSentLast    31.75
```

Mission Manager — Select UxV: 0 Grid: ☐ Probs Tgt: all

```
nSearchTgt   1
nTrackTgt    0
id
nTasks       0
id
TA
UxV0:        -2
UxV1:        -2
UxV2:        -2
```

Health — Select UxV: 0 Fuel (s): 268.80 Comm (s): 200.00

Figure 4-6: Ground Station View of the Cell Probabilities for Target Detection

where A is the transition matrix

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,N_c} \\ a_{2,1} & a_{2,2} & \dots & a_{2,N_c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N_c,1} & a_{N_c,2} & \dots & a_{N_c,N_c} \end{bmatrix} \tag{4.3}$$

with element $a_{i,j}\ \ \forall i, j \in \{1, \dots, N_c\}$ defined as the probability of a target of moving from cell i to cell j at any time. The value of a cell $\mathcal{V}_i$ can then be defined as

$$\mathcal{V}_i = C_s \sum_{k=0}^{N_s} P_i^k \tag{4.4}$$

where $C_s > 0$ is the static value associated with searching a cell.

The environment map cells are also used for path planning and obstacle avoidance. Four different environment types are defined: air, ground, water, under-water. For each cell in the environment map, each environment type is indicated as either *free* or *obstructed*. Each target is assigned an environment classification, and the transition function in (4.3) is designed to reflect this. For example, a target that is designated as being in the water environment type, will only have $a_{i,j} > 0$ if both cells i and j have a *free* water environment classification, and $a_{i,j} = 0$ otherwise. It should be noted that a given cell might have multiple *free* environment types. These environment types can also be used for path planning. Agents can use the *free* or *obstructed* cell information to plan safe paths through the environment to complete the mission. Thus, safe paths can be generated that can maximize the detection probability, in a general discretized environment framework.

4.2.2 Target Tracking

As the environment is searched, the discretized probability distribution is updated and the transition function propagates this information to the neighboring cells. Once an updated cell probability meets some pre-determined threshold $P_i > \alpha$, the search map is removed and a track task is created for that target. Let the target state be defined by

$$\hat{x} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \tag{4.5}$$

where (x, y) is the target position and (v_x, v_y) the velocity. At each time step, if the target is in the agent's field of view, the state is updated with the vision estimate. If not, the state is updated using a constant velocity model with

$$\hat{x}_{k+1} = A_{targ}\hat{x}_k \tag{4.6}$$

where,

$$A_{targ} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

with covariance

$$\Sigma_{k+1} = A_{targ}\Sigma_k A_{targ}^T + Q_{targ} \tag{4.8}$$

Once the target is tracked, the revisit time and location can be determined based on the area of the covariance ellipse which can be calculated using

$$area = \pi \cdot \det(\Sigma_k) \tag{4.9}$$

The OPM projects the target's current state into the future, and stops when the area of the uncertainty ellipse has reached the area of a cell. That place and time is then set as the revisit location. A radix heap implementation of Dijkstra's algorithm [103] is used to calculate the shortest path from the revisit location to every cell in the environment. This is used to ensure that at each time step an agent can arrive at the task on time while avoiding obstacles.

4.2.3 Robust Decentralized Task Assignment

As tasks are created in the simulation environment, a task assignment algorithm is needed to divide the task set amongst the members of the fleet. In the OPM, planning is done periodically using the Robust Decentralized Task Assignment (RDTA) algorithm [43, 44]. RDTA is a two-stage task assignment algorithm that is robust to inconsistencies in the SA of the fleet. The assignment is done periodically, while agents use consensus strategies to converge on the SA in between assignment periods.

The assignment algorithm works in two stages. In the first stage, each agent creates a list of all possible permutations $\mathcal{P}$ of N_v tasks, and uses the petal algorithm [14, 104] to create ρ_i candidate plans by performing the single optimization of (4.10) ρ_i times

$$\begin{aligned} \max_{x_k} \sum_{k \in \mathcal{P}} S_k x_k & \\ \text{subject to} \quad \sum_{k \in \mathcal{P}} \sum_{j=1}^{N_v} V_{jk} x_k & \leq 1 \\ x_k \in \{0, 1\} & \end{aligned} \tag{4.10}$$

where S_k is the score achieved by selecting petal k, and $x_k = 1$ if petal k is selected. At each optimization round, the selected assignment $k^\star$ is removed from the list of possible plans for the next iteration.

In the second stage, each agent i sends its ρ_i plans $\mathcal{M}_i = \{k_1^*, k_2^*, \ldots k_{\rho_i}^*\}$ to each of the other agents in the fleet. Once the plans are received, the final optimization in

(4.11) is done over all candidate plans to obtain the final solution.

$$\begin{aligned}
\max_{x_k} \sum_{j=1}^{N_u} \sum_{k \in \mathcal{M}_j} & S_{ik} x_{ik} \\
\text{subject to} \quad \sum_{j=1}^{N_u} \sum_{k \in \mathcal{M}_j} \sum_{n=1}^{N_v} V_{nk} x_{jk} & \leq 1 \\
\sum_{j=1}^{N_u} x_{jk} & = 1 \\
x_k \in \{0, 1\} &
\end{aligned} \tag{4.11}$$

The final stage of optimization in the RDTA algorithm is based on consistent plans, which will ensure that there are no conflicts in the final assignment, even if the SA of the vehicles are slightly different. However, the algorithm requires a connected network in order to ensure that the candidate petals are received by each agent. This assumption is not always achieved in search and track missions, thus a timeout was implemented in the CSAT simulation to allow the second stage optimization to continue if the network is disconnected. This however comes at the risk of conflicts in the final assignment, and might significantly increase planning times as the timeout is increased. To avoid lengthy delays, each agent maintains an estimate of the network connectivity and maintains a list of agents that it is connected to either directly, or through other agents, in the network. As the task assignment is executed, each agent only activates the petal reception timeout for other agents it presumes to have some connection with. This significantly reduces the waiting time for petals presumed not to arrive.

4.3 Consensus-Based Bundle Algorithm Implementation

In the previous section, the search, track, and task assignment mechanisms for the OPM were introduced. The RDTA algorithm can provide assignments despite inconsistencies in the SA, with the limitation that vehicles must ensure that the network is connected when the petals are passed. Without this, a timeout must be used and a consistent SA can no longer be guaranteed. Upon reconnection of the network, or the reception of new information, the RDTA algorithm must redo the entire optimization in order to provide a new assignment, or resolve any possible conflicts. This will reduce its reaction time to new targets and possibly degrade performance.

In this section, a continuous planning scheme for the CBBA will be developed. As shown in the previous chapter, the CBBA will maintain convergence under varying network topologies since conflicts will be resolved through the network or rapidly upon reconnection. The algorithm will also not require the use of timeouts to provide a solution when the network is disconnected, and will converge regardless of inconsistencies in the SA. The continuous planning implementation will be shown to have much faster reaction times as new tasks are added, since it does not wait for a tasking period to edit assignments. It also does not need to redo the optimization in order to incorporate the new information into the assignment; it simply adds it to the current plan. Finally, enumerating all of the petals in the RDTA is computationally intensive, thus, the petal sizes are limited to 3 tasks. The bundle algorithm implementation will be able to handle many more targets per agent, and thus, once targets are discovered, their locations will be tracked much more efficiently.

4.3.1 Continuous Planning Scheme

The CBBA algorithm was implemented in the CSAT mission simulator as a continuous planning scheme. At each time step, each agent first checks to see if there are any available tasks that it can perform

$$H_i = (c_{ij} > y_{ij}) \wedge K_i, \quad \forall j \in \{1, \ldots, N_t\} \tag{4.12}$$

and adds as many as it can to its bundle. Communication with neighboring agents is done continuously to resolve any conflicts that arise. Search is done when an agent has excess time before visiting the next task. In this case, trajectories are created to visit the cells with high task probabilities, while ensuring that the agent stays within range of the task location in order to visit it on time. In the absence of new targets, the assignment remains fixed and each agent executes its current plan. Upon discovery of a new target, the assignment is perturbed as agents add the new task and the conflict is resolved. The resolution time of this assignment perturbation is correlated to the communication period as agents must converge on the winning bids list to remove the conflicts. Thus, for the CSAT implementation, the task assignment algorithm was placed on a separate thread which ran at a much higher rate than the main thread. In these cases, convergence was generally achieved within a single time step of the main loop.

In the CSAT mission, tasks need to be performed at a specific time or within some time window. Thus, the scoring metric used in the implementation is different than the time discounted values used in previous chapters. Figure 4-7 shows a general scoring function $f(t)$ for a task i, where S_i is maximum attainable score, P_E and P_L are the penalties per second for being early or late to the task, t_{MIN} and t_{MAX} are the earliest and latest possible arrival times, and T_{R_i} is the desired arrival time. $f(t)$ is then calculated using

$$f(t) = S_i - \begin{cases} P_E(T_{R_i} - t) & \text{if } t_{MIN} \leq t \leq T_{R_i} \\ P_L(t - T_{R_i}) & \text{if } t_{MAX} \geq t > T_{R_i} \end{cases} \tag{4.13}$$

These parameters can be modified to account for almost any task type. The value of S_i is based on a task's classification, thus, all agents capable of arriving on time will have the same task score. To break the tie, the earliest arrival time $T_{E_{ij}}$ is also passed with the bid. This is the earliest possible time that agent i is able to arrive at task j while ensuring that all of the other tasks in its bundle are visited at the desired times. If a scoring tie occurs, the agent with the lower $T_{E_{ij}}$ is assumed to have more room for error in arriving at the task on time and is thus awarded the task. This also adds a level of robustness to the assignment by awarding tasks to vehicles with more flexibility in their assignment.

As new tasks are introduced into the system, the agents react by trying to place them in the best possible configuration into their bundle. Unlike the RDTA algo-

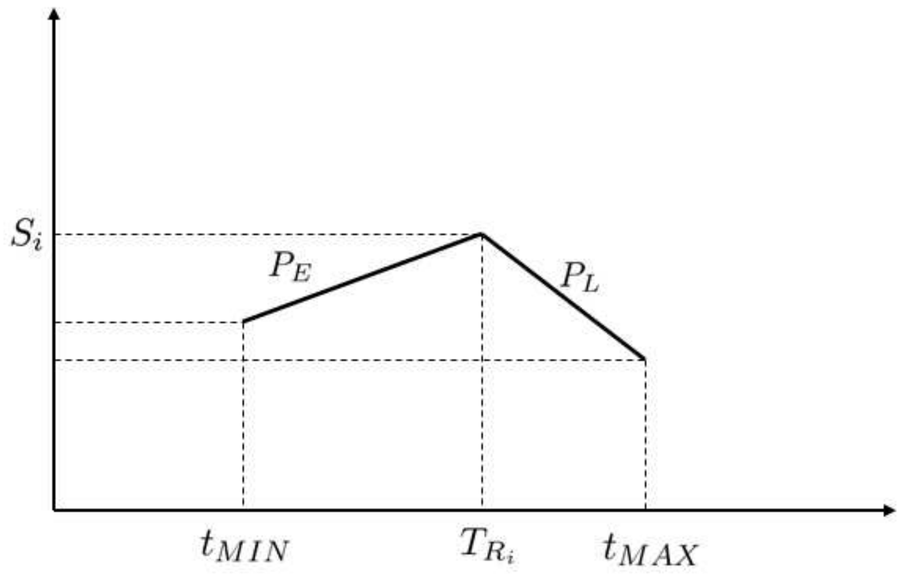


Figure 4-7: Task scoring with timing constraints $t_{MIN} \leq t \leq t_{MAX}$

rithm where the entire optimization would need to be re-done, the CBBA planner was designed to be more reactive to changes in the environment. Furthermore, a disconnected network will still create conflicts, but they will be resolved as the network continues to vary and eventually becomes reconnected. With the RDTA approach, the fleet must wait for the next optimization period and hope that conflicting agents can route their petals at that time.

However, problems might still occur with the CBAA as task information changes while the network is disconnected. Because of this, a task history list is kept by each agent. An entry is made for each discovered target with information indicating the task version number, the time the information was last updated, and whether the task is active or not. Upon each successful track of a given target, its version number is increased by one and the update time is updated accordingly. The active variable indicates that a location estimate for the target exists, and that there is a track task for that target currently in the task list. Agents communicate this information in order to coordinate the task lists. If an agent realizes that it has old information, a request is sent to the nearest neighboring agent to transmit the latest target and associated task information.

This continuous planning scheme is designed to be reactive to new information in the environment, and resolve conflicts quickly under varying network topologies. It is possible, however, that as the mission progresses the assignment may degrades slightly since agents do not update their bundles unless they are outbid. To accommodate this, a periodic re-plan might be desired in which a new assignment is calculated periodically based on the current information. This can easily be enabled in the continuous CBAA implementation by simply having each agent release each task at the desired assignment period. Agents will then try and add each task to the bundle and calculate new assignments based on the updated information.

4.3.2 Task Constraints

The value in adding a task to the current plan is the change in score that the agent will achieve with the addition of that task to the bundle. Track tasks however generally have a specific time in which they should be performed, and thus a time discounted scoring system might not be the best choice. Furthermore, agents in the CSAT mission are subject to periodic refueling and must sometimes communicate with a ground station in order to unload the collected data. This section will develop the scoring function used, and look at how Mixed-Integer Linear Programming (MILP) can be used to calculate the assignment score under various temporal constraints. For the CSAT implementation, the GNU Linear Programming Kit (GLPK) [105] was used to calculate the optimal arrival times given a task list.

Timing Constraints

In the CSAT missions, task scores are calculated using (4.13). When the CBBA inserts a task to its bundle, it tries to insert it at every location to find the insertion point that maximizes the total score. However, inserting it may alter the timing of other tasks, and thus it is important that an accurate scoring calculation be readily available. Suppose that an agent k has a task list $L = \{l_1 \dots l_{N_i}\}$, and the goal is to find the optimal time to arrive at each task given $f(t_i)$, $i \in L$, the period ΔT_i an agent must be at a task to perform it (i.e. tracking for a certain amount of time), the distance d_{i+1} from task i to task $i+1$ in the list, and the agent's velocity v_k. The problem can be solved using the following LP:

$$\begin{aligned}
& \max \sum_{i=1}^{N_i} Z_i \\
\text{s. t. } \forall\, i \in L: \quad & S_i + (t_i - T_{R_i})P_E && \geq Z_i \\
& S_i - (t_i - T_{R_i})P_L && \geq Z_i \\
& t_{i+1} - t_i && \geq \Delta T_i + \tfrac{d_{i+1}}{v_k} \\
& t_i && \leq t_{MAX} \\
& t_i && \geq t_{MIN}.
\end{aligned} \tag{4.14}$$

For each task insertion point in the current assignment, (4.14) is used to calculate the score. The insertion point for the maximal score achieved is kept and the task is inserted at that point, while the arrival times of each task are updated to reflect the new assignment.

Communicating with a Ground Station

There may exist scenarios in which agents must periodically perform a specific task during the mission. An example of this is a periodic communication with a ground station, however it can also be used for refueling, periodic surveillance, etc.... The scoring function will remain the same as in the previous section, however, the goal is now to find the optimal task timing and ground station visiting locations such that communication is made with a ground station every T_{COMM} seconds. The scoring and timing constraints of (4.14) will still be in effect, however, new constraints will be added into the optimization to achieve the assignment goals.

Before the optimization is performed, a tentative communication point is inserted into each point in the task list (Figure 4.3.2). Let $b_i \in \{0,1\}$ be a binary variable such that $b_i = 1$ if the ground station should be visited at that point along the path, and $b_i = 0$ if not. q_{i+} and q_{i-} will be the last time communication was made,

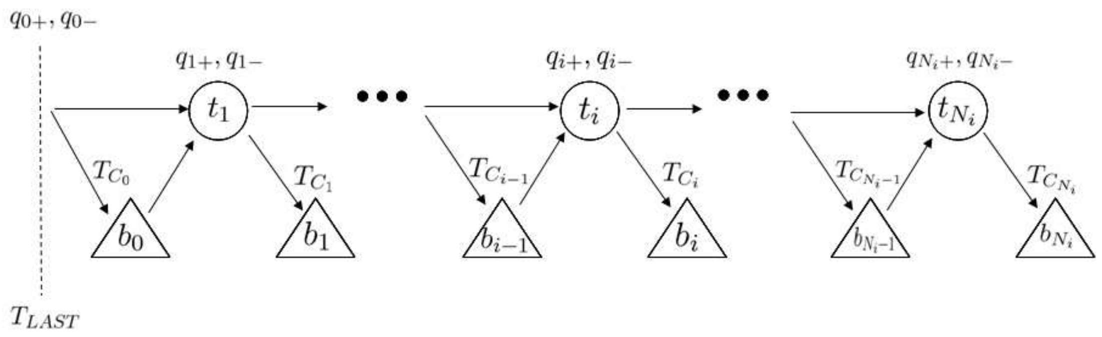


Figure 4-8: Set of tasks with possible ground station communication in between each one.

and the previous time communication was made with the ground station, so that $q_{i-} - q_{i+} \leq T_{COMM} \;\; \forall \; i$. Also, T_{C_i} is the time it takes to get into communication range from the task i, T_{LAST} is defined as the last time the agent communicated with the ground station before the optimization was performed, and M is some very large number used in the MILP.

The problem can be formulated as a MILP as follows

$$\max \sum_{i=1}^{N_i} Z_i$$

s. t.

$$\begin{aligned}
&q_{0-} &&\le T_{COMM} + q_{0+} &&\forall\ i &&\in \{l_2 \ldots l_{N_i-1}\} \\
&q_{0-} &&\le T_{C_0} + M \cdot (1 - b_0) &&q_{i-} &&\le T_{C_i} + M \cdot (1 - b_i) \\
&q_{0-} &&\le q_{1-} &&q_{i-} &&\le q_{(i+1)-} \\
&q_{0-} &&\ge q_{1-} - M \cdot b_0 &&q_{i-} &&\ge q_{(i+1)-} - M \cdot b_i \\
&q_{0-} &&\ge T_{C_0} - M \cdot (1 - b_0) &&q_{i-} &&\ge T_{C_i} - M \cdot (1 - b_i) \\
& && &&q_{i+} &&\ge T_{C_{i-1}} - M \cdot (1 - b_{i-1}) \\
&q_{N_i-} &&\le T_{COMM} + q_{N_i+} &&q_{i+} &&\ge q_{(i-1)+} \\
&q_{N_i-} &&\le T_{C_{N_i}} &&q_{i+} &&\le q_{(i-1)+} + M \cdot b_{i-1} \\
&q_{N_i+} &&\ge T_{C_{N_i-1}} - M \cdot (1 - b_{N_i-1}) &&q_{i+} &&\le T_{C_{i-1}} + M \cdot (1 - b_{i-1}) \\
&q_{N_i+} &&\ge q_{(N_i-1)+} && && \\
&q_{N_i+} &&\le q_{(N_i-1)+} + M \cdot b_{N_i-1} && && \\
&q_{N_i+} &&\le T_{C_{N_i-1}} + M \cdot (1 - b_{N_i-1}) && &&
\end{aligned} \tag{4.15}$$

If a task in the plan is within communication range, the optimization can be simplified by setting

$$q_{i-} = q_{i+} \tag{4.16}$$

This ensures that the time in between communicating with the ground station is separated by no more than T_{COMM} seconds. An assumption in this formulation is that the desired arrival time between tasks is less than T_{COMM}. If this is not the case, the formulation should be loosened to allow for violation of the T_{COMM} limit, and move the requirement to the trajectory planner to ensure that the constraint in between task visits is satisfied.

4.4 Results

Numerical results were obtained to compare the periodic RDTA strategy against the continuous CBBA strategy developed herein. Five different metrics were used for comparison: area searched, percentage of targets found, average track time, the mission index, and the response time. Two sets of simulations were performed. In the first set, the number of tasks was held constant at eight, while the communication range of the fleet was increased from a ratio ($\frac{R}{W}$) of 0.3 with respect to the map width W, to $1 \times W$. The second set involved keeping the communication range fixed at $0.6 \times W$ while the number of targets was varied from 2 to 8. Each data point is the average of six separate missions with randomized target locations and trajectories. In each case, the fleet consisted of three helicopters. The missions lasted 900 seconds, while the refuel and ground station communication times were set to 300 and 200 seconds respectively.

4.4.1 Area Searched

The percentage of the map that was searched was calculated for each algorithm. Figures 4-9 and 4-10 show the results with varying communication range and targets respectively. In both cases the amount of search that was done was slightly higher for the CBBA algorithm. This was because the algorithm was quicker to come to a decision on the track tasks, thus, there was generally more slack time that could be allocated to searching the environment than in the RDTA algorithm.

4.4.2 Targets Found

Another search metric that was calculated was the percentage of tasks that were found during the simulation. This reflects how well the agents were able to search the relevant, high likelihood areas of the environment. Figure 4-11 shows the percentage of targets that were found with increased communication range. As in the area searched, the results show that the two algorithms behave approximately the same for searching the environment, and communication has shown no effect.

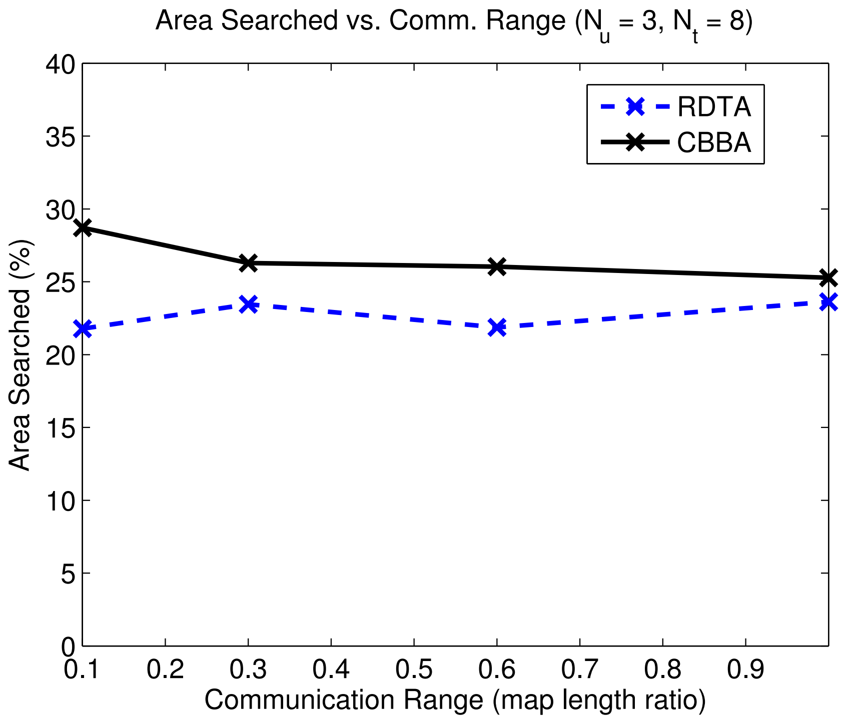


Figure 4-9: The percentage of the area searched stays relatively constant with increased communication

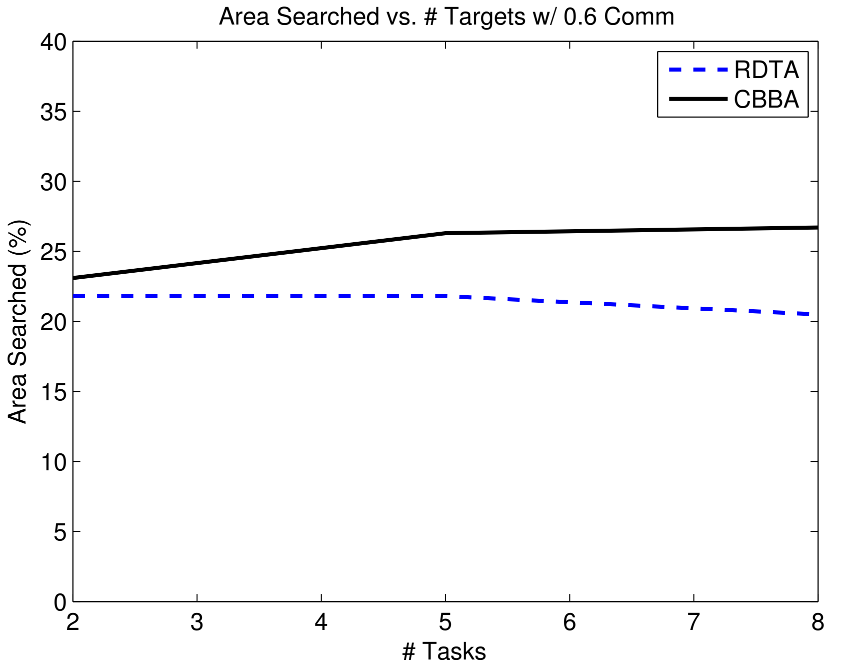


Figure 4-10: The percentage of the area searched with increased tasks in the environment

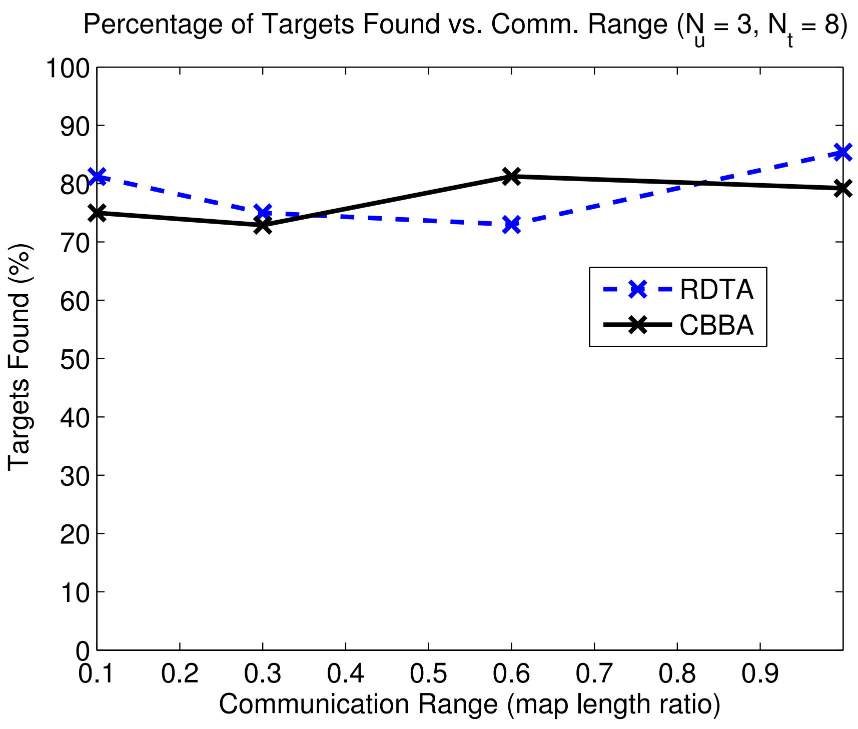


Figure 4-11: The percentage of targets found were approximately equal for both algorithms

4.4.3 Target Tracking

Once a target is discovered, track tasks are periodically performed in order to keep track of their locations. As the revisit rates are increased, the chance of losing track of the target diminishes, however so too does the agent's ability perform any other tasks or search the environment. By revisiting too infrequently, the likelihood of the task being in the estimated location diminishes and targets are more often lost.

The metric used in this section will be the percentage of time the estimated position of a target was accurate after it was first discovered. Let P_{track} be the average percentage of time that a target location was within a certain distance of the estimated position $\sqrt{(x-\hat{x})^2+(y-\hat{y})^2} \leq d_e$, where d_e is a pre-defined distance, (x, y) is the actual target position while $(\hat{x}, \hat{y})$ is the estimated position. P_{track} can then be calculated as

$$P_{track} = \frac{\sum_{i=1}^{N_{disc}} \sum_{\forall\ t>t^i_{disc}} (\sqrt{(x(t)-\hat{x}(t))^2 + (y(t)-\hat{y}(t))^2} \leq d_e)}{N_{disc} \cdot T_M} \tag{4.17}$$

where t^i_{disc} is the time at which task i was first discovered, and N_{disc} is total number of tasks that were found during the mission. This value gives a good indication as to how well the discovered targets were tracked, and penalizes the fleet for tasks that were lost by including those times in the calculation.

Figure 4-14 shows the target track information for different communication ranges. The CBBA tracking performance is much higher than that of RDTA since it is much quicker to make decisions and is therefore more likely to arrive at a task before it is lost. Furthermore, due to the computational efficiency, the CBBA is able to handle more targets per agent than RDTA is, which will also improve its ability to track by not having to wait for a new assignment once it is finished with the current plan. As the communication range is increased from 0.1, the value of CBBA algorithm is able to make use of the increased connectivity to provide better assignments and improve the tracking performance. However, with a fully connected graph (far right), the convergence time becomes more of a factor and the performance slightly decreases. For the RDTA algorithm with low connectivity, agents do not have to wait to receive the bids from the rest of the fleet, thus their reaction time is much faster for lower communication environment. Although the tracking performance is slightly increased when a small amount of communication is added, as it is increased further the convergence time of the algorithm starts to affect the tracking time and the performance slightly degrades.

4.4.4 Mission Value

Although the time-tracked percentage is a good indication of how well the algorithm was able to keep track of discovered targets, it does not give a good indication as to how well it performed the mission. For example, if there were 8 targets, and only one target was found but was tracked the entire time, the percentage tracked would read

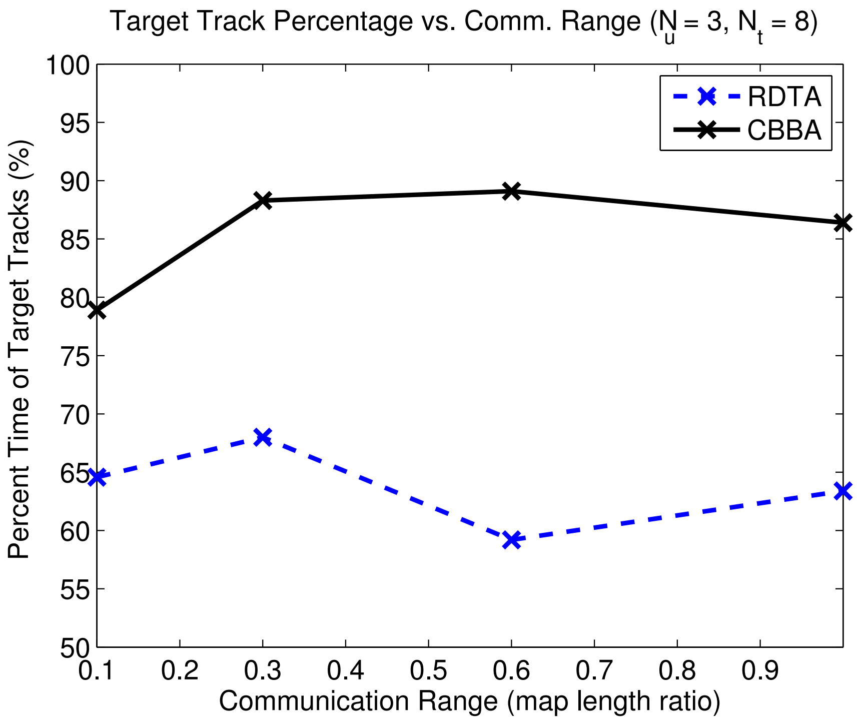


Figure 4-12: The average percentage of time that the estimated position of a target was accurate after it was first discovered

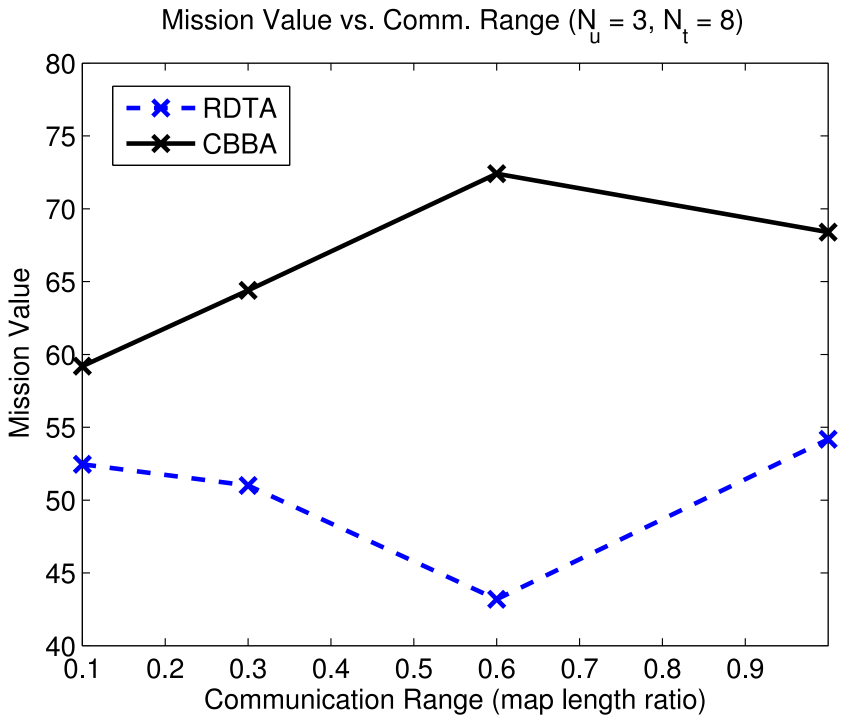


Figure 4-13: The Mission Value (MV) reflects the ability of the fleet to find the targets and keep track of them. A maximum value of 100% is achieved if both were done perfectly.

100%. However, in reality, this was simply a function of the vehicles not searching the environment well enough. This motivates the need for a metric that can combine the track information with search metrics. The metric proposed here is the Mission Value (MV), and will be the defined as

$$MV = P_{track} * \frac{N_{disc}}{N_t} \tag{4.18}$$

where P_{track} is the average percentage of time that a discovered target was within a pre-defined radius of the estimated location, N_{disc} is the total number of targets that were discovered, and N_t is the total number of targets in the environment. Notice that the value of this is maximal when all targets are found and perfectly tracked for the duration of the mission.

Figure 4-13 shows the average MV for both algorithms as a function of the communication range. The CBBA is shown to perform the search and track mission much more efficiently than RDTA can. Furthermore, it is shown that up to a certain communication range, the CBBA algorithm improves, which indicates how it is able to make efficient use of the network structure to improve the mission performance. For RDTA however, the MV actually decreases as communication is increased from 0.1 to 0.6. This is due to the algorithm timeout that was implemented. With very little communication (~ 0.1), RDTA does not wait to receive plans from the other agents and continues with the second stage of optimization right away. This leads to increased reaction time and the fleet is able to visit all of the track tasks, however, conflicts will exist in the assignment. As the communication is increased, the agents knowledge of the network becomes uncertain and agents will be more likely to believe there is a link between other agents when there is not. This initiates the plan reception timeout in the second stage and causes significant delays if the plans are not received. This reduces reaction time and reduces the chance that a task is in the estimated location by the time it is visited. If the communication range is further increased to a fully connected network (~ 1), the knowledge of the network structure is improved and the plans are received quickly and the assignment time is improved.

4.4.5 Response Time

In search and track missions, new targets are continuously discovered and task assignment algorithms should be able to effectively handle this new information. The state estimate may be initially poor or the target may be evading, thus it is important to reduce any delays in performing the new task. This section will discuss the average response time of both algorithms. The metric is based on the average time it took for a newly discovered target to be tracked by any agent.

Figures 4-14 and 4-15 response time as a function of communication range and the number of targets. Once again, it is shown that the CBBA algorithm can respond much more quickly to new information because of the efficient manner in which tasks are added to the fleet. It is interesting to note that although the CBBA algorithm has a slightly improved response time as the network range is increased, RDTA becomes much worse. This is because the increase connectivity forces the algorithm to wait for the rest of the fleet's plans and significantly slows the assignment process.

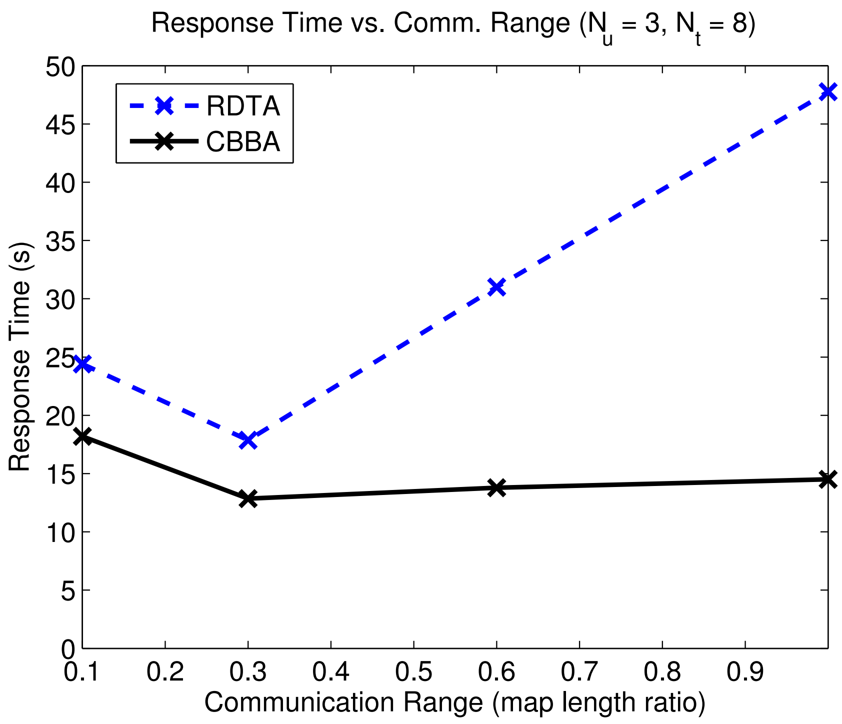


Figure 4-14: Average response time as a function of communication range.

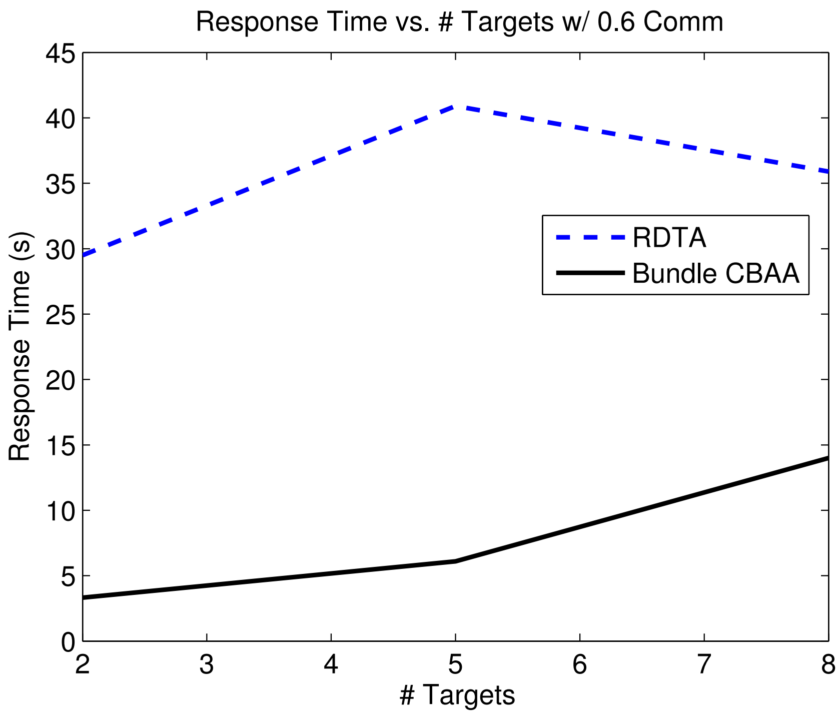


Figure 4-15: Average response time as a function of the number of targets in the environment

Chapter 5

Conclusion

5.1 Summary

This thesis has presented new approaches to multi-agent task assignment that increases both robustness to inconsistencies in the situational awareness, and robustness to time varying network topologies. The consensus-based class of optimization algorithms was developed by incorporating ideas from market-based systems, as well as from the information consensus literature. These types of algorithms were shown to be useful in highly dynamic environments with limited communication such as those found in search and track missions.

Chapter 2 introduced the Consensus-Based Auction Algorithm (CBAA). The algorithm is a greedy single assignment strategy that performs consensus on a winning bids list in order to efficiently resolve conflicts in the fleet. The presented algorithm was shown to guarantee 50% optimality in the worst case, and to give much better expected performance for some illustrative abstractions. Regarding convergence of the algorithm, the CBAA was shown to converge in finite time where the upper bound of the convergence time can be easily quantified. Under some mild assumptions, the algorithm was also shown to converge regardless of inconsistencies in situational awareness and the evolution of the network topology. Numerical results suggested that the solution of the CBAA is very close to optimum. In environments with sparse communication, the algorithm was shown to have better performance than competing strategies such as the ETSP auction algorithm and a greedy based solution.

Chapter 3 extended the CBAA to the multiple assignment problems in two ways. The first one, which is a straightforward extension of CBAA, performs multiple iterations of CBAA routines. It was shown that this iterative extension converges faster than traditional sequential auction strategies. The second algorithm, the Consensus-Based Bundle Algorithm (CBBA), resolves potentially poor performance of the iterative CBAA and provided faster convergence under slight assumptions on the scoring mechanism. In the CBBA, bundles of tasks are created and conflicts are resolved in parallel, which enables various permutations of assignment patterns to improve the performance of the assignment. It was shown that the CBBA guarantees 50% optimality in the worst case, while the actual performance for more practical settings is much better than this worst case. Numerical experiments confirmed fast convergence and good performance of the CBBA, while representing much better scalability than the implicit coordination method that finds the optimal solution. Finally, the CBBA was compared against the Prim Allocation algorithm, a standard sequential auctioning approach, and was shown to have much faster convergence times will providing better assignments.

Chapter 4 presented the implementation of a continuous time CBBA into a sophisticated search and track mission simulator. Instead of performing a periodic assignment on a set list, new tasks are inserted into the fleet as they arise and the assignments re-configure themselves to attain the highest score. Simulations were done to compare this approach with a periodic RDTA assignment system that is currently in. The CBBA was shown to exhibit faster response times to newly discovered targets, was able to handle more track tasks and was able to attain higher mission performances. The CBBA algorithm was also shown to make efficient use of the communication structure in order to improve the performance of the fleet.
In summary, the objectives of this thesis were to develop task assignment algorithms that were:

1. Flexible to varying network structures and communication linkages

2. Robust to dynamic and uncertain environments
3. Guaranteed convergence with an inconsistent SA
4. Provided fast convergence times with optimal or near-optimal solutions

For each algorithm, convergence was shown to be bounded within varying network topologies provided that the union of networks over time were connected. Furthermore, in each case, it was shown that inconsistencies in the SA did not affect the convergence of the algorithm, but did however affect the value of the final assignment. Finally, each algorithm was shown to converge much faster than other competing approaches, while giving near optimal assignments.

5.2 Future Work

The algorithms developed in this thesis have shown promise in missions involving erratic communication links with high levels of uncertainty. Further work should be done to refine these algorithms to ensure that they are scalable and robust to all environment types. Although the scoring systems used in this thesis were often based on time or distance, it is presumable that many other scoring functions exist, perhaps giving even better results. It would also be beneficial to generalize the problem so that the insertion algorithm used to insert a task into a bundle could theoretically be for any type of system. Finally, the sub-modularity requirement on the scoring function should be re-visited in hope of relaxing this constraint on the scoring mechanism.

For search and track missions, this thesis focused on comparing against the RDTA algorithm that was previously implemented into the CSAT architecture. Results indicated that the CBBA was an improvement, but the task assignment literature for these types of missions has yet to be completely explored. Different algorithms exist and should be implemented for comparison. Furthermore, there are many other applications other than search and track missions that require algorithms robust to network variations, thus, the CBBA should be explored as a possible solution for other such scenarios.

Finally, different metrics should be considered as comparisons for the CSAT mission. Analysis should be done to determine the fleet's mission efficiency. This analysis would look for conflicting assignments and provide an estimate of the amount of time agents did overlapping tasks. This would be especially useful in the low communication analysis. Also, for these simulations only a single vehicle type was used, however, much larger and more complex scenarios could be simulated.

Appendix A

CBAA Performance with Specific Scoring Structure

Simulations were performed to verify the CBAA performance in specific mission scenarios. In each case, scores were calculated using

$$c_{ij} = C_j \cdot \lambda^{\frac{d_{ij}}{v_i}} \tag{A.1}$$

where $C_j = 100$ is the maximum task score, $\lambda = 0.95$ is the time discount factor, d_{ij} is the distance from agent i to task j, and $v_i = 40m/s$ is the constant speed of the agents.

In Figure A-1, tasks and agents were uniformly distributed throughout a $2000m \times 2000m$ grid, while Figure A-2 shows the same grid, but with the agents and tasks uniformly distributed along parallel lines. In Figure A-3, the tasks were uniformly distributed in the top right quadrant while the agents were uniformly distributed in the bottom left. Finally, in Figure A-4, agents were uniformly distributed along a circle of radius $R_u = 1000m$, while the tasks were uniformly distributed around a concentric circle of $R_t = \alpha R_u$, where $\alpha = 0.5$. In all cases, the performance was much better ($< 3\%$ optimality gap) than in the randomly generated scoring matrix cases ($< 6\%$ optimality gap) presented in chapter 2.

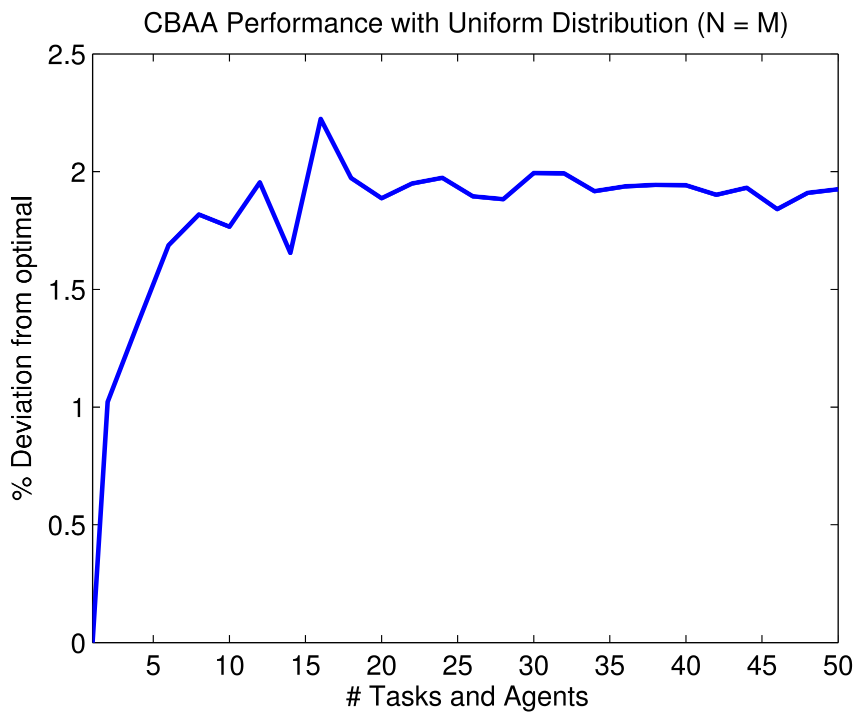


Figure A-1: The deviation of the CBAA algorithm with uniformly generated tasks and agents in a grid

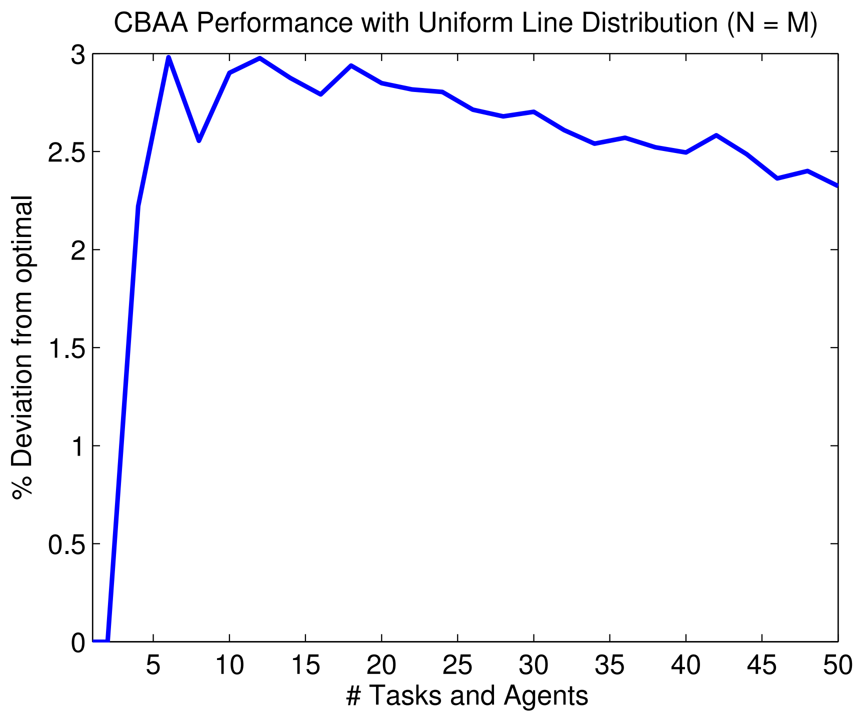


Figure A-2: The deviation of the CBAA algorithm with uniformly generated tasks and agents along parallel lines

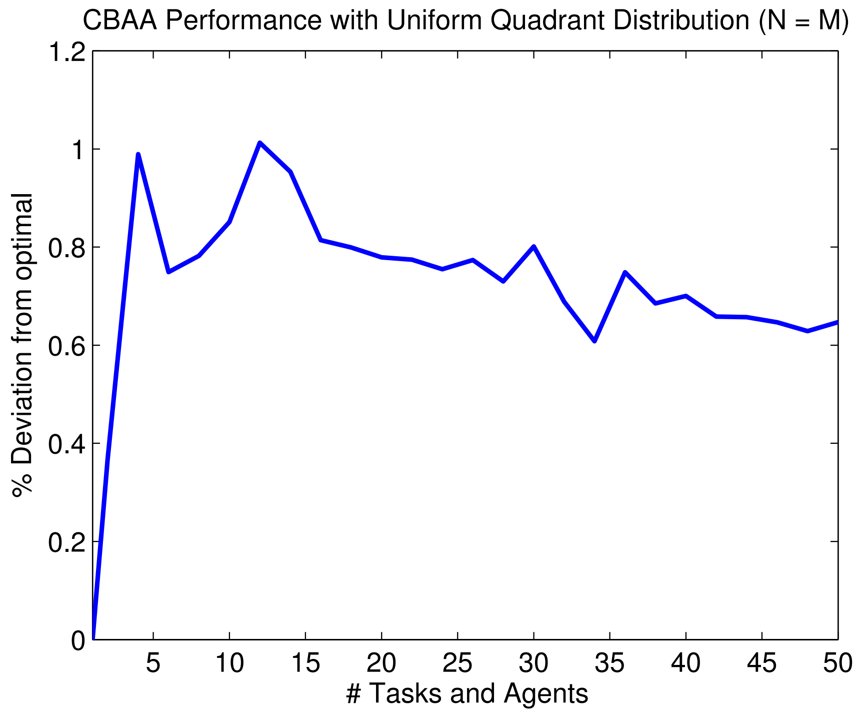


Figure A-3: The deviation of the CBAA algorithm with uniformly generated tasks in one quadrant of a grid, and uniformly distributed agents in another

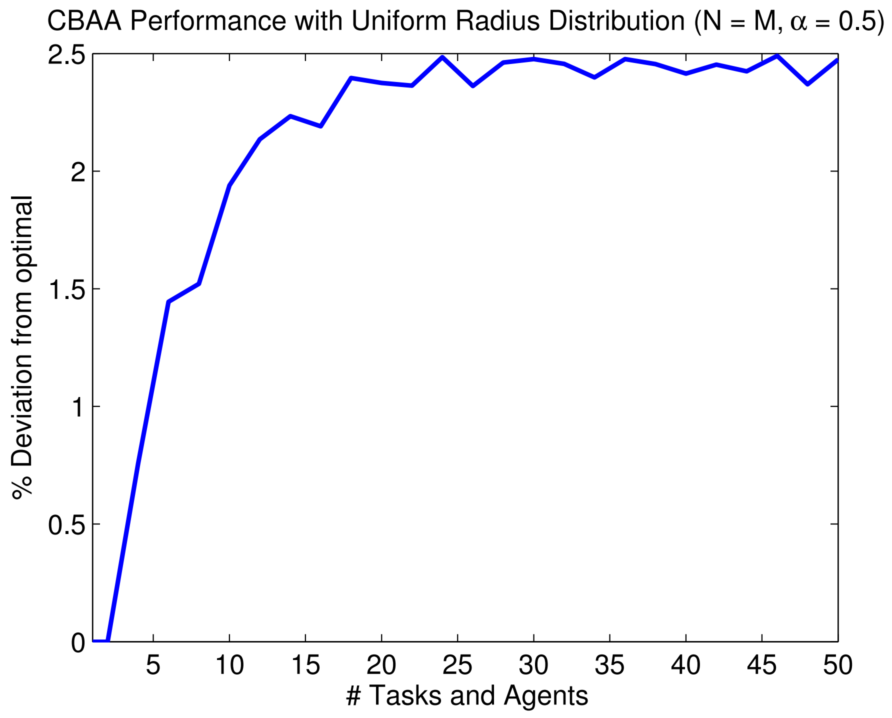


Figure A-4: The deviation of the CBAA algorithm with uniformly generated tasks and agents around concentric circles with $\alpha = 0.5$

Appendix B

Algorithm Details

B.1 Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP) [94] is a problem in which, given a list of cities and a cost of traveling in between each of them, a salesman must find the cheapest way of visiting each one while returning to the start position. The solution to this problem is often called the TSP tour of the cities. The problem is known to be $\mathcal{NP}$-hard, and can be formulated using integer programming methods [106] as follows

$$\begin{aligned} \max \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} x_{ij} & \\ \text{subject to} \quad & \sum_{i=1}^{N} x_{ij} = 2 \\ & \sum_{\delta(S)} x_{ij} \geq 2, \quad S \subset \mathcal{N}, S \neq \emptyset \\ & x_{ij} \in \{0, 1\} \end{aligned} \tag{B.1}$$

where N is the total number of cities to visit, c_{ij} is the cost of traveling from city i to city j, $x_{ij} = 1$ if the path from city i to city j is active, and S is any non-empty subset of the set of nodes $\mathcal{N}$. The first constraint ensures that each node participates in exactly two edges along the tour, while the second constraint makes sure that the entire path is connected and does not consist of multiple sub-tours.

B.2 Cheapest Insertion Algorithm

The cheapest insertion algorithm [93, 95] is an approximate algorithm for solving the Traveling Salesman Problem (TSP). The algorithm starts with a sub-graph consisting of the starting node, and iteratively selects the lowest cost city to insert into the sub-tour. This procedure is continued until every city has been added to the tour

Algorithm 3 Cheapest Insertion Algorithm:

```
1: Start with initial sub-tour $S$ of city $s$ only
2: while $|S| < N$ do
3:     procedure INSERT INTO SUB-TOUR
4:         $\forall\ k \notin S$
5:         Find $\text{argmin}_k\{\ \min_{i,j}\{c_{ik} + c_{kj} - c_{ij}\}\ \}$
6:         Insert city $k$ in between cities $i$ and $j$ in $S$
7:     end procedure
8: end while
```

The Cheapest Insertion Algorithm can be shown to produce tours no longer than twice the length of the optimal tour, and computations on the order of $n^2 lg(n)$

Bibliography

[1] M. Valenti, *Approximate dynamic programming with applications in multi-agent systems*. PhD thesis, Massachusetts Institute of Technology, 2007.

[2] J. Paduano, O. Toupet, and J. How, "CSAT STTR Phase II Proposal." Unpublished, 2006.

[3] A. F. S. A. Board, "UAV Technologies and Combat Operations," Tech. Rep. SAF/PA 96-1204, 1996.

[4] "Army Unmanned Aerial Vehicle (UAV) Systems: A Cost-Effective Combat Multiplier."

[5] O. of th Secretary of Defense, "Unmanned aerial systems roadmap," tech. rep., 2002.

[6] B. Shah and H. Choset, "Survey on urban search and rescue robotics," *Journal of the Robotics Society of Japan*, vol. 22(5), pp. 582–586, 2004.

[7] A. Davids, "Urban search and rescue robots: From tragedy to technology," *IEEE Intelligent Systems*, vol. 17(2), pp. 81–83, 2002.

[8] B. Dragt, F. Camisani-Calzolari, and I. Craig, "An overview of the automation of load-haul-dump vehicles in an underground mining environment," tech. rep., University of Pretoria, 2005.

[9] J. Manley, "Autonomous underwater vehicles for ocean exploration," in *Proceedings of the IEEE OCEANS*, 2003.

[10] A. L. Meyrowitz, D. R. Blidberg, and R. Michelson, "Autonomous vehicles," *Proceeding of the IEEE*, vol. 84(8), pp. 1147–1164, 1996.

[11] Hydro-International, "AUV Product Survey." www, September 2006.

[12] "Darpa urban challenge homepage." Available at http://www.darpa.mil/GRANDCHALLENGE/.

[13] M. Valenti, B. Bethke, J. How, D. P. de Farias, and J. Vian, "Embedding Health Management into Mission Tasking for UAV Teams," in *American Control Conference*, 2007.

[14] 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.

[15] C. Schumacher, P. Chandler, and S. Rasmussen, "Task allocation for wide area search munitions," in *Proceedings of the American Control Conference*, 2002.

[16] 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.

[17] 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.

[18] L. Xu and U. Ozguner, "Battle management for unmanned aerial vehicles," in *Proceedings of the IEEE Conference on Decision and Control*, 2003.

[19] D. Turra, L. Pollini, and M. Innocenti, "Fast unmanned vehicles task allocation with moving targets," in *Proceedings of the IEEE Conference on Decision and Control*, Dec 2004.

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

[21] K. Nygard, P. Chandler, and M. Pachter, "Dynamic network flow optimization models for air vehicle resource allocation," in *Proceedings of the American Control Conference*, 2001.

[22] 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.

[23] D. Castanon and C. Wu, "Distributed algorithms for dynamic reassignment," in *Proceedings of the IEEE Conference of Decision and Control*, 2003.

[24] J. Curtis and R. Murphey, "Simultaneous area search and task assignment for a team of cooperative agents," in *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[25] T. Shima, S. J. Rasmussen, and P. Chandler, "UAV Team Decision and Control using Efficient Collaborative Estimation," in *Proceedings of the American Control Conference*, 2005.

[26] P. Chandler, "Decentralized control for an autonomous team," in *Proceedings of AIAA Unmanned Unlimited Systems, Technologies, and Operations*, 2003.

[27] W. Ren, R. Beard, and D. Kingston, "Multi-agent kalman consensus with relative uncertainty," in *Proceedings of American Control Conference*, 2005.

[28] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49(9), pp. 1520–1533, 2004.

[29] M. Alighanbari and J. P. How, "An unbiased kalman consensus algorithm," in *Proceedings of the American Control Conference*, 2006.

[30] C. C. Moallemi and B. V. Roy, "Consensus propagation," *IEEE Transactions on Information Theory*, vol. 52(11), pp. 4753–4766, 2006.

[31] 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.

[32] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, 2003.

[33] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Transactions on Automatic Control*, vol. 49(9), pp. 1465–1476, 2004.

[34] A. Tahbaz-Salehi and A. Jadbabaie, "On consensus over random networks," in *44th Annual Allerton Conference*, 2006.

[35] R. Beard and V. Stepanyan, "Synchronization of information in distributed multiple vehicle coordinated control," in *Proceedings of the IEEE Conference on Decision and Control*, 2003.

[36] Y. Hatano and M. Mesbahi, "Agreement over random networks," in *43rd IEEE Conference on Decision and Control*, 2004.

[37] C. W. Wu, "Synchronization and convergence of linear dynamics in random directed networks," *IEEE Transactions on Automatic Control*, vol. 51(7), p. 12071210, 2006.

[38] W. Kang and A. Sparks, "Task Assignment in the Cooperative Control of Multiple UAV's," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2003.

[39] D. Dionne and C. A. Rabbath, "Multi-UAV Decentralized Task Allocation with Intermittent Communications: the DTC algorithm," in *Proceedings of the American Control Conference*, 2007.

[40] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms," in *Proceedings of Robotics: Science and Systems Conference*, 2005.

[41] M. Flint, T. Khovanova, and M. Curry, "Decentralized control using global optimization," in *Proceedings of the AIAA*, 2007.

[42] P. Chandler and M. Pachter, "Hierarchical control for autonomous teams," in *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2001.

[43] M. Alighanbari and J. P. How, "Decentralized task assignment for unmanned aerial vehicles," in *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, 2005.

[44] M. Alighanbari and J. How, *Robust and Decentralized Task Assignment Algorithms for UAVs.* PhD thesis, MIT, 2007.

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

[46] 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.

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

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

[49] D. P. Bertsekas and D. A. Castanon, "Parallel synchronous and asynchronous implementations of the auction algorithm," *Parallel Computing*, vol. 17, pp. 707–732, 1991.

[50] 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.

[51] A. M. Kwasnica, J. O. Ledyard, D. Porter, and C. DeMartini, “A new and improved design for multiobject iterative auctions,” *Management Science*, vol. 51(3), p. 419434, 2005.

[52] P. Milgrom, “Putting auction theory to work: The simultaneous ascending auction,” *The Journal of Political Economy*, vol. 108(2), pp. 245–272, 2000.

[53] M. G. Lagoudakis, M. Berhaultt, S. Koenigt, P. Keskinocak, and A. J. Kleywegt, “Simple auctions with performance guarantees for multi-robot task allocation,” in *Proceedings of the IEEE/RSI International Conference on lntelllgent Robots and Systems*, 2004.

[54] S. Sariel and T. Balch, “Real time auction based allocation of tasks for multi-robot exploration problem in dynamic environments,” in *Proceedings of the AIAA Workshop on "Integrating Planning Into Scheduling"*, 2005.

[55] A. Ahmed, A. Patel, T. Brown, M. Ham, M. Jang, and G. Agha, “Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism,” in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2005.

[56] M. L. Atkinson, “Results Analysis of Using Free Market Auctions to Distribute Control of UAVs,” in *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, 2004.

[57] T. Lemaire, R. Alami, and S. Lacroix, “A Distributed Task Allocation Scheme in Multi-UAV Context,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004.

[58] M. Hoeing, P. Dasgupta, P. Petrov, and S. OHara, “Auction-based Multi-Robot Task Allocation in COMSTAR,” in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007.

[59] P. B. Sujit and R. Beard, “Distributed Sequential Auctions for Multiple UAV Task Allocation,” in *Proceedings of the American Control Conference*, 2007.

[60] S. L. Smith and F. Bullo, "Target assignment for robotic networks: Asymptotic performance under limited communication," in *Proceedings of the American Control Conference*, 2007.

[61] X. Zheng, S. Koenig, and C. Tovey, "Improving sequential single-item auctions," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[62] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain, "Auction-based multi-robot routing," in *Proceedings of Robotics: Science and Systems*, 2005.

[63] D. C. Parkes and L. H. Ungar, "Iterative combinatorial auctions:theory and practice," in *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[64] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt, "Robot exploration with combinatorial auctions," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[65] A. Andersson, M. Tenhunen, and F. Ygge, "Integer programming for combinatorial auction winner determination," in *Proceedings. Fourth International Conference on MultiAgent Systems*, 2000.

[66] S. de Vries and R. Vohra, "Combinatorial auctions: A survey," *INFORMS Journal of Computing*, vol. 15(3), pp. 284–309, 2003.

[67] M. H. Rothkopf, A. Pekec, and R. M. Harstad, "Computationally manageable combinatorial auctions," tech. rep., Rutgers University, 1998.

[68] T. Sandholm, "Algorithm for optimal winner determination in combinatorial auctions," *Artificial Intelligence*, vol. 135(1-2), pp. 1–54, 2002.

[69] M. Nandy and A. Mahanti, "An improved search technique for optimal winner determination in combinatorial auctions," in *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004.

[70] M. Mito and S. Fujita, "On heuristics for solving winner determination problem in combinatorial auctions," *Journal of Heuristics*, vol. 2004, pp. 507 – 523, 10(5).

[71] M. Alighanbari, L. Bertuccelli, and J. How, "Filter-embedded uav task assignment algorithms for dynamic environments," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004.

[72] J. Bellingham, M. Tillerson, M. Alighanbari, and J. How, "Cooperative Path Planning for Multiple UAVs in Dynamic Uncertain Environments," in *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[73] A. Ryan, J. Tisdale, M. Godwin, D. Coatta, D. Nguyen, S. Spry, R. Sengupta, and J. K. Hedrick, "Decentralized control of unmanned aerial vehicle collaborative sensing missions," in *Proceedings of the American Control Conference*, 2007.

[74] J. Rubio, J. Vagners, and R.Rysdykz, "Adaptive Path Planning for Autonomous UAV Oceanic Search Missions," in *AIAA 1st Intelligent Systems Technical Conference*, 2004.

[75] M. Flint, M. Polycarpou, and E. Fernandez-Gaucherand, "Cooperative control of multiple autonomous UAV's search for targets," in *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[76] M. Polycarpou, Y. Yang, and K. Passino., "A cooperative search framework for distributed agents," in *Proceedings of the IEEE International Symposium on Intelligent Control*, 2001.

[77] P. Vincent and I. Rubin, "A Framework and Analysis for Cooperative Search Using UAV Swarms," in *ACM Symposium on Applied Computing*, 2004.

[78] R. Hashemi, L. Jin, G. Anderson, E. Wilson, and M. Clark, “A comparison of search patterns for cooperative robots operating inremote environment,” in *Proceedings. International Conference on Information Technology: Coding and Computing*, 2001.

[79] H. Wollan, “Incorporating heuristically generated search patterns in search and rescue,” Master’s thesis, University of Edinburgh, 2004.

[80] B. Moret, M. Collins, J. Saia, and L. Yu, “The ice rink problem,” in *Proceedings of the 1st Workshop on Algorithm Engineering*, 1997.

[81] V. Ablavsky and M. Snorrason, “Optimal search for a moving target: A geometric approach,” in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2000.

[82] L. Bertuccelli and J. How, “Search for dynamic targets with uncertain probibility maps,” in *Proceedings of the American Control Conference*, 2006.

[83] S. Hansen, T. McLain, and M. Goodrich, “Probabilistic searching using a small unmanned aerial vehicle,” in *AIAA InfotechAerospace Conference and Exhibit*, 2007.

[84] J. Hespanha and H. Kizilocak, “Efficient computation of dynamic probabilistic maps,” in *Proceedings of the 10th Mediterranean Conference on Control and Automation*, 2002.

[85] A. Richards, J. Bellingham, M. Tillerson, and J. How, “Coordination and Control of Multiple UAVs,” in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2002.

[86] R. W. Beard and T. W. McLain, “Multiple UAV cooperative search under collision avoidance and limited range communication constraints,” in *Proceedings of the IEEE Conference on Decision and Control*, 2003.

[87] A. Richards and J. How, "Mixed-integer programming for control," in *American Control Conference*, 2005.

[88] C. H. Lee and K. G. Shin, "Optimal task assignment in homogeneous networks," *IEEE Transactions onN Parallel Computing and Distributed Systems*, vol. 8(2), pp. 119–129, 1997.

[89] "Personal communication with Han-Lim Choi, PhD Candidate, Aero/Astro, MIT.," 2008.

[90] H. David and H. Nagaraja, *Order Statistics.* Wiley, 2003.

[91] D. B. Wilson, "Generating random spanning trees more quickly than the cover time," in *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing*, 1996.

[92] S. Fujishige, "Submodular functions and optimization," in *Annals of Discrete Mathematics*, Elsevier Science, 1991.

[93] B. Golden, L. Bodin, T. Doyle, and W. Stewart, "Approximate traveling salesman algorithms," *Operations Research*, vol. 28(3), pp. 694–711, 1980.

[94] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Travelling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

[95] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6(3), pp. 563–581, 1977.

[96] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* The MIT Press, 1990.

[97] C. Tovey, M. G. Lagoudakis, S. Jain, and S. Koenig, "The generation of bidding rules for auction-based robot coordination," in *Proceedings of the 3rd International Multi-Robot Systems Workshop*, 2005.

[98] H. Choset, "Coverage for robotics - a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113 – 126, 2001.

[99] S. Hert, S. Tiwari, and V. Lumelsky, "A Terrain Covering Algorithm for an AUV," *Autonomous Robots*, vol. 3(2), pp. 91 – 119, 1996.

[100] V. Lumelsky, S. Mukhopadhyay, and K. Sun, "Dynamic path planning in sensor-based terrain acquisition," *IEEE Transactions on Robotics and Automation*, vol. 6(4), pp. 462–472, 1990.

[101] S. Park, *Avionics and Control System Development for Mid-Air Rendezvous of Two Unmanned Aerial Vehicles.* PhD thesis, Massachusetts Insitute of Technology, 2004.

[102] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[103] R. K. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *Journal of the ACM*, vol. 37(2), pp. 213 – 223, 1990.

[104] G. Laporte and F. Semet, *The Vehicle Routing Problem*, ch. Classical Heuristics for the Capacitated VRP, pp. 109–128. 2002.

[105] *GNU Linear Programming Kit Reference Manual.*

[106] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization.* Athena Scientific, 1997.