Decentralized Task Allocation for Dynamic, Time-Sensitive Tasks

by

Noam Buckman

B.S. Mechanical Engineering and Mathematics,
Massachusetts Institute of Technology (2016)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Masters of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

Author . . . .
Department of Mechanical Engineering
August 19, 2018

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

Read by . . . .
John Leonard
Samuel C. Collins Professor of Mechanical and Ocean Engineering
Thesis Reader

Accepted by . . . .
Rohan Abeyaratne
Quentin Berg Professor of Mechanics
Chairman, Committee on Graduate Students

Decentralized Task Allocation for Dynamic, Time-Sensitive Tasks

by

Noam Buckman

Submitted to the Department of Mechanical Engineering
on August 19, 2018, in partial fulfillment of the
requirements for the degree of
Masters of Science in Mechanical Engineering

Abstract

In time-sensitive and dynamic missions, autonomous vehicles must respond quickly to new information and objectives. In the case of dynamic task allocation, a team of agents are presented with a new, unknown task that must be allocated with their original allocations. This is exacerbated further in decentralized settings where agents are limited to utilizing local information during the allocation process.

This thesis presents a fully decentralized, dynamic task allocation algorithm that extends the Consensus-Based Bundle Algorithm (CBBA) to allow for allocating new tasks. Whereas static CBBA requires a full resetting of previous allocations, CBBA with Partial Replanning (CBBA-PR) enables the agents to only partially reset their allocations to efficiently and quickly allocate a new task. By varying the number of existing tasks that are reset during replan, the team can trade-off convergence speed with amount of coordination. By specifically choosing the lowest bid tasks for resetting, CBBA-PR is shown to converge linearly with the number of tasks reset and the network diameter of the team.

In addition, limited replanning methods are presented for scenarios without sufficient replanning time. These include a single reset bidding procedure for agents at capacity, a no-replanning heuristic that can identify scenarios that does not require replanning, and a subteam formation algorithm for reducing the network diameter. Finally, this thesis describes hardware and simulation experiments used to explore the effects of ad-hoc, decentralized communication on consensus algorithms and to validate the performance of CBBA-PR.

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

Acknowledgments

I would like to thank my advisor Prof. Jonathan How for his advising over the past two years, especially for introducing me to many of the research problems in this thesis. Thank you to Prof. Han-Lim Choi for the valuable late night video calls discussing my research. Thank you Dr. Golnaz Habibi for meeting regularly with me and guiding my research. Thank you to Prof. John Leonard for serving as the department reader for this thesis.

I would like to thank all the members of the Aerospace Controls Lab for their thoughtful discussions, general camaraderie, and overall enhancing of my experience at MIT. I would like to specifically thank those that took the time to provide feedback about my research throughout my time at MIT: Michael Everett, Matt Giamou, Brett Lopez, Dong-Ki Kim, Kasra Khosoussi, and Chris Fourie.

Thank you additionally to my fellow LIDS-ACL inhabitants who worked next to me both in Stata and Building 31. Shayegan Omidshafiei who mentored me as a new student and constantly answered all of my questions about research and graduate school. To Erin Evans (and Kaia) for constantly providing humor and energy to the lab. And to Nikita Jaipuria, my desk-mate, who is a constant source of friendship and advice in all aspects of graduate school and life.

Thank you Lockheed Martin for funding my first year and for providing constant feedback on my research and to the Department of Defense for funding my research through the National Defense Science & Engineering Graduate Fellowship.

Thank you to my all of the people outside MIT who have supported me. To my Camberville friends that kept me busy outside of lab. To my family for your constant support and inspiration. And to Nomi Hachen, for being at my side along the way, whether proofreading my research presentations or providing encouragement, and generally adding a lot of joy and happiness to my life.

Finally, this thesis is dedicated in memory of my grandfather David Lieberman, who was a big supporter of my education at MIT and continues to be a role model for hard work and integrity.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

1 Introduction 11

1.1 Problem Statement . . . 15

1.2 Research Challenges . . . 16

1.3 Contributions . . . 20

2 Preliminaries 23

2.1 Centralized Dynamic Vehicle Routing Methods . . . 23

2.2 Consensus Theory . . . 25

2.3 Decentralized Subgradient Methods . . . 26

2.4 Decentralized Simplex Methods . . . 28

2.5 Auction Algorithms . . . 29

2.6 Consensus-Based Bundle Algorithm (CBBA) . . . 31

2.7 Summary . . . 37

3 Decentralized Dynamic Task Allocation by Partial Replanning 39

3.1 Replanning for a New Task . . . 39

3.2 Effects of Replanning in Auction Algorithms . . . 41

3.3 Bundle Resetting in CBBA . . . 45

3.4 CBBA with Partial Replanning (CBBA-PR) . . . 48

3.5 Convergence . . . 51

3.6 Results . . . 55

3.7 Summary . . . 61

4 Dynamic Allocation with Limited Resetting 63

4.1 Bidding at Full Capacity 64

4.2 Full Reset Performance Heuristic 67

4.3 Task Discovery and Subteam Formation 72

4.4 Summary 75

5 Effects of Broadcasting Over Ad-hoc Networks on Team Consensus 77

5.1 Contention and Effects on Delay 78

5.2 802.11 Distributed Coordination Function 79

5.3 Decentralized Broadcast Experiments 81

5.4 Summary 86

6 Hardware and Simulation Experiments 89

6.1 Dynamic Task Simulator 90

6.2 Raspberry Pi Experiments 95

6.3 Summary 100

7 Conclusion and Future Work 101

7.1 Conclusion 101

7.2 Future Work 102

List of Figures

1-1 Defense application of decentralized sensors 11
1-2 Commercial application of distributed task allocation 12
1-3 Dynamic Task Allocation 20

2-1 CBBA Bidding 34

3-1 Replan with No Bundle Reset 41
3-2 CBBA with Partial Reset 41
3-3 Bias in Greedy Bidding 43
3-4 CBBA full reset strategy for allocation a new task 46
3-5 CBBA-PR 48
3-6 Simulation of Dynamic Task Allocation 56
3-7 Example Timeline of Convergence 57
3-8 Convergence of Reset Algorithms Before New Task 58
3-9 Convergence of Reset Algorithms After New Task 58
3-10 Partial replan performance 60

4-1 Bidding at Capacity 64
4-2 Single Reset at Capacity 65
4-3 T-Only Heuristic 69
4-4 Decentralize procedure for estimating the performance of a full replan 69
4-5 Replanning based on performance heuristic 70
4-6 Solution quality of heuristic 71
4-7 Discovering Task 73

4-8 Replan Radius 74
4-9 Fast CBBA Convergence 75

5-1 Congestion 79
5-2 Hidden Node 80
5-3 Exposed Node 80
5-4 Raspberry Pi Placement in 6th Floor Building 32 at MIT 81
5-5 Network PSR 83
5-6 Comparing Ad-Hoc Degradation 84
5-7 Rebroadcasting Delay Results 85
5-8 Rebroadcasting Reliability Results 86

6-1 Simulations 91
6-2 WiFi Status for Ideal Communication 92
6-3 WiFi Status for Real Communication 92
6-4 Raspberry Pis placed in Building 32 for consensus experiments 96
6-5 Raspberry Pis in Building 31 for CBBA experiment tests 96
6-6 Replan Time Experiments 99

Chapter 1

Introduction

The availability of affordable, high-powered computing as well as the mature development of control systems has led to a proliferation of autonomous systems in commercial, defense, and recreational worlds. Large demonstrations of flying quadcopter teams can be spotted in Superbowl Halftime shows and self-driving cars can be seen driving down the streets of Cambridge, MA. The appeal of these autonomous systems is two-fold. First, autonomous systems remove the need for humans to be involved in accomplishing various tasks. From replacing human farmers with autonomous tractors [5] to reducing vehicular deaths by removing human drivers [6] to increasing warfighter safety by removing pilots from the cockpit [7] , autonomous systems has the potential to reduce cost and increase safety in various applications.

The second is leveraging the computational power of autonomous systems to solve

(a) US Army Urban Operations (2025) [1]

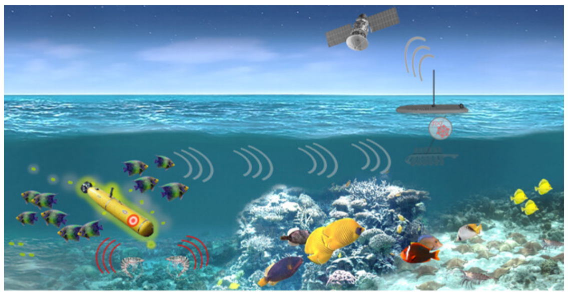
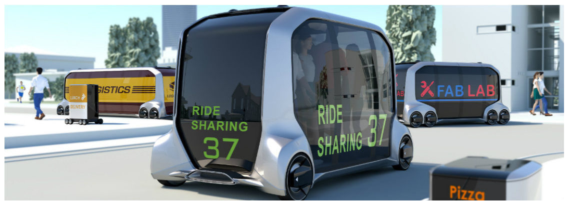

(b) Persistent Aquatic Living Sensors [2]

Figure 1-1: Defense applications of decentralized task allocation include networks of sensors that work together to fuse various sensor modalities and respond quickly to threats



(a) Toyota Ridesharing Concept [?]

(b) Amazon Robotics [4]

Figure 1-2: Commercial applications of distributed task allocation such as Amazon and Toyota use large teams of networked robots to distribute tasks across multiple agents

increasingly complex problems in decreasing amount of time, surpassing the abilities of human predecessors. With improved deep learning architectures, computers have begun to surpass even human abilities to identify objects [8]. The famous Deep Blue system defeated the world champion in chess and recently AlphaGo defeated the world champion in Go [9]. In these cases, computers are able to outperform humans due to their increased speed of computation and vast amount of memory.

This thesis focuses on the second advantage of autonomous system, namely the ability of computers to quickly solve increasingly complex problems in time-sensitive scenarios with increased performance compared to humans. One way to surpass human performance is to increase the number of agents. Instead of four humans searching for a missing hiker, search and rescue teams can send 100 drones to survey the area. In disaster environments, teams of ground vehicles and aerial vehicles can work together to find survivors in extreme circumstances. However, with an increase in agents leads to an increase in the complexity of the problem. How do the agents coordinate together? How do robots work together to out perform any single robot? The ability to *coordinate* is crucial for improved performance of robots. And while humans have refined language and communication over the past thousands of years, teamwork is still an evolving field. How do humans negotiate on teams? The challenges of coordination are exacerbated as engineers must code these interactions in a large scale system and require systems to solve increasingly complex problems.

Applications of distributed autonomous systems is quickly emerging in present industries and near-future application. In autonomous mobility-on-demand technologies, companies such as Ford and Uber are creating large systems of autonomous cars that are tasked with routing to numerous riders and safely taking them to their end destination. From the local perspective, vehicles must fuse sensor measurements to detect oncoming traffic and pedestrians, provide control signals to the motors and wheel to safely steer the vehicle, and interface with the human rider. From a more global perspective, the cars must coordinate such that each customer is serviced within an acceptable Quality of Service (QoS), usually requiring that no two vehicles are visiting the same customer.

In defense applications, robots must be able to endure the most extreme environments with limited and unreliable communication and complex mission settings. Up until recently, UAVs have been largely used for passively collecting information, hovering at high altitude with various sensors. However, future systems envision fully autonomous planes replacing the role of classically trained pilots[7]. They will operate in hostile environments, making critical decisions based on the new information. Simply put, autonomous vehicles will need to make *decisions*.

For example, where as previously, autonomous aerial vehicles were tasks with surveying opposing military forces or gathering intelligence on weapons depots, now quad-copters are tasked with intercepting incoming adversaries or actively pursuing unknown targets. Agents must now first sense their environments, reason about their situational awareness, and make time-critical decisions. If a team of robots must intercept an incoming target, agents may collectively sense the target, fusing the measurements from each agent to reach a consistent map of the environment and the placement of the incoming target. The team may also delegate one or more agents to pursue and capture the incoming target. The team will need to decide which agent is the ideal one to accomplish task and ensure that the team is in agreement. Finally, the team may wish to assess the status of the incoming target and resume any previous operation of the team.

The ability for the team to allocate certain tasks to individual agents within a team is known as the task allocation problem. In the task allocation problem, a set of tasks such as visiting a location, picking up a customer, or taking a measurement must be divided among agents. Generally, the agents will have different abilities or preferences towards accomplishing the task. In the case of a heterogeneous team of robots, each robot may have different innate abilities, in which case, the team should try to match agents with the tasks they can best perform. In other cases, the robots themselves may be homogeneous, however they can better execute the mission if they do an effective job in dividing up the tasks among the agents. Finally, there are situations where there is a mixture of robot heterogeneity and incentive to effectively divide tasks, in which case effective coordination greatly increases the performance of the team.

If individual tasks are assigned to each agent then the agents are solving the *task assignment* problem. Because only one task is assigned to each agent, agents do not need to consider the exponential number of combinations of tasks that could be assigned to each agent. This not only greatly reduces the complexity of the problem for a generic problem, it also enables the team to assign a new task independently of the existing tasks in the system. In contrast, in a task allocation setting, where the new task may negatively impact the original allocations, simply assigning the new task to an agent is not a viable strategy.

The focus of this research will be on allocating tasks that arrive online during the allocation of tasks. In static versions of the task allocation problem, the team is provided with a set of tasks *a priori* that must be allocated as quickly as possible, providing high quality allocations. In the dynamic task allocation setting, a *new* task may arrive during or even after the entire task allocation process. The primary question in this thesis is how to create the algorithms that will both effectively allocate the new task among the team while maintaining a quick response time. As such, this thesis is primarily interested in tasks that are time-sensitive where the actual time it takes for the team to allocate the tasks directly impact the team's performance. For example, if the team was allowed infinite time to allocate a new task, the team could simply stop and re-solve the original problem. Rather in this work, the team must sacrifice some solution quality to ensure that the team returns with a new allocation quickly. For example, if a high-speed incoming adversarial agent must be captured, the team can not wait a long time re-calculating a new solution. If it waits too long, the current situational awareness may no longer be accurate, in the worst-case, the adversarial is no longer within reach. Likewise, if a team of autonomous cars must pause and replan their entire routes for every new customer that would like a ride, even though the routes are highly efficient, the customers will not accept the long wait times for replanning.

1.1 Problem Statement

1.1.1 Static Task Allocation Problem

The goal of the task allocation problem is to allocate a set of n_t tasks to n_r agents where each tasks must be assigned to only one agent. In the decentralized task allocation problem, each agent must solve this optimization and the team should arrive at a conflict-free assignments, meaning that the team must agree on a single solution. The agents can assign to themselves up to L_t tasks, constrained by either a physical limitation or a planning horizon for the agent. The decentralized task assignment problem can then be formed as an integer program:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{n_r}\left(\sum_{j=1}^{n_t} c_{ij}(\boldsymbol{x}_i, \boldsymbol{p}_i) x_{ij}\right) \\
\text{subject to:} \quad & \sum_{j=1}^{n_t} x_{ij} \leq L_t \quad \forall i \in \mathcal{I} \\
& \sum_{i=1}^{n_r} x_{ij} \leq 1 \quad \forall j \in \mathcal{J} \\
& \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} x_{ij} = \min\{n_r L_t, n_t\} \\
& x_{ij} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{I} \times \mathcal{J}
\end{aligned}$$

where $\mathcal{I}$ is the set of all robots, $\mathcal{J}$ is the set of all tasks, $x_{ij} \in \{0,1\}$ is a decision variable for task j being assigned to agent i, $\boldsymbol{p}_i$ is the allocation of tasks for each robot i, and c_{ij} is the reward for servicing task i given allocation $\boldsymbol{p}_i$.

1.1.2 Description of Dynamic Task Allocation

This work focuses on a the dynamic task allocation problem, where a team of agents must allocated a new task T^* during or after having allocated n_t tasks. In this case, the agents begin with an initial assignment of tasks (obtained from solving the static task allocation problem) and must now allocate the new task T^* among the agents to obtain new allocations $\boldsymbol{p}_i'$. A new task set is defined $\mathcal{J}' = \mathcal{J} \oplus T^*$, and a new task allocation problem is formulated as

$$
\begin{aligned}
\max \qquad & \sum_{i=1}^{n_r}\left(\sum_{j=1}^{n_t+1} c_{ij}'(\boldsymbol{x}_i', \boldsymbol{p}_i')x_{ij}'\right) \\
\text{subject to:} \quad & \sum_{j=1}^{n_t+1} x_{ij}' \le L_t \quad \forall i \in \mathcal{I} \\
& \sum_{i=1}^{n_r} x_{ij}' \le 1 \quad \forall j \in \mathcal{J}' \\
& \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} x_{ij}' = \min\{n_r L_t, n_t + 1\} \\
& x_{ij}' \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{I} \times \mathcal{J}'
\end{aligned}
$$

1.2 Research Challenges

There are three main characteristics to the decentralized dynamic task allocation problem that are critical for any algorithm to effectively be used in real-world systems: tractable computational complexity, decentralized implementation, and the ability to handle dynamic tasks and environments.

1.2.1 Computational Complexity

The task allocation problem can be viewed as a general version of the Traveling Salesperson Problem (TSP) or similarly a Vehicle Routing Problem (VRP), both of which have been shown to be NP-hard when trying to find an optimal solution [10, 11]. Likewise, one can see that the problem will be NP-hard by the fact that the optimization is an integer program which is another example of an NP-hard problem. The complexity of this problem means that the only way to obtain the optimal allocation of tasks is by iterating all permutations of task allocations, which is exponential in the number of tasks and agents in the problem. Unlike NP-complete problems, it is not even possible to *validate* a feasible solution as optimal if the team was given one. Thus, any tractable algorithm , i.e., one that can be solved in polynomial time, will not be able to guarantee an optimal solution. Instead, a tractable solution should arrive at a feasible solution, meeting the constraints of the problem such as conflict-free allocations, and should make an attempt at arriving at solutions that are close to optimal. One approach is to derive approximation algorithms [12] which can guarantee solutions that are within a constant factor of the optimal solution. Another approach is to use heuristics to direct the search towards solutions that have a high likelihood of success, speeding up the algorithm to not require an exhaustive search of solutions.

It is important, however, to distinguish between task allocation which considers the groups of tasks assigned to each agents with task assignment, where individual tasks are assigned to an agent without effects subsequent tasks assigned to the agent. If this is the case, the assignment problem can be represented as a bipartite graph where the edges are the robot costs for servicing a given task. In this case, the problem can be solved in polynomial time by utilizing known algorithms such as the Hungarian method which has been shown to be strongly polynomial with a runtime of $O(n_t^3)$ [13]. This distinction is important as many state-of-the-art decentralized or dynamic algorithms solve a task assignment problem, or some variation of simplifying the optimization problem. In this thesis, the team value function will depend on the entire task allocation resulting in an NP-hard optimization problem. As such, an algorithm is that sought that will provide an approximate solution to the task allocation problem that is both decentralized and dynamic.

1.2.2 Decentralization

The use of unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) in large teams has become increasingly desired and viable as robot hardware has decreased in size and cost. Likewise, there is increasing interest in solving large, more complex missions that require multi-agent teams to accomplish a varied number of tasks. Decentralized algorithms have allowed planners to scale with larger team sizes, amortizing computation and communication across the robot teams. In addition, decentralized algorithms, which only rely only peer-to-peer communication, can be used in environments without a communication infrastructure or in environment with constrained centralized communication. For example, a team of UAVs operating in a foreign terrain, may not have access to classic communication infrastructure that one may be accustomed to in local settings, especially for missions utilizing airspace or underwater environments. Likewise, in an adversarial setting, where opponents may look to target a central planner, decentralized algorithms provide robustness to single-point failures caused by a central planner or communication infrastructure.

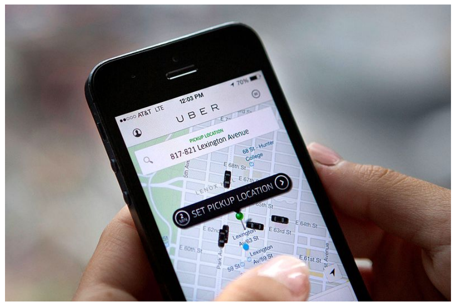
In contrast to distributed algorithms, which allows some centralized computation in the form of dividing tasks or computation, decentralized algorithms rely on only peer to peer communication, where all other computation and decision is made locally. For example, in a distributed setting, the agents may be requested to perform some local computation and then return to a central planner with their results for final coordination. In a decentralized algorithm, no central planner is allowed. As such, many distributed algorithm stem from the computer architecture community which may have a central kernel or process manager that can coordinate multiple threads of computation using some shared memory. In contrast, decentralized algorithms are useful in robotic applications where multiple computers are working in physically distinct areas without a central coordinator.

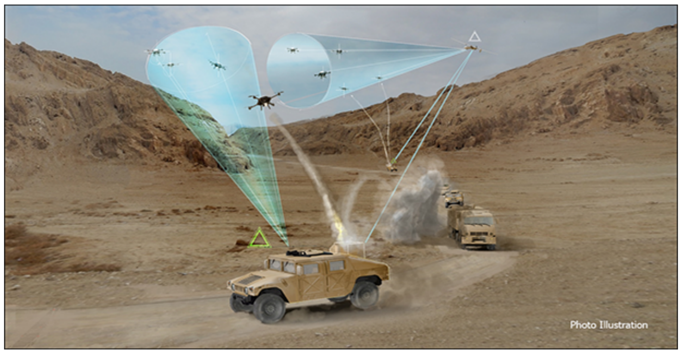
One difficulty with decentralized algorithms is that the agents may have disparate global information, either having limited, local information or disagreement on the overall team strategy [14]. If the agents all has the same information, each agent could run the equivalent planner and arrive at the same solution, not requiring any form of consensus or communication. However, if there is local information that is only accessible to individual agents, a common occurrence with limited range sensing, or some information is deemed private, such as robot location or task preference, team-wide communication will be need to arrive at a conflict-free solution. This incurs a large cost on the system, namely that the team must not only solve an NP-hard combinatorial problem, they must do so while also ensuring consistency either on their local information or agreement on the allocations themselves. This fundamental challenge has led to much research on analyzing consensus algorithms where agents must agree on global information.

As such, many decentralized algorithms utilize *implicit coordination* where the agents first agree on their state information and then can implicitly coordinate since they have agreement on state. The main drawback from these strategies is that sharing state information and reaching agreement is a high-bandwidth activity. Not to mention, it is not clear how to necessarily fuse the information from each agent into one cohesive picture. While attempts have been done in Cooperative SLAM [15] they still require large amounts of bandwidth to share their local measurements. Instead, the agents can share the allocation itself, without full agreement on their own measurements and state, to arrive at a coordinated decision known known as *plan consensus*[14]. While there is a trade-off in performance for scenarios with highly disparate information, this thesis assumes that agents generally have global consistency on the tasks information, i.e., they agree on the existence and location of the tasks. However, the local state information (e.g., position) and local reward function are not known by other agents (either for privacy or bandwidth reasons), requiring a decentralized algorithm to coordinate the allocation.

1.2.3 Dynamics

Lastly, this research differs from other approaches to task allocation and vehicle routing in that it is concerned with algorithms that can adapt to dynamic environments and tasks. In most decentralized task allocation problems, algorithms solve a static version of the problem. In the *static* decentralized task allocation problem, all tasks are known at the beginning of the algorithm and the environment does not change during the running of the algorithm. In reality, however, the environment is constantly changing with new information appearing and new tasks arriving. For example, in the vehicle routing problem, new customers may appear while the vehicles are traveling to their destinations or during the execution of the solver. In the Dial-A-Ride prob-


(a) Uber rideshare request [?]



(b) DARPA Mobile Force Protection. making [16]

Figure 1-3: Examples of dynamic tasks that arrive online while a network of agents. In ridesharing, a new customer appears that must be serviced. In defense applications, mobile units must detect and neutralize incoming attackers

lem, customer requests appear online and must be serviced by the vehicles in a similar fashion as many of the present ride sharing companies. If the problem is small enough (i.e., few tasks and agents) most modern solvers can be run periodically, returning answers at a faster time scale than the dynamics of the environment, incurring only minor delays in travel time. However, as the tasks and agents scale to larger numbers, repeating the expensive computation of obtaining routes or allocations no-longer becomes a burden on the system, and it can no longer return solutions on the time-scale of the dynamics. Thus in order to return faster solutions, especially in time-sensitive mission settings, teams must sacrifice some solution quality for speed so that the team can adequately respond to new tasks. Furthermore, dynamic algorithms should be able to reuse previous solutions when new information requires the team to replan rather than fully resolving the static task allocation problem. To address the realities of dynamic environments and tasks, this thesis will focus on the decentralized *dynamic* task allocation problem, specifically the problem of new tasks arriving into the system during and after the original solving of a static task allocation problem.

1.3 Contributions

This thesis extends the previous work on Consensus-Based Bundle Algorithm [17, 58, 59, 66**?** **?**] to allow for allocating new tasks without a full re-solving of the decentralized task allocation problem. Specifically, the main contributions of this thesis are:

1. (Chapter 2) An overview of related works in decentralized dynamic task allocation, with specific attention to the dynamic vehicle routing problem and decentralized optimization methods such as Consensus-Based Bundle Algorithm (CBBA)

2. (Chapter 3) A replanning algorithm called Consensus-Based Bundle Algorithm with Partial Replanning (CBBA-PR) that allows for provable convergence while increased coordination that can be tuned by a team for various response times

3. (Chapter 4) Methods for effectively allocating a new task with only a limited resetting. This includes a bidding strategy for single task resets, a heuristic for evaluating no reset performance, and a subteam formation algorithm for decreasing network diameter

4. (Chapter 5) Experimental study on effects of real ad-hoc communication on a team's ability to reach consensus using Raspberry Pi's

5. (Chapter 6) Description of hardware and simulation platform for testing decentralized planning and communication algorithms

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Preliminaries

This chapter includes an introduction to consensus theory which is fundamental to any multi-agent planning algorithm that requires agreement on global information. Then an overview of related works in centralized dynamic vehicle routing algorithms to provide an overview of state-of-the-art approaches to solving the NP-hard problem. This chapter also presents an overview of fully decentralized algorithms including decentralized task assignment, optimization, and auction algorithms. Finally, the Consensus-Based Bundle Algorithm is described which is the basis for the algorithms presented in the subsequent chapters of this thesis.

2.1 Centralized Dynamic Vehicle Routing Methods

One of the earliest instances of dynamic task allocation is the Dynamic Vehicle Routing Problem (DVRP), an NP-hard problem. in which new customers dynamically appear and must be visited by the vehicles along a route of customer pickup. Refs. [18] and [19] provide excellent literature reviews on centralized algorithms that handle dynamic customers, with most methods having been in the operations research community for distribution optimization. As the optimal solution can only be found by enumerating all possible routes or allocations, most methods for solving the vehicle routing program utilize heuristics to direct the search of solutions. Shared memory solutions [20] are commonly used where shared pools of feasible solutions or customer information are used by multiple agents to iteratively improve solutions. In the Ant Colony System [21], agents create and share an a priori heuristic $\eta_{i,j}$ and a posteriori pheromone $\tau_{i,j}$ that are used to guide agents in adding new customers to their existing routes. In this case, the pheromones are updated when the servicing of customer j after customer i yields a good solution. This is then shared by the agents and can be reused when a new customer arrives. In such a case, the problem is re-solved after updating the shared pheromone matrix

$$\tau'_{ij} = (1 - \gamma_r)\tau_{ij} + \gamma_r \tau_0 \tag{2.1}$$

where γ_r is a parameter that regulates the pheromone matrix and τ_0 are the initial pheromones from the beginning of the static problem for each customer pair.

Another shared memory approach is Tabu Search [22], where an initial depth first search is performed to obtain a feasible solution assigning vehicles to customers (or robots to task), and then a set of swaps are allowed to iteratively improve the solution. Since Tabu Search iteratively improves its solution it can be extended to dynamically arriving customers by allowing swaps that include the new tasks [23, 24].

Finally, in Genetic Algorithms [25], a pool of solutions are initially maintained and then an offspring solution is created from randomly chosen parents. The fitness of the offspring is related to the quality of the child solution such that better solutions are more likely to survive and create subsequent solutions. In the dynamic version of the Genetic Algorithm, the optimization is re-solved and can be sped up by anticipating future customers and preemptively solving for future customers [26].

While the methods above can be distributed across agents [24, 27**?** , 28], they still require a centralized event planner and global information so that each agent can independently solve the vehicle routing problem. In the fully decentralized problem, no single agent can coordinate the planning process and generally only a limited local information will be known to each agent. In addition, no runtime convergence is provided for these algorithms, rather they are assumed to find a feasible solution quickly and iteratively improve plans for a predetermined planning time. In time-sensitive and decentralized algorithms, simply agreeing on a solution across the team is difficult, in addition to finding one of sufficient quality.

2.2 Consensus Theory

In many multi-agent missions, agents must reach team-wide agreement on global information state, this could be classifying measurement across a sensor network [29], a collective action such as flocking [30], or a combination of estimation and control [31]. In all these problems, agents have limited communication with a neighborhood $\mathcal{N}_i$ of agents, generally limited by physical distance between robots. Agents can usually reliably communicate with their neighboring agents in a bidirectional manner, though random links have been considered [32–35].

The communication topology of the team can be represented by a directed graph $G = (V, E)$ with a set of nodes $V = \{1, 2, \dots n_r\}$ representing every agent $i \in \mathcal{I}$ and set of edges $E = \{(i, j) \in V \times V\}$ representing the communication connections between agents. The neighbors of agents with whom agent i can communicate is denoted as $\mathcal{N}_i = \{j \in V : (i, j) \in E\}$. It is convenient to represent the topology of G using an *adjacency* matrix $A = [a_{ij}]$ where $a_{ij} = 1$ if i, j are neighbors in the graph G.

The foundational consensus protocol and convergence analysis known as the average-consensus is described in [36, 37]

$$\dot{x}_i(t) = \sum_{j \in \mathcal{N}_i} a_{ij}(x_j(t) - x_i(t)) \tag{2.2}$$

Specifically, the linear system above converges to the average initial state of the system:

$$x(\infty) = \frac{1}{n} \sum_i x_i(0) \tag{2.3}$$

A major contribution of [36] was its analysis of the general class of $\mathcal{X}$-consensus algorithms which includes the average consensus ($\frac{1}{n}\sum_i x_i$), max-consensus ($\max_i x_i$), and min-consensus ($\min_i x_i$) in terms of the topology of the network, specifically the Laplacian of G. They show that the dynamics of the linear system in Eqn. 2.2 can be rewritten in the compact form:

$$\dot{x} = -Lx \tag{2.4}$$

where L is the graph Laplacian of G defined as

$$L = D - A \tag{2.5}$$

where D is the degree matrix of G and $d_i = \sum\limits_{i \neq j} a_{ij}$ and zero off diagonal elements. For the directed network, the second smallest eigenvalue of the Laplacian λ_2, also known as the algebraic connectivity of a graph, provides a measure of the speed of consensus algorithms. Specifically, Ref. [36] shows that the disagreement vector δ exponentially vanishes with a speed of at least λ_2.

2.3 Decentralized Subgradient Methods

A fully decentralized algorithms is provided by Ref. [38] for minimizing the sum of costs for a team of agents where each agent has a local cost function $f_i(x)$ where $f_i : R^n \to R$, and $f_i(x)$ is only known agent i. Then the optimization can be formulated as

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{R}} f_i(x) \\ \text{subject to:} \quad & x \in R^n \end{aligned} \tag{2.6}$$

where $x \in R^n$ is a global decision vector that the team must agree upon.

The main contribution of the subgradient method is that it allows each agent to locally compute a gradient using its own local cost function while sharing information with its teammates to reach both agreement on the global decision vector and obtain a globally optimal solution. Rather than computing a global gradient, as would be the case in a single-agent optimization, a subgradient is computed by agent, where the subgradient of the value function $s_F(\bar{x})$ defined as

$$F(\bar{x}) + s_F(\bar{x})'(x - \bar{x}) \leq F(x)\ \forall x \in \text{Domain}(F)\ . \tag{2.7}$$

The subgradient can be viewed as the partial derivative respect to the vector $\bar{x}$ which can then be used in a local descent towards new decision vector that lowest the local cost function.

In addition, to ensure that the agents jointly optimize their team value function and reach an agreed solution, each agent shares their subgradients with their neighbors, learning from the neighbors the team-wide gradient. They propose the following local update rule to optimize the team-wide function

$$x^i(k+1) = \sum_{j=1}^{n_r} a^i_j(k) x^j(k) - \alpha^i(k) d_i(k) \tag{2.8}$$

where $a^i(k) = (a^i_1(k), \ldots, a^i_m(k))'$ is a vector of weights, $\alpha^i(k) > 0$ is a scalar step-size used by agent i, and the vector $d_i(k)$ is a subgradients of agent i's value function $f_i(x)$ at $x = x^i(k)$. Notice that the update rule 2.8 is similar to a consensus rule, where the agents are simply reaching consensus on some decision vector $x^j(x)$. The first term acts as a consensus term which allows the agents to reach agreement on a decision vector and the second term relates to the gradient descent towards an optimal solution. The authors also provide a proof of the optimality for the solution provided by the distributed subgradient algorithm, showing that the cost is a constant factor from the optimal solution. In addition, they analyze the convergence rate based on the step-size α used in the gradient descent.

Improvements to the original decentralized subgradient method include agents with capacity limits [39] and agent dynamics [40]. Two main assumptions are used in the subgradient methods that do not enable its use in dynamic task allocation. First, the value function itself is convex and second, the decision vector has continuous values, allowing for the computation of subgradients. In task allocation, the decision vector corresponding to the allocations are binary $\bar{x} \in \{0, 1\}$ where 1 denotes that a task is assigned to agent i. In addition, the value function for task allocation is not always guaranteed to be convex and thus gradient approaches may not converge to a solution.

2.4 Decentralized Simplex Methods

For a subset of task assignment problems, those that can be represented as a linear program, simplex methods such as the Hungarian method [13] can be used to provide solutions. In the task assignment problem, where each agent is assigned a single task, the optimization can be formulated as the following integer linear program:

$$\begin{aligned} \max \quad & \sum_{i,j} c_{ij} x_{ij} \\ \text{subject to:} \quad & \sum_{i=1} x_{ij} = 1 \quad \forall j \\ & \sum_{j=1} x_{ij} = 1 \quad \forall i \\ & x_{ij} \geq 0 \end{aligned} \tag{2.9}$$

where x_{ij} is a binary assignment variable $\{0, 1\}$ and the cost function is a matrix of costs that is indexed by robot i and task j. Note that unlike the cost function in the task allocation optimization 1.1.2, the cost c_{ij} is only a function of i and j, not the entire allocation $x_i, \boldsymbol{p}_i$. As a result, the cost function is coded as a static matrix c_{ij} that is common in linear program, leading to what as known as the linear sum assignment problem (LSAP). This program can be solving using its dual program

$$\begin{aligned} \max \quad & \sum_{j} \beta_j - \sum_{i} \alpha_i \\ \text{subject to:} \quad & \beta_j - \alpha_i \leq c_{ij} \quad \forall i, \forall j \end{aligned} \tag{2.10}$$

The dual problem can be thought of an equivalent optimization where α_i is the price that each robot i will pay to do task j and β_j is the reward for servicing task j. In the dual problem, the constraint is that the price that robot i is willing to pay for the task can not exceed the reward of the task. According to duality in linear program, the optimum of the dual problem will be equal to the optimum of the primal, i.e., the cost total of that robot i would do will be exactly the cost for doing the task. A popular approach to solving this dual-primal problem is the Hungarian method which iteratively creates a feasible dual solution and then iteratively improves by constructing a bipartite graph. Refs. [41] and [42] provide decentralized methods for sharing the edges that are chosen between iterations, successfully distributing the Hungarian method. Both methods provide a convergence guarantee of $O(n_t^3)$ time until the team converges to the same optimal solution. While the task assignment problem can be extended to multi-task problems, where multiple tasks must be assigned to the agents, a matrix c_{ij} is still needed to formulate the ILP, requiring that the cost of any given task-agent pair is *independent* of x_{ij}. Simply put, the task assignment assumes that each task can be assigned independently to each agent without affecting the specific costs of the other tasks. In task allocation problems where the value function is dependent on the *entire* allocation, such as in vehicle routing problems, an ILP can not be formulated and the decentralized Hungarian method can not be used.

2.5 Auction Algorithms

Auction algorithms were one of the first distributed algorithms proposed for convex optimization based on the economic parallels of optimization and price auctioning [43–46]. First formalized in Ref. [43], the auction algorithm consists of agents bidding for each task based on the agent's personal score function and the maximum bidding being assigned to the new task by a central auctioneer. Specifically, if an agent i can increase its score by servicing a new task j^* the agent will bid the marginal increase or profit margin for the task

$$b_{ij^*} = a_{ij^*} - w_{ij^*} + \epsilon\,, \tag{2.11}$$

where $w_{ij^*} = \max\limits_{j \neq j^*} a_{ij} - p_j$.

Then, during the assignment phase, the task is given to the agent with the highest bid

$$p_j = \max_i b_{ij} \,. \tag{2.12}$$

More importantly, at the end of this auction process, the team converges to assignment which is within $n\epsilon$ of the optimal solution. Note that the ϵ is needed if ties to remove cycles due to ties, however, if a strict tie-breaking rule is always used (such as higher ID wins) then ϵ can be made arbitrarily small and reach the optimal solution. A limitation of this original process is that it is not completely decentralized, as a central auctioneer is still used to mark a winner for the team. In addition, this considers each task independently and not the combinatorial nature of a task allocation problem, and thus the conventional auction algorithm can not be used in the dynamic task allocation problem.

One of the strengths of the auction algorithm is that once an adequate utility function is found for a given problem, many other problems can be solved in a distributed fashion. For example, the auction algorithm can be applied to coverage control, where a team of agents are tasked with covering a specific area [47–49]. In one of the first robot applications of the free-market auction algorithm, Ref. [49] presents an auction algorithm for controlling a group of robots that must visit various cities in a traveling salesperson problem, where the reward function is related to the distance of each robot to the city. Auction based methods have also been expanded to the patrolling problem, where robots must continuously monitor various locations [50] and applied to distributed imaging in agricultural settings [51] where UAVs bid on areas to visit in order to maximize the amount of field imaged. In addition, others have derived decentralized methods for auctioning tasks using consensus across a team [45, 52–55].

In addition, auction algorithms can address bandwidth concerns for real communication networks as it effectively encodes agent preferences into lower bandwidth auction bids. In addition, recent work has analyzed the robustness of various auction algorithms in harsh communication environments [56], enabling these algorithms to be utilized in real-world robot communication settings.

2.6 Consensus-Based Bundle Algorithm (CBBA)

The main contributions of this thesis are based on the Consensus-Based Bundle Algorithm, the first fully decentralized algorithm for solving the task allocation problem for a team of robots. In the following section, the CBBA algorithm is described in detail and an explanation for the mechanisms leading to its convergence and optimality guarantees.

Consensus-Based Bundle Algorithm [17] is a decentralized auction based algorithm designed to solve the static task allocation problem, where all the task are known at the beginning. The algorithm alternates between two main phases: the *bundle building* phase and the *consensus* phase of the algorithm (Figure 2-1). In the bundle building phase, the agents iteratively generate a list of tasks to service by bidding on the marginal increase for each task. In the consensus phase, the agents resolve differences in their understanding of the winners of each task. Before proceeding, the following five lists used in CBBA will be defined:

1. A *path*, $\boldsymbol{p}_i \triangleq \{p_{i1}, \dots p_{i|\boldsymbol{p}_i|}\}$ is a list of tasks allocated to agent i. The path is in the order by which agent i will service the tasks.

2. A corresponding *bundle*, $\boldsymbol{b}_i \triangleq \{b_{i1}, \dots b_{i|\boldsymbol{b}_i|}\}$ is the list of tasks allocated to agent i in the order by which agent i *bid* on each task, i.e., task b_{im} is added before b_{in} if $m < n$. The size of $\boldsymbol{b}_i$, denoted $|\boldsymbol{b}_i|$ cannot exceed the size of $\boldsymbol{p}_i$ and an empty bundle is denoted $\boldsymbol{b}_i = \varnothing$.

3. A list of winning agents $\boldsymbol{z}_i \triangleq \{z_{i1} \dots z_{in_t}\}$, where each element $z_{ij} \in \mathcal{I}$ indicates who agent i believes is the winner of task j for all tasks in $\mathcal{J}$. If agent i believes that no one is the winner of task j, then $z_{ij} = -1$.

4. A corresponding list of winning bids $\boldsymbol{y}_i \triangleq \{y_{i1} \dots y_{in_t}\}$ where y_{ij} is agent i's belief of the highest bid on task j by winner z_{ij} for all j in $\mathcal{J}$. If agent i believes that no one is the winner of task j, then $y_{ij} = -\infty$.

5. A list of timestamps $\boldsymbol{s}_i \triangleq \{s_{i1}, \dots s_{in_r}\}$ where each element s_{ik} represents the

Algorithm 1 CBBA Phase 1: Bundle Build

```
1: $\boldsymbol{y}_i(t) = \boldsymbol{y}_i(t-1)$
2: $\boldsymbol{z}_i(t) = \boldsymbol{z}_i(t-1)$
3: $\boldsymbol{b}_i(t) = \boldsymbol{b}_i(t-1)$
4: $\boldsymbol{p}_i(t) = \boldsymbol{p}_i(t-1)$
5: while $|\boldsymbol{b}_i(t)| < L_t$ do
6:     $c_{ij} = \max_{n \le |\boldsymbol{p}_i(t)|+1} S_i^{\boldsymbol{p}_i(t) \oplus_n j} - S_i^{\boldsymbol{p}_i(t)}, \forall j \in \mathcal{J} \setminus \boldsymbol{b}_i(t)$
7:     $h_{ij} = I(c_{ij} \ge y_{ij}), \forall j \in \mathcal{J}$
8:     $J_i = \arg\max_j c_{ij} \cdot h_{ij}$
9:     $n_{i,J_i} = \arg\max_n S_i^{\boldsymbol{p}_i(t) \oplus_n J_i}$
10:    $\boldsymbol{b}_i(t) = \boldsymbol{b}_i(t) \oplus_{end} J_i$
11:    $\boldsymbol{p}_i(t) = \boldsymbol{p}_i(t) \oplus_{n_{i,J_i}} J_i$
12:    $y_{i,J_i}(t) = c_{i,J_i}$
13:    $z_{i,J_i}(t) = i$
14: end while
```

timestamp of the last information that agent i received about a neighboring agent k, either directly or indirectly.

2.6.1 Phase 1: Bundle Building

Unlike other algorithm which enumerate every possible allocation of tasks for agent i, in CBBA the agents greedily bid on a bundle of tasks. In the bundle building phase (Algorithm 1), an agent i determines the task J_i that will yield the maximum increase in marginal score when inserted into its previous path. If this score is larger than the current team winner, agent i will add the task J_i to its bundle. This process is repeated until it can no longer add tasks to its path, concluding by updating its list of winners and bids, $\boldsymbol{z}_i$ and $\boldsymbol{y}_i$.

2.6.2 Phase 2: Consensus

In the second phase of CBBA, each agent i communicates their updated lists, $\boldsymbol{z}_i, \boldsymbol{y}_i$ and $\boldsymbol{s}_i$ to their neighboring agents and resolve any conflicts in their belief of winners. If two neighbors disagree on a specific task $\bar{j}$ located at location $\bar{n}_i$ in their bundles, the two agents are required to reset not only task $\bar{j}$ but also any tasks located in the

Algorithm 2 Centralized Sequential Greedy Algorithm (SGA)

```
1: Given $\mathcal{I}, \mathcal{J}$
2: for $n = 1 \dots n_t$ do
3:     $(i_n^\star, j_n^\star) = \arg\max_{i,j \in \mathcal{I} \times \mathcal{J}} c_{i,j}$
4:     $\mathcal{J} \to \mathcal{J} \setminus \{j_n^\star\}$
5:     $\boldsymbol{b}_i(t) = \boldsymbol{b}_i(t) \oplus_{end} J_i$
6:     $\boldsymbol{p}_i(t) = \boldsymbol{p}_i(t) \oplus_{n_i,J_i} J_i$
7: end for
8: $c_{ij} = \max_{n \le |\boldsymbol{p}_i(t)|+1} S_i^{\boldsymbol{p}_i(t) \oplus_n j} - S_i^{\boldsymbol{p}_i(t)}, \forall j \in \mathcal{J} \setminus \boldsymbol{b}_i(t)$
9: $h_{ij} = I(c_{ij} \ge y_{ij}), \forall j \in \mathcal{J}$
10: $J_i = \arg\max_j c_{ij} \cdot h_{ij}$
11: $n_{i,J_i} = \arg\max_n S_i^{\boldsymbol{p}_i(t) \oplus_n J_i}$
12: $y_{i,J_i}(t) = c_{i,J_i}$
13: $z_{i,J_i}(t) = i$
```

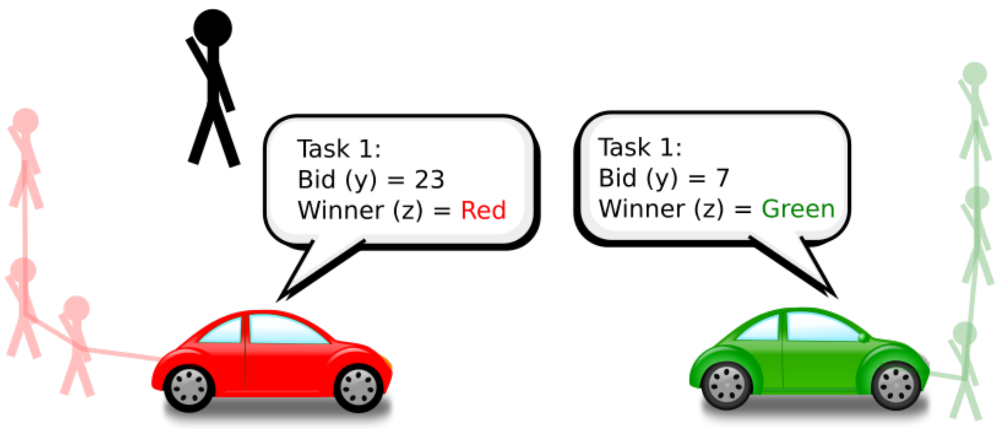
bundle after $\bar{n}_i$

$$\begin{aligned} &y_{i,b_{in}} = -\infty, \qquad z_{i,b_{in}} = -1 \quad \forall n > \bar{n}_i \\ &b_{in} = \varnothing, \quad n \ge \bar{n}_i \end{aligned} \tag{2.13}$$

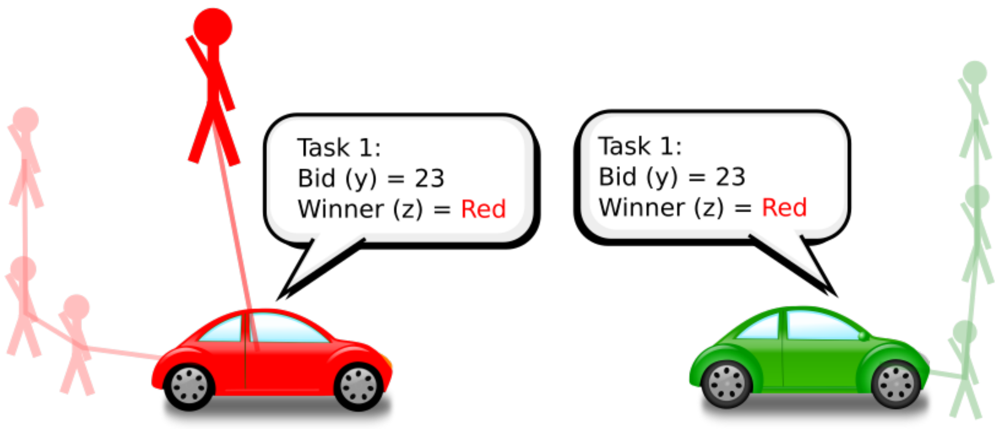
where b_{in} denotes the nth entry of bundle $\boldsymbol{b}_i$ and $\bar{n}_i = \min\{n : z_{i,b_{in}} \ne i\}$. The resetting of subsequent tasks is necessary for the proper convergence of CBBA, as the bids for those subsequent tasks $(y_{i,b_{in}})$ were made assuming a bundle consisting of the reset task $\bar{j}$.

2.6.3 Convergence of CBBA

Along with providing a procedure for decentralized allocation, Ref. [17] was able to show that CBBA is able to converge in $O(n_t D)$ rounds of communication, where D is the network diameter, and that CBBA converges to an approximate solution, with guaranteed 50% optimality for certain value functions. Specifically, they show that CBBA converges to the same solution as the centralized sequential greedy algorithm (SGA). SGA is a centralized algorithm (Algorithm 2) with a polynomial runtime of $O(n_t)$, during which a central planner assigns tasks iteratively to each agent. Once a task is assigned to an agent, it is removed from the pool of possible task to be assigned, allowing for a runtime that is linear with the number of tasks. In addition, the tasks are assigned greedily, sequentially choosing the task-agent pair which will



(a) Each agent initially calculates their bid for a task and then communicates the bid to its neighboring agents



(b) Once the agents receive neighboring bids, a winner is determined and each agent updates their belief on the winning agent and bid

Figure 2-1: The bundle building and consensus steps of Consensus-Based Bundle Algorithm in a vehicle routing scenario

lead to the largest team improvement. The convergence result is essential to CBBA as it provides the user with a convergence guarantee that is tractable and can be used for large team with many tasks. Practically, this means that users can guarantee that a conflict-free solution will be returned within a prescribed amount of time which may be of critical importance to a time-sensitive mission setting.

Since the mechanism by which CBBA converges to the SGA solution is utilized in subsequent proofs in this thesis, an overview of the important lemmas and proof from CBBA will presented here. To prove convergence and optimality of the algorithm, CBBA requires that the score function has diminishing marginal gains (DMG) where value functions are considered DMG if the following is true

$$c_{ij}[\boldsymbol{b}_i] \geq c_{ij}[\boldsymbol{b}_i \oplus_{end} \boldsymbol{b}] \tag{2.14}$$

for all $\boldsymbol{b}_i$, b, j. The DMG property roughly means that the score of any specific task will not increase if an additional set of tasks are added to the bundle. One result of DMG value functions is that the scores within an agent's own bundle will always be decreasing ($y_{b_{in},j} \geq y_{b_{im},j}\ \forall n > m$), a characteristic of the bidding that also leads to CBBA's convergence.

In addition, the proof relies on the following two lemmas, Lemmas 1 and 2 [17], to prove that during the running of CBBA the team sequentially agree on the SGA solution. Specifically, after $O(nD)$ rounds of communication, the team will agree on the first n tasks allocated using a sequential greedy allocation ($j_1^*, j_2^*, \ldots j_n^*$). Also, the bids for the task will be optimal, $y_{i,j_n^*} = c^*_{ij_n^*}\ \forall i \in \mathcal{I}$, and the agents will remain in agreement on those scores for the duration of the task allocation.

Lemma 1. *[17] Consider the CBBA process with synchronous conflict resolution over a static network with diameter D for the case that every agent's scoring scheme is DMG. Suppose that after completing phase 2 of some iteration t*

$$z_{i,j_k^*}(t) = i_k^*, y_{i,j_k^*}(t) = c^{(k)}_{i_k^*,j_k^*} \quad \forall i \in \mathcal{I} \quad \forall k \leq n \tag{2.15}$$

where (i_k^*, j_k^*)*'s are assignment pairs from the SGA procedure and* $c_{i_k^*, j_k^*}^{(k)}$*'s are the corresponding score values. Then, the following holds.*

1. *The first* $L_i^{(n)} \triangleq |\boldsymbol{b_i}^{(n)}|$ *entries of agent i's current bundle coincide with those of the bundle at the nth SGA step* $\boldsymbol{b_i}^{(n)}$

$$\boldsymbol{b_i}^{1:L_i^{(n)}} = \boldsymbol{b_i}^{(n)} \tag{2.16}$$

2. *The bid that agent* i_{n+1}^* *places on task* j_{n+1}^* *is*

$$y_{i_{n+1}^*, j_{n+1}^*} = c_{i_{n+1}^*, j_{n+1}^*}^{(n+1)} \tag{2.17}$$

and this value satisfies

$$y_{i_{n+1}^*, j_{n+1}^*} \geq y_{ij}(t) \forall (i,j) \in \mathcal{I}_{n+1} \times \mathcal{I}_{n+1} \tag{2.18}$$

3. *Entries in (22) do not change over time, or*

$$z_{i^*, j_k^*}(s) = z_{i, j_k^*}(t), \; y_{i^*, j_k^*}(s) = y_{i, j_k^*}(t) \tag{2.19}$$

for all $s \geq t$ *and for all* $k \leq n$*.*

4. *The value of the bid that agent* i_{n+1}^* *places on task* j_{n+1}^* *will remain the same throughout the later iterations, and no agents will bid higher than this value on task* i_{n+1}^* *in the later iterations:*

$$y_{i_{n+1}^*, j_{n+1}^*}(s) = y_{i_{n+1}^*, j_{n+1}^*}(t) \geq y_{i, j_{n+1}^*}(s) \tag{2.20}$$

$\forall s \geq t$ *and* $\forall i \in \mathcal{I}$

5. *After D iterations, every agent will have agreed on the assignment* (i_{n+1}^*, j_{n+1}^*)*;*

in other words

$$y_{i,j^*_{n+1}}(t+D) = y_{i,j^*_{n+1}}(t),\ z_{i,j^*_{n+1}}(t+D) = i^*_{n+1}, \tag{2.21}$$

for all $i \in \mathcal{I}$

Lemma 2. *[17] Consider a CBBA process with synchronized conflict resolution over a static network of diameter D, where every agent's scoring scheme is DMG. Then, every agent agrees on the first n SGA assignments by iteration nD. In other words,*

$$z_{i,j^*_k}(nD) = i^*_k \quad \forall i \in \mathcal{I} \quad \forall k \le n \tag{2.22}$$

$$z_{i,j^*_k}(nD) = c^{(k)}_{i^*_k,j^*_k} \quad \forall i \in \mathcal{I} \quad \forall k \le n \tag{2.23}$$

Combining Lemma 1 and Lemma 2, they are able to show that the bundles of each agent are sequentially built to include the same allocation as the SGA solution. As time goes on during the CBBA process, the agents begin agreeing on the first n tasks allocated by the centralized planner. In addition, the bids on those first n tasks are the SGA values, $c^{(k)}_{i^*_k,j^*_k}$ for all agents. Finally, the agents remain in agreement on the first n tasks throughout the CBBA process, "locking-in" on those allocations. This intuition of sequentially building agreement on the centralized SGA solution will be the basis of the replanning method presented in the next chapter, and by which the linear convergence is proven.

2.7 Summary

This chapter presented an overview of centralized and decentralized algorithms for various versions of the task allocation problem. In addition, an overview of consensus and auction algorithms were presented, which are the basis for most decentralized optimization and allocation algorithms, Finally, this chapter described the a decentralized allocation algorithm, CBBA, which included the proof of its convergence which will be utilized later in this thesis. While these methods provide solutions to various types of task allocation problems, none are able to tractably allocate new tasks for the combinatorial, decentralized task allocation problem. In the next chapter, the dynamic task allocation is specifically discussed and a novel approach to partially re-solving the task allocation problem is presented.

Chapter 3

Decentralized Dynamic Task Allocation by Partial Replanning

3.1 Replanning for a New Task

This chapter investigates the decentralized dynamic task allocation problem where a team of robots must respond to a new task that appears during or after the original allocating of tasks, with the goal of assigning the new task to an agent while considering their existing allocations. This is in contrast to the static task allocation problem which assumes that all the tasks are known before the team executes the task allocation solver. The Dynamic Vehicle Routing Problem (DVRP), or similar Dial-A-Ride Problem [57] where online requests occur during the operation of the vehicles, will be used as a running example to demonstrate an application of autonomous agents (cars) allocating tasks (riders). In addition, this chapter specifically seeks a decentralized algorithm that relies only on peer-to-peer communication to ensure robustness and scalability.

In the previous chapter, a literature review of decentralized static optimization algorithms were presented, concluding with a description of CBBA which provides an $O(n_t D)$ runtime algorithm with an optimality guarantee of 50%. As for dynamic algorithms, Ref. [?] presents an online solver by enforcing strict task swapping, but solves the task assignment problem in which task scores are independent of each other. Ref. [?] uses queuing theory to analyze the stability of a dynamic system given stochastic task arrivals and [18? ? ?] utilize partitioning algorithms for dividing up areas to service. In partitioning algorithms, the space is first divided (by a central planner) and assigned to the agents, such that each agent services any incoming tasks that arrive in the assigned regions. Ref. [?] proposed a learning method based on Markov games applying a Distributed Stochastic Algorithm to obtain policies that are learned offline and applied as tasks arrive. While Ref. [?] performs better than other search methods, they can only prove Nash equilibrium and thus may get stuck in local minima. Similarly, [? ?] use a simplified utility function to quickly allocate tasks to agents, however, task-specific value functions (based on time of service, specific agent) were not considered.

Recent improvements to CBBA include adaptations for dynamic network connectivity [66?] and servicing mission-critical tasks [?]. In Ref. [?], each agent must also reaches consensus on the Performance Impact (PI) which calculates the change in score if a task is added or removed from their bundle. The added information allows for iteratively improving the assignments at the expense of requiring additional consensus for each agent-task pair. Ref. [?] extends the PI framework to the dynamic case by initially allocating the new task to an available agent and then using the PI architecture to iteratively improve the allocation.

In [58], the authors more directly adapt CBBA to allow for new tasks by triggering a full replan, resetting any previous allocations, and restarting CBBA with all existing and new tasks. In this case, the team must wait in the worst-case $n_t D$ rounds of bidding before a conflict-free solution is obtained. In general, resetting the previous allocations (or bundles in CBBA) allows for increased coordination and better servicing of the new task. Figure 3-1 shows a ridesharing routing example where agents are not allowed to replan before adding a new task (customer) to their allocation, and thus are only able to consider inserting the new customer somewhere along its original allocation (route). While in Figure 3-2, agents allow teammates to service tasks previously allocated to them in order to better service the new task, improving the final solution quality of the team. In general, there will be a trade-off

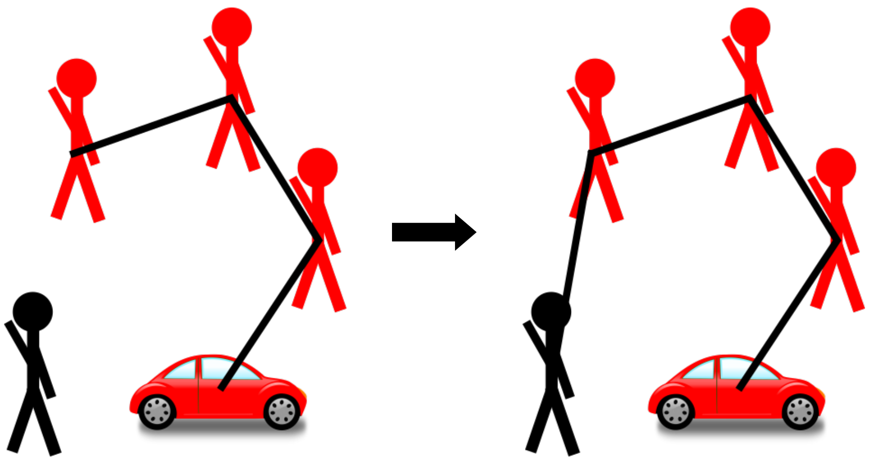


Figure 3-1: Agents allocation new task without allowing resetting of previous allocations. New task is then inserted into the agent's previous path.

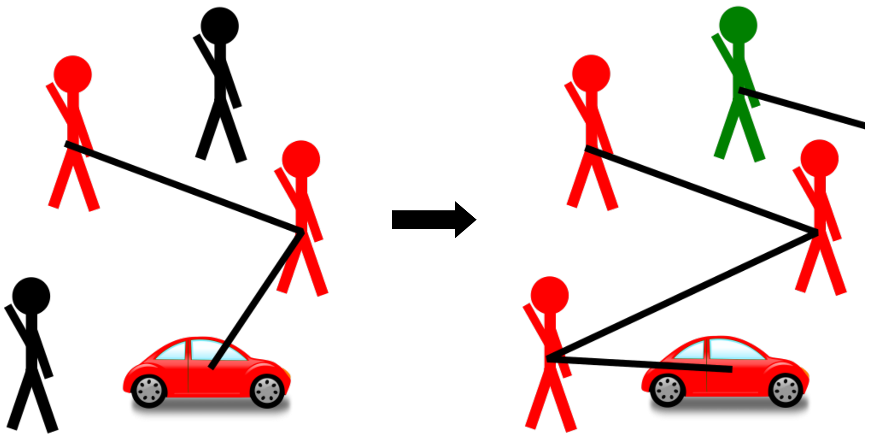


Figure 3-2: The agent may consider resetting a portion of its previous path to allow a neighboring agent to service an existing task (green) and enabling a better allocation of the new customer

between algorithms with increased coordination (resetting previous allocations) and those with quick convergence (keeping previous allocations). The following section further describes the benefits and costs of replanning via resetting in decentralized task allocation algorithms such as CBBA.

3.2 Effects of Replanning in Auction Algorithms

3.2.1 Benefits of Resetting Previous Allocations

Even in scenarios where a new task does not greatly impact the previous allocations, greedy auction algorithms such as CBBA may benefit from resetting their previous allocations and considering the new task. Since sequential greedy algorithms lock-in their previous allocations and only consider inserting a new task into their existing allocation, they are highly impacted and constrained by any previous allocations they made. For example, consider the scenario where a few of the agents are at capacity $|\boldsymbol{p}_i| = L_t$ at the time when a new task arrives. In a car sharing application, this can occur if a subset of agents are previously assigned a full car of passengers. When a new task arrives, the agents begin building bundles starting from their previous allocation, considering the optimal location to insert the new task. The subsequent bid for the new task is the marginal improvement to the *overall* bundle

$$y_{iT^*} = S(\boldsymbol{p}_i') - S(\boldsymbol{p}_i) = \max_n S(\boldsymbol{p}_i \oplus_n T^*) - S(\boldsymbol{p}_i), \tag{3.1}$$

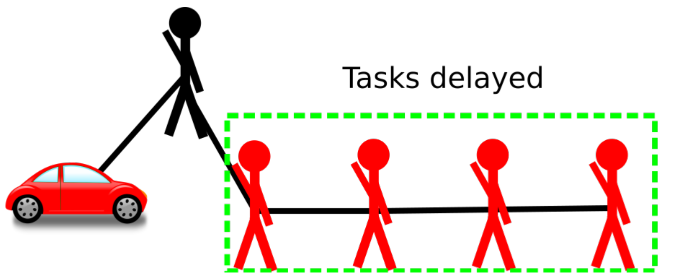
where $S(\boldsymbol{p}_i \oplus_n T^*)$ denotes that T^* is inserted into the path $\boldsymbol{p}_i$ at location n.

If agents are at capacity $|\boldsymbol{p}_i| = L_t$ they can not bid on the new task since they can no longer add the new task to their bundle. In effect, their previous allocation is *preventing* them from bidding on the new task. This can result in pathologically poor allocations if the task is high value and only few agents can properly service the task. In this scenario, it is clear that it may be beneficial to reset their allocations *before* bundle building to allow agents to bid on the new task.

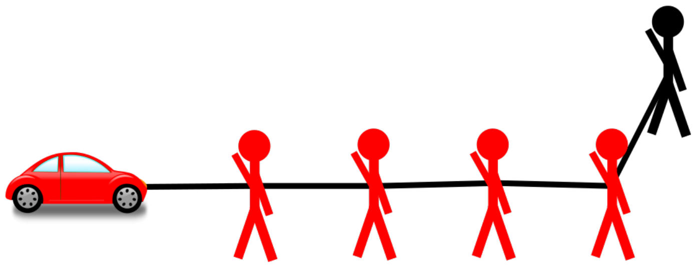
A more subtle benefit of replanning in CBBA that will be discussed below is that it elevates the bias of greedy auction algorithms to place the new task at the end of its existing path, as to not adversely affect the previously allocated tasks. For without any resetting, the marginal gain of placing a new task in the beginning of a path is diminished because the new tasks delays the execution of previously allocated tasks, reducing the score of those tasks. As an example, a common score function that can capture the time cost of delaying servicing is the time-discounted reward function

$$S(\boldsymbol{p}_i) = \sum_{j \in \boldsymbol{p}_i} \lambda_j^{\tau_j^{\boldsymbol{p}_i}} R_j \tag{3.2}$$

where $\tau_j^{\boldsymbol{p}_i}$ is the time to arrive at task j along path $\boldsymbol{p}_i$, R_j is the intrinsic value of the



(a) New task placed at beginning of path



(b) New task placed at end of path

Figure 3-3: Placing new task early in the path delays subsequent tasks reducing the marginal bid of the new task. Greedy bidding of CBBA causes a bias towards placing the new task at the end of its existing path to minimize task delays.

task, and $0 < \lambda_j \leq 1$ is the time-discount of the task. With a time-discounted reward, adding a new task will delay the arrival time of later tasks. As a result, an agent's bid on the new task will be discounted by any preallocated tasks located later in its path. Specifically, if a task T^* is placed at location n^* in an agent's bundle, the new marginal score will be

$$y_{iT^*} = \sum_{j \in \boldsymbol{p}_i'} \lambda_j^{\tau_j^{\boldsymbol{p}_i'}} R_j - \sum_{j \in \boldsymbol{p}_i} \lambda_j^{\tau_j^{\boldsymbol{p}_i}} R_j \,. \tag{3.3}$$

Note that $\boldsymbol{p}_i' = \boldsymbol{p}_i^{0:n^*} \oplus T^* \oplus \boldsymbol{p}_i^{n^*:|\boldsymbol{p}_i|}$ so substituting this for $\boldsymbol{p}_i'$ in (3.3) yields

$$y_{ij} = \left(R_{T^*} \lambda_j^{\tau_{T^*}^{\boldsymbol{p}_i'}} + \sum_{j \in \boldsymbol{p}_i} \lambda_j^{\tau_j^{\boldsymbol{p}_i'}} R_j \right) - \sum_{j \in \boldsymbol{p}_i} \lambda_j^{\tau_j^{\boldsymbol{p}_i}} R_j \tag{3.4}$$

$$= R_{T^*} \lambda_{T^*}^{\tau_{T^*}^{\boldsymbol{p}_i'}} + \sum_{j \in \boldsymbol{p}_i} \lambda_j^{\tau_j^{\boldsymbol{p}_i'}} R_j - \lambda_j^{\tau_j^{\boldsymbol{p}_i}} R_j \tag{3.5}$$

For simplicity, the first term will be denoted $S_{\boldsymbol{p}_i'}(T^*)$ as it represents the time-discounted reward of servicing T^* along the new path $\boldsymbol{p}_i'$. The second term is the difference in value for tasks that were in the original path $\boldsymbol{p}_i$. This term can be simplified by noting that the time-discounted value for a task located in the path *before*

T^* will be unaffected by the addition of T^*. Thus the second term can be simplified to

$$y_{ij} = S_{\boldsymbol{p}'_i}(T^*) + \sum_{j \in \boldsymbol{p}_i^{n:|\boldsymbol{p}_i|}} \lambda_j^{\tau_j^{\boldsymbol{p}'_i}} R_j - \lambda_j^{\tau_j^{\boldsymbol{p}_i}} R_j \tag{3.6}$$

$$= S_{\boldsymbol{p}'_i}(T^*) - \sum_{j \in \boldsymbol{p}_i^{n:|\boldsymbol{p}_i|}} (\lambda_j^{\tau_j^{\boldsymbol{p}_i}} - \lambda_j^{\tau_j^{\boldsymbol{p}'_i}}) R_j \tag{3.7}$$

$$= S_{\boldsymbol{p}'_i}(T^*) - \sum_{j \in \boldsymbol{p}_i^{n:|\boldsymbol{p}_i|}} \lambda_j^{\tau_j^{\boldsymbol{p}_i}} (1 - \lambda_j^{\delta}) R_j \tag{3.8}$$

where δ is the time delay due to servicing T^* which in this case will be a fixed delay for all tasks serviced after T^*. If the time-discount is constant across tasks, then the marginal gain can be written as

$$y_{iT^*} = S_{\boldsymbol{p}'_i}(T^*) - (1 - \lambda^{\delta}) S_{\boldsymbol{p}'_i}(\boldsymbol{p}_i^{n^*:|\boldsymbol{p}_i|}). \tag{3.9}$$

This bid contains two terms, the first is the time-discounted reward for servicing T^*, which will be highest when T^* is serviced early in the path. The second term is the discount that results from delaying the tasks located later in the path due to the insertion of T^*. This means that the optimal placement of T^* must trade-off placing T^* early in its bundle to allow for quick servicing of T^* with increased delaying of later tasks that are serviced after T^*.

While this trade-off is intrinsic to the dynamic task allocation problem, the greedy auction scheme used by CBBA exacerbates the issue in the case of dynamic new tasks as they must discount all the tasks allocated during the initial running of CBBA. If, however, the task was known at the beginning of CBBA, T^* would only be discounted by tasks previously allocated at that point in CBBA which will generally be few if T^* itself is a high-value task. In the extreme case, if the preexisting paths consists of many tasks, the placement of T^* will be biased towards the end of the path to minimize extrinsic discount (Figure 3-3) even if T^* is of relatively high value. However, had T^* be known at the beginning of CBBA, it could be placed earlier in the bundle without needing to consider later tasks. Thus running a full replan of CBBA can allow for a properly greedy allocation of T^* without being biased by previous allocations.

3.2.2 Costs of Resetting Previous Allocations

While full replanning allows for the highest level of coordination, it also is quite disruptive to the planning process, requiring substantial time to fully replan the allocations. First, there will be an explicit cost due to the delays of replanning. For example, if the agents must pause their execution of their plans to reach a new consensus then they will lose team-value due to the delayed execution of the tasks. If the team is allocating tasks that are not time-sensitive then the cost of replanning delays will be minimal. However, if tasks are time-sensitive and the environment is rapidly changing, the frequent delays due to replanning may cost more than the gains of better solutions that come from higher coordination.

A second cost of replanning is that it may inadvertently cause an instability to the system known as churning [**?**], where agents are constantly in a state of replanning and disagreement on the task allocations. For example, if tasks are sequentially arriving at a frequency that is faster than the time it takes to fully re-solve the problem, then the team will never actually execute their plans. Additionally, if the agents relax their requirement for consensus and begin executing their tasks without reaching full agreement with their teammates, agents may arrive at tasks that were previously serviced by a teammate, making their allocations less and less relevant.

3.3 Bundle Resetting in CBBA

For a quick response, one could consider absolutely no replanning, without allowing any resetting of an agent's previous allocation, $\boldsymbol{p}_i(t-1)$, $\boldsymbol{b}_i(t-1)$. This approach, which this thesis will call CBBA with *No Bundle Reset*, can be found in the original version of CBBA [17], having the Bundle Build process begin each round with $\boldsymbol{p}_i(t) = \boldsymbol{p}_i(t-1)$ and $\boldsymbol{b}_i(t) = \boldsymbol{b}_i(t-1)$. The advantage of CBBA with No Bundle Reset is that the convergence of the algorithm is virtually unaffected by the new task. For example, in the case where the team has already reached convergence on the original n_t tasks and

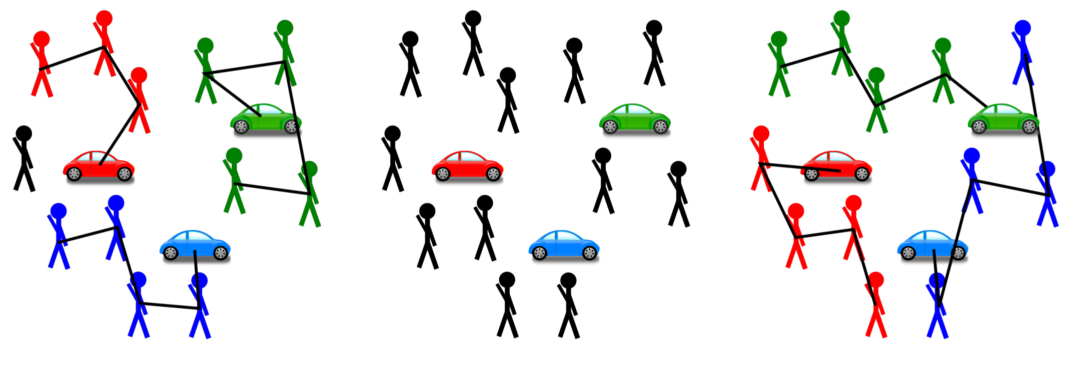


Figure 3-4: CBBA full reset strategy for allocation a new task

arrived at some allocations $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_i$, the agents will never consider reallocating their existing tasks and simply bid on inserting the new task into their existing bundles $\boldsymbol{p_i'} = \boldsymbol{p}_i \oplus_n T^*$ where n is the location for which inserting T^* maximizes the marginal gain of the agent. By effectively only bidding on T^* and not allowing any bidding on other tasks in its paths, the team is able to reach agreement very quickly in $O(D)$ time by simply communicating their single bid on T^*. While it is beyond the scope of this thesis to provide quality guarantees for the no reset solution, intuitively it is clear that a no reset solution provides very little flexibility to the robot team in allocating T^*. For example, in a highly constrained systems where many robots are at capacity $|\boldsymbol{p}_i(t-1)| = L_t$ or there are only a few robots that can service specific tasks, then only those robots under capacity and with the ability to service T^* will be considered for T^*. In these constrained scenarios, robot teams will need reset their previous allocations to consider the new task.

A later addition to CBBA was to begin the Bundle Build process by fully resetting the previous allocations, $\boldsymbol{b}_i(t) \to \varnothing$ and $\boldsymbol{p}_i(t) \to \varnothing$ [59]. This approach, CBBA with *Full Bundle Reset* (Figure 3-4), gives the agents maximum flexibility in allocating the new task, in that they are not bound by their previous allocations. While this full bundle reset increases the team coordination, one possible shortcoming of any bundle resetting approach is that it will no longer guarantee convergence for the original task allocation problem, as the algorithm is introducing additional resetting at each round of Bundle Build.

Claim 1. *If all tasks are known at the beginning of CBBA, both CBBA with Full Bundle Reset and CBBA with No Bundle Reset arrive at the SGA solution in* $O(n_t D)$

Proof. CBBA's convergence to the centralized sequential greedy algorithm's (SGA) solution relies on the fact that at some time t the team will agree on the first n tasks in the SGA solution and then subsequently agree on this solution for the rest of time (Lemma 1 [17]). The authors use induction to show that the team will first agree on the highest valued task (the first task allocated in the greedy solution) and after nD rounds of communication, will agree on the first n tasks in the SGA solution (Lemma 2 [17]). In the case of a full reset at the beginning of Bundle Build, ones needs to show that the reset will not break Lemma 1, i.e., that if the team agrees on the first n SGA tasks, they will continue to agree on those tasks for $s > t$. First, denote the list of agreed SGA tasks at time t, as $\mathcal{J}_n^* = j_1^* \dots j_n^*$ and the SGA winners of those tasks as $i_1^* \dots i_n^*$. Note that according to Lemma 1, at time t, all agents are in agreement on the bids for the first n-SGA tasks:

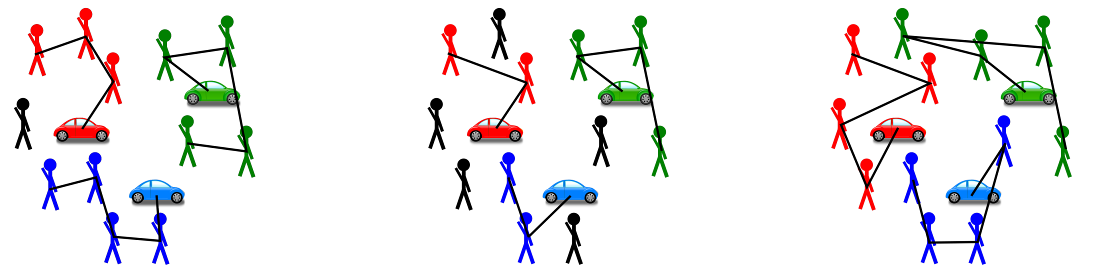
$$y_{ij} = c_{ij}^* \quad \forall j \in \mathcal{J}_* \quad \forall i \in \mathcal{I} \tag{3.10}$$

As such, at some time t, agent i will have a bundle $\boldsymbol{b}_i$ that consists of agreed-on SGA tasks $\boldsymbol{b}_i^{0:n_i^*}(t)$, where n_i^* is the number of tasks in $\mathcal{J}_{(n)}^*$ that are assigned to agent i by the SGA solution. The rest of the bundle will consist of other tasks from $\mathcal{J}$ that may or may not be in consensus with the rest of the team, $\boldsymbol{b}_i^{n_i^*:|\boldsymbol{p}_i|}(t)$. At time $t+1$, when the agent resets its bundle at the beginning of Bundle Build, it will begin greedily choosing tasks from $\mathcal{J}$ to add to its now empty bundle. However, when agent i calculates its own bid on a task j_k^* in $\mathcal{J}_n^*$ where $i_k^* \neq i$ (i.e., for tasks whose SGA winner is not i), agent i will always be outbid the current team winner since their bids are greedily optimal. Instead, agent i will first re-assign itself any of the tasks in $\mathcal{J}_n^*$ that have i as the SGA winner, since those tasks will have the highest bids for agent i by definition, since they are the centralized sequential greedy bids. As a result, after the full bundle reset the agent i rebuilds its first n_i^* in its previous bundle, $\boldsymbol{b}_i(t+1)^{0:n_i^*} = \boldsymbol{b}_i(t)^{0:n_i^*}$. This means that even in a full bundle reset, Lemma 1 and Lemma 2 hold, and thus convergence to the SGA is guaranteed in $O(n_t D)$. □

While both approaches, CBBA with Full Bundle Reset and CBBA with No Bundle Reset, converge to the same solution in $O(n_t D)$, when a new task is introduced during the execution of CBBA, these two approaches diverge in terms of solutions and convergence guarantees. First, in the proof above, the full reset converged to the sequential greedy solution because the Bundle Build process rebuilds the first part its previous bundle $\boldsymbol{b}_i(t)^{0:n_i^*}$, even after fully resetting its allocation. However, if a new task is now considered in the building process, agent i is not guaranteed to rebuild $\boldsymbol{b}_i(t)^{0:n_i^*}$. In fact, it may be the case that the sequential greedy solution for n_t+1 tasks, $\mathcal{J}'^*$, will be completely different to the solution for original static n_t task allocation problem. Thus if the team arrives at a solution to the initial n_t tasks and new task arrives, the agents may need an additional $O(n_t D)$ rounds of communication to allocate T^*. In summary, CBBA's existing approaches to allocating a new tasks is either to to allow a full rerunning of CBBA (full reset), requiring $O(n_t D)$ rounds of communication, or a quick consensus on a winner for the new task, without allowing any reallocation of the existing tasks (no reset).

3.4 CBBA with Partial Replanning (CBBA-PR)

3.4.1 Partial Resetting of Local Bundles



(a) Initial bundles $\mathbf{b}_1 \dots \mathbf{b}_i$ and new task T^*

(b) Each agent resets lowest $n_{i,reset}$ tasks in bundle

(c) Converges to modified allocations with T^*

Figure 3-5: Dynamic task allocation using CBBA-PR by partially resetting the last task in each agent's bundle at the beginning of Bundle Build. The tasks are chosen to be the last tasks auctioned in the bundle (not the order of physical path) to ensure convergence of CBBA-PR

Algorithm 3 CBBA-PR with Partial Local Replan (Fixed Bundle Size)

```
1: $\mathcal{J}_{i,reset} = \{b_{im}(t-1)\ \forall m \geq n_{reset}\}$
2: for all $j \in \mathcal{J}_{i,reset}$ do
3:     $\boldsymbol{b}_i(t) = \boldsymbol{b}_i(t-1) \ominus j$
4:     $\boldsymbol{p}_i(t) = \boldsymbol{p}_i(t-1) \ominus j$
5:     $z_{i,j}(t) = -1$
6:     $y_{i,j}(t) = \infty$
7: end for
8: Phase 1: Bundle Build($\boldsymbol{p}_i(t), \boldsymbol{b}_i(t), \boldsymbol{y}_i(t), \boldsymbol{z}_i(t)$)
9: Phase 2: Consensus
```

To better trade-off coordination with the speed of convergence, the CBBA with Partial Replan (CBBA-PR) is proposed, which enables each agent to reallocate a portion of their existing allocation at each round of CBBA. In CBBA-PR, each agent resets part of their bundle at the beginning of Bundle Build, releasing their $n_{i,reset}$ lowest bid tasks from their previous bundles (and keeping the remaining tasks). The $n_{i,reset}$ can be chosen by the team depending on the amount of replanning or response speed that is necessary for the team. For example, in the case where new tasks are frequently appearing and the team wants to converge before another new task arrives, they may choose $n_{i,reset}$ to be very small. On the other hand, if the new tasks are particularly high-valued, the team can allow for more coordination by selecting a larger number of tasks to reset. Furthermore, the amount of resetting may change during the duration of CBBA. If the new task arrives early on in the team's allocation of the original n_t tasks, they may allow for more resetting. While if the team has already converged on all n_t original tasks, they may limit the amount of resetting, to not waste the computation for the original tasks.

An important requirement for the tasks chosen for resetting is they must be the *lowest* tasks in each agent's respective bundles. This is to ensure the convergence of CBBA, for if tasks are reset in any other order (randomly chosen or maximum bids), CBBA will not have diminishing valued bids, and the team will not converge to a conflict-free solution. Instead, if the agents reset only the lowest n_{reset} tasks in each bundle to reset, Lemmas 1 and 2 can be re-used to prove that the team sequentially agree on a conflict-free solution.

Algorithm 4 CBBA with Partial Team Replan

```
1: Given: $\mathcal{I}_{reset}$, $t_{response}$
2: $d = Diameter(\mathcal{I}_{reset})$
3: $n_{reset} = \frac{t_{response}}{d \times \Delta_{comm}}$
4: $\boldsymbol{y}_i^s = Sort(\boldsymbol{y}_i)$
5: $\mathcal{J}_{reset} = \boldsymbol{y}_i^s[n_{reset} : n_t]$
6: for all $j \in \mathcal{J}_{reset}$ do
7:     $\boldsymbol{p}_i(t) = \boldsymbol{p}_i(t-1) \ominus j$
8:     $\boldsymbol{b}_i(t) = \boldsymbol{b}_i(t-1) \ominus j$
9:     $y_{ij}(t) = -\infty$
10:    $z_{ij}(t) = -1$
11: end for
12: Phase 1: Bundle Build($\boldsymbol{p}_i(t), \boldsymbol{b}_i(t), \boldsymbol{y}_i(t), \boldsymbol{z}_i(t)$)
13: Phase 2: Consensus
```

3.4.2 Partial Resetting with Team-Wide Reset

One limitation of the local partial reset strategy is that while average convergence will generally be better than a full reset, one can not guarantee that worst-case performance will improve. For example, if an agent only has one task to reset, and that task happens to be the first task in the centralized SGA solution, a full replan may occur. However, if the team has converged on the first n_t tasks before T^* arrives, then a worst-case performance can be guaranteed of $O(n_{reset}D)$ where $n_{reset} = n_r \times n_{i,reset}$ is the *total* number of tasks reset by the team. In this scenario, the team can choose the n_{reset} lowest bid tasks from across the entire team. Since the team has already reach consensus on the original centralized greedy solution, those n_{reset} lowest solutions will in fact be the last n_{reset} tasks allocated by the SGA. Since the higher bid tasks will remain allocated after the partial reset, the team is guaranteed to converge within $O(n_{reset}D)$ rounds of communication.

In this procedure, CBBA with Partial Team Replan (Algorithm 4), when a new task appears, each agent sorts the final bid array $\boldsymbol{y}_i$, enabling the agents to identify the n_{reset}-lowest SGA tasks, $\mathcal{J}_{reset}$ (Line 4). Any agent with a task from $\mathcal{J}_{reset}$ in their previous bundle, will reset the task by removing it from $\boldsymbol{b}_i$ and $\boldsymbol{p}_i$ and resetting the values in $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$. By doing so, the team is able to get increased coordination from reallocating existing tasks while still guaranteeing convergence that is $O(n_{reset}D)$, where n_{reset} can be chosen to fit the team's desired response time. In addition, if only a subset of the team $\mathcal{I}_{reset}$ is chosen to participate in the replanning, the team can reuse the known assignments in $\boldsymbol{z}_i$ to specifically reset n_{reset} tasks that were assigned to agents in $\mathcal{I}_{reset}$, ensuring that none of the reset tasks are "wasted" on agents that are not participating in the replan. Conversely, the team can choose a combination of n_{reset} tasks and desired subteam of diameter d, reusing $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ to achieve replanning within a desired convergence. With this subteam and subtask selection, the team can choose between selecting a large subteam with few tasks per robot to reallocate or a small subteam with robots fully resetting previous allocations. In general, this ideal mix of d and n_{reset} for a given scenario will be dependent on the mission characteristics.

3.5 Convergence

To prove the convergence of the two partial replan algorithms, first, the convergence of a local reset is proven for the static task allocation problem, yielding the same convergence and solution as a full reset or no reset strategy. Then, the convergence is shown when a new task must be allocated using a partial replan, which is shown to be related to the relative SGA ordering of those tasks reset during the replan. Finally, the convergence of CBBA-PR is derived for the case when specifically the lowest n_r tasks from the original running of CBBA are reset before T^* is allocated, and then the more general case of a local bundle reset.

3.5.1 Partial Resetting During Static Task Allocation

Claim 2. *Suppose that at time $t-1$ the team agrees on the first n SGA tasks, and thus have bundles that correspond to the nth round of SGA $\boldsymbol{b}_i(t-1)^{1:L_i^{(n)}} = \boldsymbol{b}_i^{(n)}$.*

If agent i resets its bundle to $\boldsymbol{b}_i^{-} = \boldsymbol{b}_i(t-1)^{1:n_r}$ then it will immediately rebuild $\boldsymbol{b}_i(t) = \boldsymbol{b}_i^{(n)}$ during Bundle Build.

Proof. Let's denote the SGA tasks that are released during the reset process $\boldsymbol{b}_i(t-1)^{n_r:L_i^{(n)}} \triangleq \{j^*_{k_{i1}}, j^*_{k_{i2}}, \dots j^*_{k_{ir}}\}$, where $k_{i1} \dots k_{ir}$ equals the SGA round for which task

$j^*_{k_r}$ is assigned in the central SGA. Note that $\boldsymbol{b}_i$ may have more tasks in the bundle, but for convergence analysis only the tasks that coincide with the SGA bundle $\boldsymbol{b}_i^{(n)}$ matter.

Because bundles are built incrementally, the SGA rounds corresponding to each reset task will be in increasing order: $k_{i1} < k_{i2} < \ldots < k_{ir} < n$ (i.e., tasks that are earlier in the bundle correspond to tasks allocated earlier in SGA). Likewise, because the scores are monotonically decreasing in subsequent rounds n of SGA,

$$c_{i^*,j^*_{k_1}} > c_{i^*,j^*_{k_2}} > \ldots > c_{i^* j^*_{k_r}} \tag{3.11}$$

.

In addition to shortening its bundle $\boldsymbol{b}_i$, agent i will also update its information arrays for the tasks it reset: $y_{ij} = -\infty \; \forall j \in \boldsymbol{b}_i^{n_r:L_i^{(n)}}$. Importantly, i does not change winning bids on any tasks allocated to other agents so, $z_{\tilde{i}j}(t) = z_{\tilde{i}j}(t-1)$ for $\tilde{i} \neq i$. Additionally, because all the agents agreed to the first n SGA tasks at the time of reset, $z_{i,j^*_k} = i^*_k$ and $y_{i,j^*} = c^{(k)}_{i^* j^*_k}$ for $i^*_k \neq i$ and $k \leq n$. This means that every agent is in global agreement on the bids for tasks allocated at rounds $k \leq n$.

At the beginning of Bundle Build, each agent i will choose the task $J_i \in \mathcal{J} \setminus \boldsymbol{b}_i^{1:n_r}$ with the highest rewards that also outbids any teammates:

$$J_i = \arg\max_j c_{ij} \cdot \mathbf{1}(c_{ij} > y_{ij}) \tag{3.12}$$

where the second term ensures that agents only add tasks if they outbid teammates and the first term selects the task with the maximum score.

If any agent i considers adding an "earlier" SGA task, i.e., bidding $c_{ij^*_k}$ on task j^*_k where $k < k_{i1}$, then i will be outbid by i^*_k (the true SGA winner for j^*_k) since $y_{i,j^*_k} = c^{(k)}_{i^* j^*_k}$ and $c_{ij^*_k} < c^{(k)}_{i^* j^*_k}$. So, agent i will not consider adding any task that has an earlier SGA round than k_{i1}.

As such, i's first non-zero bid will be for $j^*_{k_{i1}}$ which will also be the maximum score as it is the next earliest SGA task (and thus scores for subsequent tasks $j^*_k : k > k_{i1}$ are monotonically decreasing). As a result, $J_i = j^*_{k_1}$ and will be added to $\boldsymbol{b}_i$ after a single iteration of Bundle Build.

Bundle Build will repeat with an updated bundle that includes $j^*_{k_1}$, $\boldsymbol{b}_i \to \boldsymbol{b}_i^{1:n_r} \oplus_{end} j^*_{k_1}$. At this point, by the same logic, $j^*_{k_{i2}} \dots j^*_{k_{ir}}$ will each incrementally be added to i's bundle, resulting in a complete rebuilding $\boldsymbol{b}_i(t) = \boldsymbol{b}_i^{(n)}$. □

Theorem 3.5.1. *If all tasks are known initially (static task allocation), CBBA with partial replanning will converge in $O(n_t D)$ to the same solution as CBBA with Full Reset*

Proof. Claim 2 showed that at the time of reset, each agent immediately rebuilds their SGA bundle. As such, all parts of Lemma 1 still hold true and so too Lemma 2, thus CBBA with partial replanning will also converge in $O(n_t D)$ to the SGA solution. □

3.5.2 Partial Resetting During Dynamic Task Allocation

Now the effect of adding a new task T^* after the team has already converged on a conflict-free solution for the initial n_t tasks is considered. Note that in the worst case, the team can do a full reset and converge in $(n_t + 1)D$ rounds of communication.

Claim 3. *Assume CBBA has converged and a new task T^* appears, if the team resets the last n_r tasks allocated in SGA: $j^*_{n_t}, j^*_{n_t-1} \dots j^*_{n_t-n_r+1}$ at the beginning of Bundle Build, then CBBA converges in $(n_r + 1)D$ rounds of CBBA bidding.*

Proof. At the point of reset (t_{reset}), each agent $i \in \mathcal{I}$ has bundles that correspond to $\boldsymbol{b}_i = \boldsymbol{b}_i^{(n)}$ where $n = n_t - n_r$. First, it will be shown that $\boldsymbol{b}_i(t)^{1:L^{(n)}} = \boldsymbol{b}_i^{(n)}$ for all subsequent rounds of CBBA $t > t_{reset}$, i.e., the first n SGA task remain allocated to their respective SGA winners.

A task $j^*_k \in \boldsymbol{b}_i^{(n)}$ will remain in i's bundle indefinitely if no other agent $\tilde{i}$ outbids i. Formally, for all $j^*_k \in \boldsymbol{b}_i^{(n)}$ the following will be shown to be true

$$c_{\tilde{i}j^*_k}[\boldsymbol{b}_{\tilde{i}}(t)] < y_{ij^*_k} \quad \forall \tilde{i} \in \mathcal{I} \setminus i \quad \forall t > t_{reset} \tag{3.13}$$

where $y_{ij^*_k}$ is agent i's bid on j^* at the time of reset.

First, note that because i is the agent assigned to j_k^* by SGA, (i.e., $i = i_k^*$) then $y_{ij^*} = c_{i_k^* j_k^*}$ and also all the agents in the team are in agreement on this, so their bids will also reflect the SGA score:

$$y_{\tilde{i}j_k^*} = c_{i_k^* j_k^*},\ z_{ij_k^*} = i_k^* \qquad \forall \tilde{i} \in \mathcal{I}, \forall k \leq n \tag{3.14}$$

Now consider how an arbitrary agent $\tilde{i} \neq i$ bids at the point of reset. Bundle Build begins with $\tilde{i}$ considering its scores for T^* and all remaining tasks that are not in its bundle ($\mathcal{J} \setminus \boldsymbol{b}_{\tilde{i}}(t_{reset})$):

$$J_i = \operatorname{argmax} c_{\tilde{i}j}[\boldsymbol{b}_{\tilde{i}}(t_{reset})] \cdot \mathbf{1}(c_{\tilde{i}j}[\boldsymbol{b}_{\tilde{i}}(t_{reset})] < y_{\tilde{i}j}) \qquad \forall j \in \{\mathcal{J} \setminus \boldsymbol{b}_{\tilde{i}}(t_{reset})\} \cup \{T^*\} \tag{3.15}$$

During this first round of Bundle Build, agent $\tilde{i}$ will not be able to bid on any task $j_k^* : k \leq n$ because $y_{\tilde{i}j_k^*} = c_{i_k^* j_k^*}$ by (3.14) and $c_{\tilde{i},j_k^*}[\boldsymbol{b}_{\tilde{i}}(t_{reset})] < c_{i_k^* j_k^*}$ since by definition, the SGA solution will have the highest reward.

As such, $\tilde{i}$ can append to its bundle either one of the reset tasks $j_k^* : k > n$ or T^*, yielding an *unknown* bundle $\boldsymbol{b}_{\tilde{i}} = \boldsymbol{b}_{\tilde{i}} \oplus_{end} J_{\tilde{i}}$. Being that the value function is DMG, bids on $j_k^* : k \leq n$ will only be decreased as a result of the new added task:

$$c_{\tilde{i}j_k^*}[\boldsymbol{b}_{\tilde{i},reset} \oplus_{end} J_{\tilde{i}}] < c_{\tilde{i}j_k^*}[\boldsymbol{b}_{\tilde{i},reset}] < y_{\tilde{i}j_k^*} \qquad \forall k \leq n \tag{3.16}$$

Thus, for any subsequent round of Bundle Build after the partial reset, agent $\tilde{i}$ will only consider adding the reset tasks or T^*, meaning any non-reset tasks $j_k^* : k \leq n$ will remain assigned to i_k^* for the remainder of the replan.

For the remaining bidding in CBBA, the team will thus only allocate the remaining n_r+1 tasks (the reset tasks and T^*) which will take in the worst-case $D(n_r+1)$ rounds of CBBA. □

Theorem 3.5.2. *The convergence of CBBA with Partial Reset is $O((n_t - k_{min})D)$ where $k_{min} = \min_{j \in \mathcal{J}_{reset}} j_k^*$*

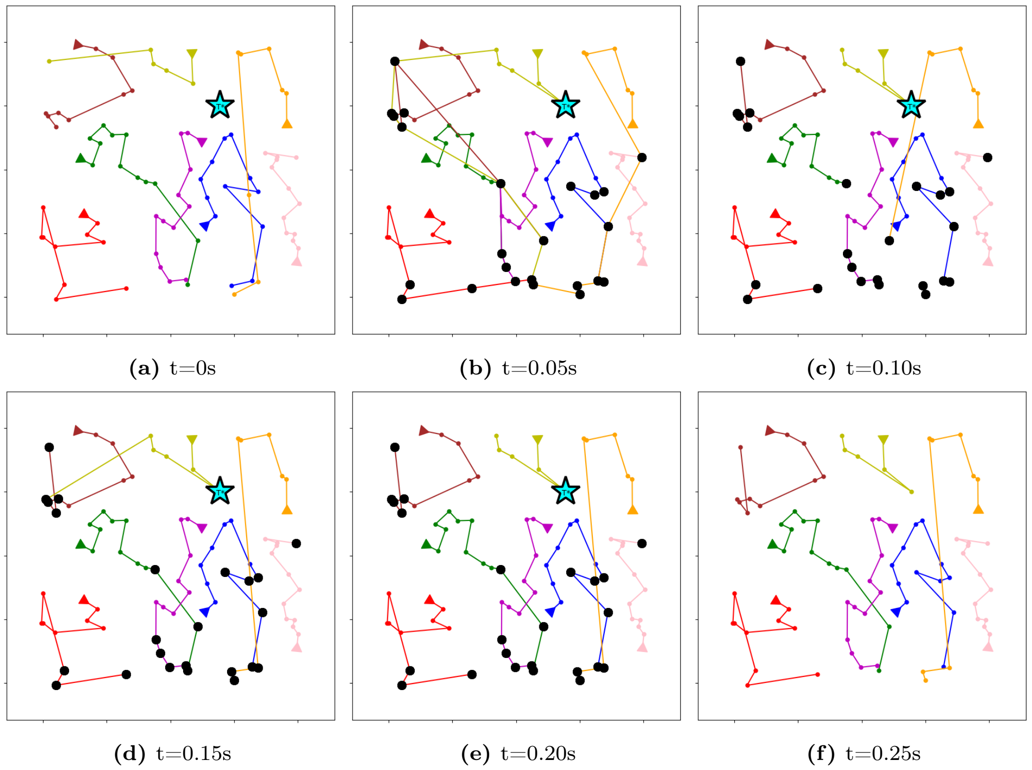
Proof. k_{min} represents the "earliest" task reset in the centralized SGA solution during the greedy auction. If the first $k_{min} - 1$ tasks are not reset, then they will remain allocated to their respective winners since they are assigned to the SGA winners, i.e., $z_{i,j}(t) = i^*$. At that point, the remaining $n_t - k_{min}$ must be allocated which will take $O((n_t - k_{min})D)$ rounds of communication. $\square$

This means that the convergence of CBBA with Partial Reset is not specifically dependent on $n_{i,reset}$ but rather which specific tasks are reset with respect to the original SGA solution. If however, the team *knows* the SGA solution, then it can specifically choose $\mathcal{J}_{reset}$ such that only the lowest k_{min} tasks are reset, thus ensuring the convergence of the partial replan. This can be done if the initial CBBA has run to completion and agreement has been reached on the first n_t solution. If that is the case, then $y_{i,j} = c^*_{i^*,j^*}$ for all the agents involved in CBBA. Likewise, in CBBA with Team Replan, each agent can identify the lowest n_{reset} tasks from the SGA solution by simply choosing the lowest n_{reset} bid tasks in y_i, ensuring that $k_{min} = n_t - n_{reset}$, ensuring a convergence of $O(n_{reset}D)$. An alternative way of viewing of the convergence of CBBA with Partial Reset is that the team is effectively resetting CBBA to some round k_{min}, locking in all allocation before that, and then allowing bidding on T^* and resuming with the centralized greedy auction. The later the new is task is allowed to be placed in the auction (i.e., the higher k_{min}, smaller n_{reset}) the shorter the convergence as the auction is nearly over. The earlier the placement of T^*, the longer the convergence as more of the centralized auction must be redone.

3.6 Results

3.6.1 Simulation

A UAV task allocation simulator was created to validate the convergence and quality of solutions for various replanning strategies. The simulator is implemented in Python and allows for varying communication conditions, dynamic robot movements, and newly appearing tasks. CBBA with Partial Replan is run locally on multiple instances of the Robot class and the Simulator only facilitates message passing between agents and the revealing of new tasks the team. A vehicle routing scenario is used with a



(a) t=0s **(b)** t=0.05s **(c)** t=0.10s **(d)** t=0.15s **(e)** t=0.20s **(f)** t=0.25s

Figure 3-6: Simulation of eight robots allocation $n_t = 80$ tasks, allocated tasks $\boldsymbol{p}_i$ are colored corresponding to the assigned robot. A new task T^* (green star) appears sequentially and tasks are released (black, filled circles) until all are allocated.

time-discounted reward function (3.2), with $n_r = 8$ agents that must visit $n_t = 80$ task locations. 100 Monte Carlo simulations are run where the initial tasks are placed in randomly located location, initialized with $R_{ij} = 1$ and $\lambda_{ij} = 0.95$ and time-discounted function (3.2). Once the team converges on an initial solution $\mathbf{p}_1 \ldots \mathbf{p}_i$, a new task T^* arrives that must be allocated by the team. This process is repeated 8 times for a total arrival of 8 new tasks. For each simulation scenario, the setting is saved so that multiple strategies can be run and compared. Figure 3-6 shows an example simulation, where initially a new task appears (top left), then tasks are reset, and a final allocation is reached (bottom right). Note that significant changes and disagreement during the replanning phase since the team is resetting a subset of previous tasks while allocating the new task.

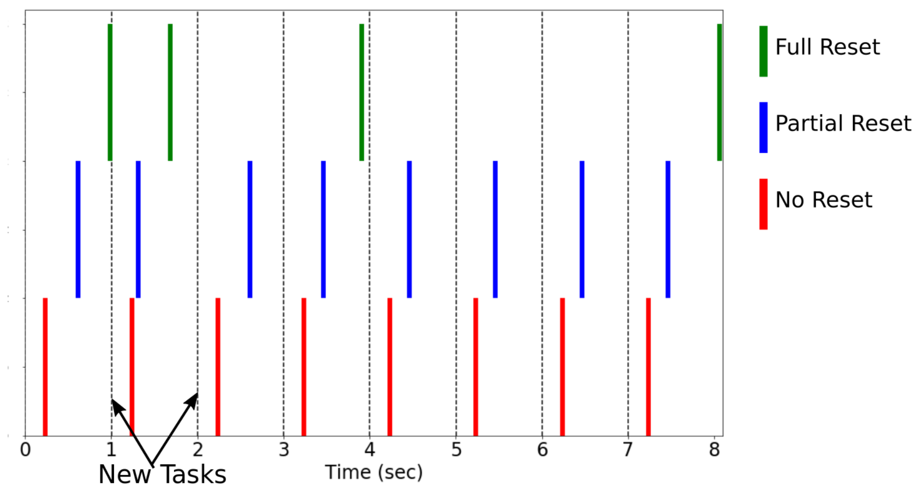


Figure 3-7: A timeline of the team's consensus on a conflict-free solution as new tasks arrive into the system (dotted line). No-reset replanning leads to a constant, quick response to the new tasks (red), while the full reset leads to periods without consensus ($t = 4 - 7$ seconds). Partial resetting (blue) provides intermediate response time that varies depending on the new task.

3.6.2 Comparing Convergence

First, simulations are run for multiple new tasks to show the effect of the various replanning strategies. Figure 3-7 shows the timeline of the team's convergence for a single simulation instance. As there were 8 new tasks, the team should reach consensus 8 times as the mission progresses. When a full reset strategy is used, convergence can not be guaranteed to occur before the next new task arrives. As such, there is a segment of time when the team is not in agreement, leading to multiple periods of new tasks before consensus. In contrast, both the partial reset and no-reset are able to provide a fast enough response to ensure convergence, and is thus not in a constant state of replanning.

Figures 3-8 and Figure 3-9 compare the number of rounds of CBBA bids required to reach consensus in the static and dynamic cases, respectively. For each simulation, the team of agents compare the four strategies outlined above: no bundle resetting, partial local bundle reset, partial team reset, and a full bundle reset. In both cases of partial resetting, the team initially resets a total of $n_{reset} = 24$ tasks, where in the partial *local bundle reset* each of the 8 agents resets 3 tasks and in the partial *team reset*, the team resets the 24 lowest valued tasks (with each individual agent resetting a variable number of task).

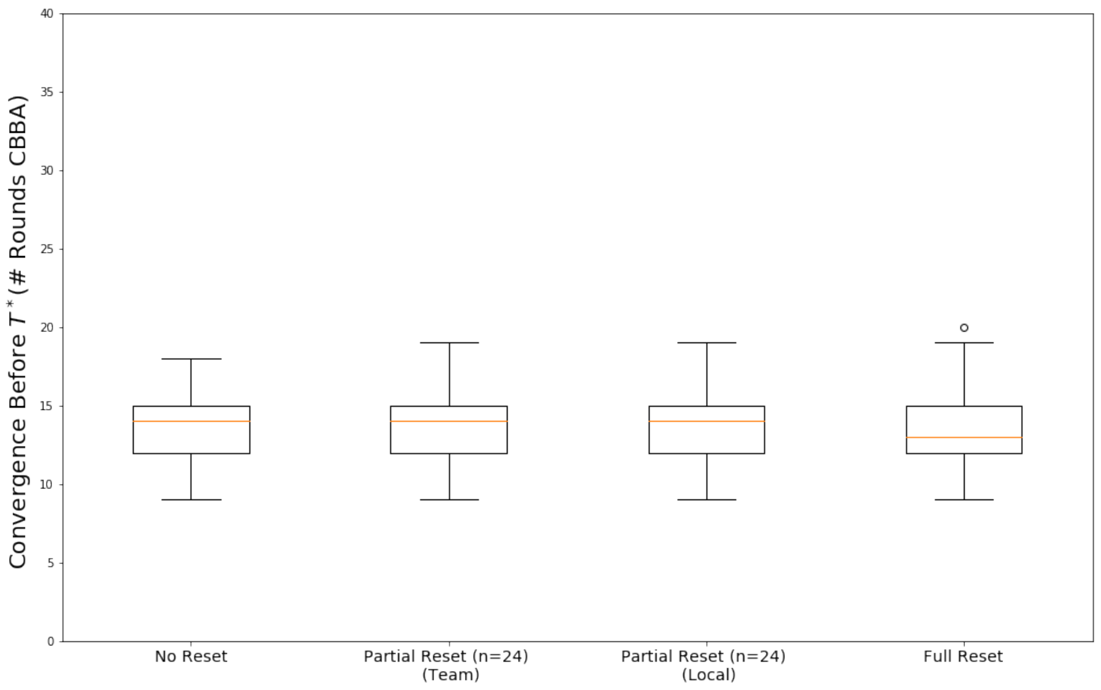


Figure 3-8: Convergence time for the initial static allocation before the new task T^* arrives. In all four replan strategies, the convergence time is the same, as expected by the theoretical convergence. In all cases, the runtime is $O(n_t D)$ regardless of the full resetting or no resetting of bundles at each round of CBBA.

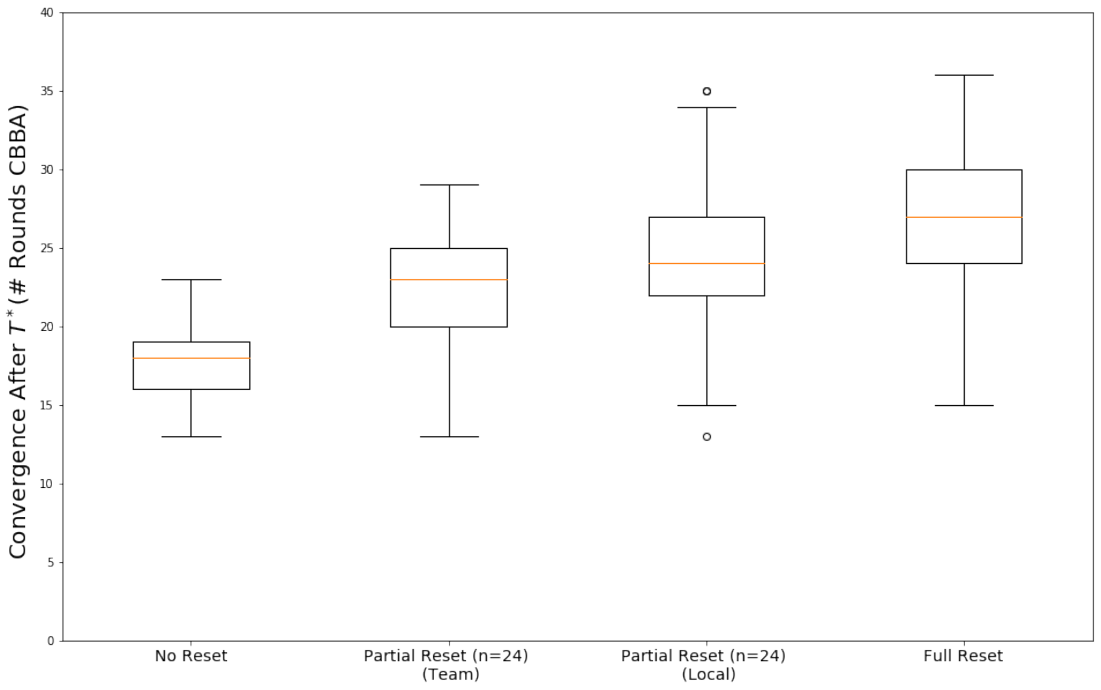
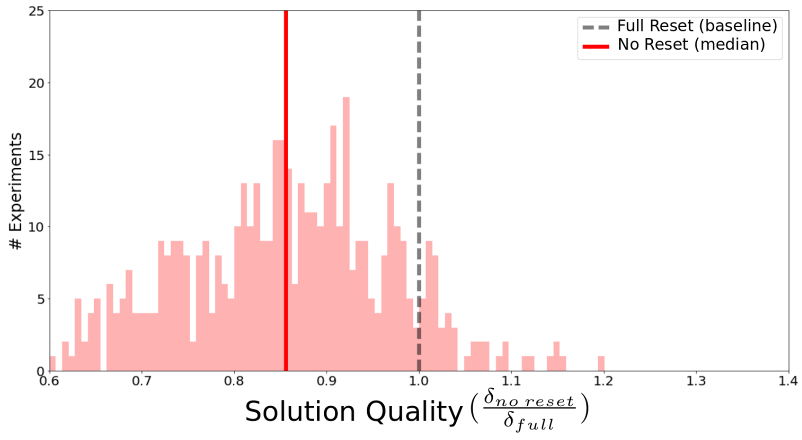


Figure 3-9: When a new tasks arrives, the number of rounds on average and worst-case is highest for a full reset replan and shortest for the no reset strategy. Choosing the lowest-n tasks to reset for a global replan converges faster than a fixed number of tasks reset in each bundle and provides intermediate performance as a whole.

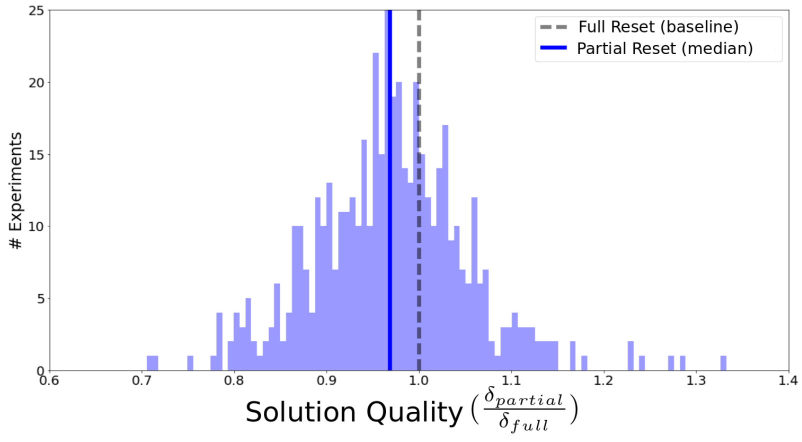
Figure 3-8 shows the number of rounds until convergence for the static case, before any new task arrive. As expected, all four strategies perform with equal convergence times. Figure 3-9 compares the convergence after a new task is introduced and must be allocated by the team. In this case, all four strategies require increased rounds of CBBA, with the no reset requiring only a minimal amount to ensure consensus on one bid and a full reset requiring the most rounds of bidding since in some cases, all tasks must be fully rebid. Between the two partial resetting strategies, local bundle reset and team reset, the local reset overall performs worse than the local bundle reset, with some simulations requiring the same amount of bidding as in the full reset case. This is expected, as only the worst case can be guaranteed to be less than a full-reset if the lowest team wide tasks are chosen for resetting. However, on average, the local bundle reset does perform faster than a full reset, suggesting that there is still a speed-up from a partial local bundle reset.

3.6.3 Comparing Solution Quality

To understand the performance gains of partial replanning, the solution quality of the resulting replan strategies are compared to the full reset strategy. While the full reset is not an optimal solution, it can serve is a baseline for "best" performance since it does have the 50% approximation of CBBA and intuitively has the highest level of coordination. The performance of each algorithm is measured by the increase on team score $\delta = \sum_{i \in \mathcal{J}} S_i(\boldsymbol{p}_i') - \sum_{i \in \mathcal{J}} S_i(\boldsymbol{p}_i')$ caused by servicing the new task, where $\boldsymbol{p}_i'$ is the solution *after* all the new tasks are allocated. Figure 3-10 shows the performance of both no reset (Fig. 3-10a) and partial reset (Fig. 3-10b) in an unconstrained setting, i.e., $L_t n_r > n_t$. As expected, the no reset and partial reset perform worst than the baseline full reset, however, the faster partial reset algorithm outperforms no resetting and generally performs more similar to a full reset. Note that the high variance in solution quality is due to the full reset still being suboptimal due to its greedy nature. However, in more constrained setting where the number of feasible solutions is fewer, partial and full reset will more consistently outperform no reset approaches.



(a) No-reset



(b) Partial reset

Figure 3-10: Performance of partial replanning compared to no replanning. (a) Shows the performance of a no-reset strategy and (b) shows the performance of a partial reset, in allocating 8 new tasks. Partial replan improves the score quality, nearing the performance of full replan baseline.

3.7 Summary

In this chapter, the CBBA with Partial Replanning algorithm was presented which is able to allocate the new task while allowing for tunable amounts of reallocation of the initial paths. CBBA-PR is shown to have a convergence that follows that of CBBA, with a convergence that is linear with the relative location of the tasks in the original CBBA allocation. In addition, if CBBA is run on the initial tasks, the final bid array can be reused to reset specific tasks to meet response time requirements of the system. Finally, simulations validated the convergence improvements of CBBA-PR compared to the full reset strategy and the solution quality improvements of CBBA-PR compared to no resetting, showing that CBBA-PR can be thought of as an effective middle-of-the-road approach compared to existing replanning algorithms. In the next chapter, the no-reset strategy will be re-considered by specifically proposing enhancements that can allow for better performance in situations when partial replanning is not a viable option.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Dynamic Allocation with Limited Resetting

The previous chapter investigated a partial replan approach that considered reallocating a subset of tasks from the original n_t tasks that were originally allocated using CBBA. One limitation of replanning with CBBA is that it is still fundamentally a greedy algorithm, producing solutions that are suboptimal. Due to this suboptimality and the inherent limitations of greedy algorithms, there will be instances where very little replanning can provide adequate solutions for the system. As such, the approach presented in this chapter is to limit new allocations to those for which the new task is only inserted into the original paths of each agent, thus not enabling reassignment of any existing tasks. The bid on the new task for each agent is still the marginal gain of the new task

$$y_{i,T^*} = \max_{n \leq |\boldsymbol{p}_i|} S(\boldsymbol{p}_i \oplus_n T^*) - S(\boldsymbol{p}_i) \tag{4.1}$$

and once the team reaches consensus on the maximum bid, the new task is allocated and the algorithm is terminated. While this approach is limiting in that it will not allow high levels of coordination, in some circumstances teams may choose to proceed with very limited replanning. In this chapter, three approaches are presented to improve the no-reset approach 1) allowing single reset for agents at capacity, 2) utilizing heuristics for triggering no task resetting, and 3) reducing network diameter

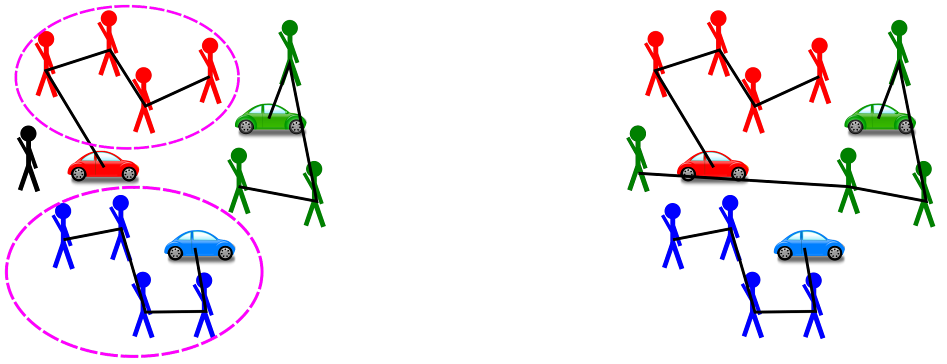


Figure 4-1: Example scenario where agents at capacity (red and blue) can not bid on the new task, leading to a pathologically poor solution, where the green agent is the only one that can be assigned the new task.

with subteam formation.

4.1 Bidding at Full Capacity

The first challenge with a no-reset strategy is that in highly constrained environment, for example when agents are at bundle capacity, $|\boldsymbol{p}_i| = L_t$, the agents can not bid on T^*, resulting in bids

$$y_{iT^*} = 0 \quad \forall i \in \mathcal{I} \quad s.t. \quad |\boldsymbol{p}_i| = L_t. \tag{4.2}$$

In cases where $L_t n_r \gg n_t$, agents will not be at capacity when a new task arrives and thus can effectively bid on the new task. However, when $L_t n_r \approx n_t$, agents will frequently fill their paths to capacity during the initial running of CBBA. In Figure 4-1, the red and blue agents are at capacity ($L_t = 4$), so when a new customer (black figure) requests a ride, the greedy algorithm prevents the red and blue agents from bidding on the new customer. In this scenario, the only remaining bidder is the green agent, resulting in a final allocation where the green agent services the new customer. If, for example, the customer is a high-value task or highly time-sensitive the final allocation will be much worse than the optimal solution (where the red or blue agent services the customer), resulting in a highly pathological result for this problem.

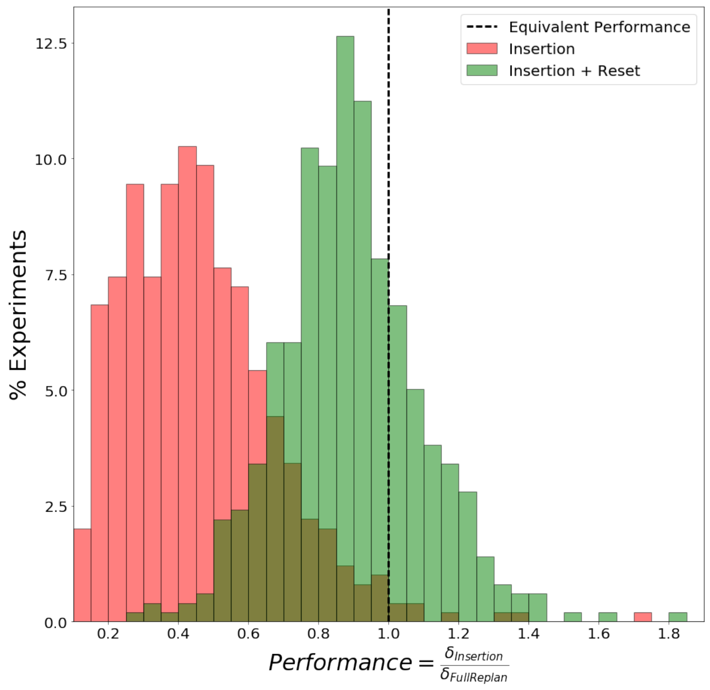


Figure 4-2: A histogram of no-reset/insertion strategy performance. compared to the full replan strategy, with single resets (green) and without any resetting (red). The single reset moves the performance closer to a performance ratio of 1, where insertion can perform as well as full replan.

4.1.1 Single Resetting at Full Capacity

The solution proposed in this thesis is to allow the agents to consider releasing a single task, $j_{i,reset}$ before bidding on T^*. In the Single Reset Auction (Alg. 5), the agents create temporary paths $\tilde{\boldsymbol{p}}_i$ by removing a single task from its path and then inserting T^*. The bid on T^*, which is equal to th marginal gain of adding T^*, must be discounted by the removal of the reset task $j_{i,reset}$

$$\tilde{y}_{i,T^*} = \max_{n \le |\boldsymbol{p}_i|} S(\boldsymbol{p}_i \oplus_n T^*) - S(\boldsymbol{p}_i) - y_{i,j_{i,reset}} \, . \tag{4.3}$$

By allowing a single-reset, the agents can better consider the gained reward by adding T^* into their own bundle. The highest bidder is then allocated T^* and their path is updated to include the new task, at which point the old task $j_{i,rest}$ is replanned by the rest of the team. The release of $j_{i,reset}$ may spark further single resetting by the team and in the worst-case, possibly result in a full replan requiring $O(n_t D)$ rounds of communication. However, as shown in the previous section, the convergence of the

Algorithm 5 Single Reset CBBA for New Task

```
1: procedure SingleResetAuction($\mathbf{p}_i, T^*$)
2:     if $|\boldsymbol{p}_i| = L_t$ then
3:         $j_{i,reset} = \arg\min_{j\in\boldsymbol{p}_i} y_{ij}$
4:         $\tilde{p}_i = \boldsymbol{p}_i \ominus j_{i,reset}$
5:         $\tilde{y}_{i,T^*} = \max_{n\le|\boldsymbol{p}_i|} S(\boldsymbol{p}_i \oplus_n T^*) - S(\boldsymbol{p}_i) - y_{i,j_{i,reset}}$
6:     else
7:         $j_{i,reset} = \varnothing$
8:         $\tilde{p}_i = \boldsymbol{p}_i$
9:         $\tilde{y}_{i,T^*} = \max_{n\le|\boldsymbol{p}_i|} S(\boldsymbol{p}_i \oplus_n T^*) - S(\boldsymbol{p}_i)$
10:    end if
11:    for $k = 1\ldots D$ do
12:        $\tilde{z}_{i,T^*}(t) = \arg\max_{m\in\mathcal{N}_i} \tilde{y}_{m,T^*}$
13:        $\tilde{y}_{i,T^*}(t) = \max_{m\in\mathcal{N}_i} \tilde{y}_{m,T^*}$
14:    end for
15:    if $z_{i,J_i}(t) = i$ then
16:        $y_{ij} = \tilde{y}_{i,T^*}$
17:        $\boldsymbol{p}_i = \tilde{\boldsymbol{p}}_i$
18:        if $j_{i,reset} \neq \varnothing$ then
19:            SingleResetAuction($\boldsymbol{p}_i$,$j_{i,reset}$)
20:        end if
21:    end if
22: end procedure
```

algorithm will be directly related to the SGA round corresponding to the released task, thus choosing the lowest bid task in a bundle will minimize the subsequent rounds of replanning

$$j_{i,reset} = \arg\min_{j\in\boldsymbol{p}_i} y_{ij}. \tag{4.4}$$

In addition, to further prevent further replanning after a task is reset, teams can either use a bid-warped bidding strategy [59] or explicitly limit the single resets to bidding on T^* and not allowing it for subsequent reset tasks ($j_{i,reset}$).

Figure 4-2 shows a histogram for the no-reset strategy without any resetting allowed (pink) and the performance when a single reset is allowed. The single reset strategy moves the performance distribution of the team to the right, closer to the baseline performance of a full-reset algorithm. Note that in this scenario, the no reset strategy with resets is able to provide solutions that are comparable to the full reset strategy. In these instances, it is especially important to allow for single resets so that agents can effectively bid on the new task.

4.2 Full Reset Performance Heuristic

In some situations, the delays from replanning may outweigh the gains from replanning, especially in time-sensitive mission settings. If the team can predict the performance of replanning before triggering a full or partial reset, then the team can tailor the amount of replanning. This is especially relevant for CBBA as it relies on a greedy solution which is inherently suboptimal, so an alternative (even naive) solution such as no bundle reset algorithm may fair well. For example, in the previous results for a no reset strategy (Figure 3-10a), the team was able to arrive at solutions that had the same or even better quality as the baseline full reset algorithm. While the majority of simulations showed improvements with increased coordination from a full reset, there are some cases where no replanning is necessary. If these scenarios can be identified by predicting the score of a full replan compared to no reset, then the team could preemptively decide to forgo the full replan. More specifically, a method is sought for calculating a performance ratio

$$H = \frac{\delta_{\text{no reset}}}{\delta_{\text{full reset}}} \tag{4.5}$$

which can trigger no resetting if $H > H^*$ where H^* is some desired performance required by the system.

The main challenge in calculating this performance ratio is that $\delta_{\text{full reset}}$ can not be calculated deterministically since the problem is NP-Hard, and thus the only method for calculating the value gain of a full reset is by performing the full reset. Instead, this chapter proposes an estimate for calculating $\delta_{\text{full reset}}$ and a decentralized method for using this estimate to compute a performance heuristic H^* for triggering a simple, no reset strategy.

4.2.1 T-Only Heuristic for Estimating Full Reset

The following heuristic is proposed to estimate the solution quality of the full reset

$$\hat{\delta}_{fullreset}(T^*) = \max_i S_i(T^*) \tag{4.6}$$

which calculates a best-case scenario in which agents are able to release all their preallocated tasks and assume that each released task is successfully serviced by a teammate. Specifically, each agent considers the following sequence of events:

1. Agent i releases all previously assigned tasks in $\mathbf{p}_i$: $\boldsymbol{p}_i \to \varnothing$
2. Remaining teammates allocate the released tasks that were previously in $\boldsymbol{p}_i$
3. Agent i only services T^*: $\boldsymbol{p_i'} = \{T^*\}$ with a new path value $S_i(T^*)$

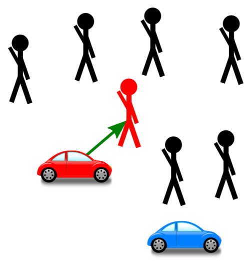
The final step, where agent i only considers servicing T^* and computes the new value becomes an estimate for that agents idealistic full replan score, estimating $\hat{\delta}_{i,\text{full reset}}$

In reality, either (1) neighboring agents will not service the tasks as well or (2) the original agent will need to retain and delay preallocated tasks, so $\hat{H}(T^*)$ will be an over-estimate on the performance of any individual agent. However, it will still provide a heuristic for the current team's performance and in scenarios with highly-capable neighboring agents, the T-only heuristic will provide a good estimate for the full replan performance. In addition, if the heuristic is consistently too optimistic, the performance ratio can be chosen to try to compensate for over estimating the full replan solution.

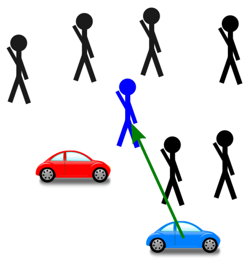
This heuristic can then be used by the entire team to calculate an estimated performance heuristic

$$\hat{H} \triangleq \frac{\max_i \delta_{i,\text{no reset}}}{\max_i S_i(T^*)} \tag{4.7}$$

which will trigger either a full reset of the existing allocations or simply assigning the new task to the highest bidder.

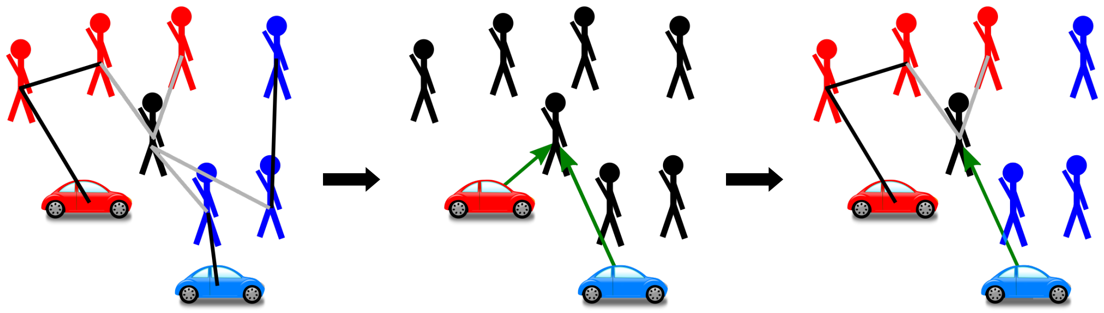


(a) T^*-only for red agent



(b) T^*-only for blue agent

Figure 4-3: Each agent calculates T-only heuristic for the gains of a full reset by "dropping" all their tasks and only bidding on the new task



(a) $\delta_{i,\text{no reset}}$ **(b)** $S_i(T^*)$ **(c)** Max-Consensus

Figure 4-4: Decentralize procedure for estimating the performance of a full replan

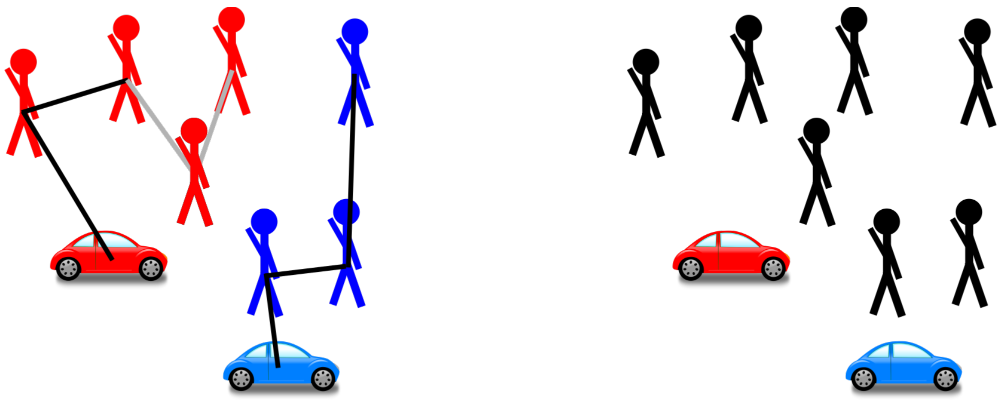

(a) $\hat{H} > H^*$: No-reset (b) $\hat{H} < H^*$: Full-reset

Figure 4-5: Team either proceeds without full resetting if $\hat{H} > H^*$ and allocates the new task to the highest no-reset bidder (a) or proceeds with further replanning (b) by conducting a full or partial reset of their existing paths (b)

4.2.2 Decentralized Procedure

For this algorithm to succeed in a wide-range of application and be relevant to decentralized solvers such as CBBA, the T-Only heuristic must also be decentrally computed the agents. This is possible by first running one round of max-consensus to reach consensus on the maximum scores for the no-reset approach and the optimistic heuristic. The full decentralized procedure is then

1. Locally compute no-reset: $\delta_{i,\text{no reset}}$
2. Locally compute T^*-only score: $\delta_{i,\text{full reset}} = S_i(T^*)$
3. Max-Consensus: $\max \delta_{i,\text{no reset}}$, $\max S_i(T^*)$
4. Performance Ratio: $\hat{H} = \frac{\max \delta_{i,\text{no reset}}}{\max S_i(T^*)}$
5. Locally trigger no-reset strategy if $\hat{H} > H^*$

Notice that all agents *locally* compute relevant score predictions and then only requires one round of consensus to achieve agreement on the level of replanning needed.

4.2.3 Results

Figure 4-6 shows a histogram of 3,000 simulations of the dynamic task allocation problem with a single new task T^* that must be allocated by the team. The blue

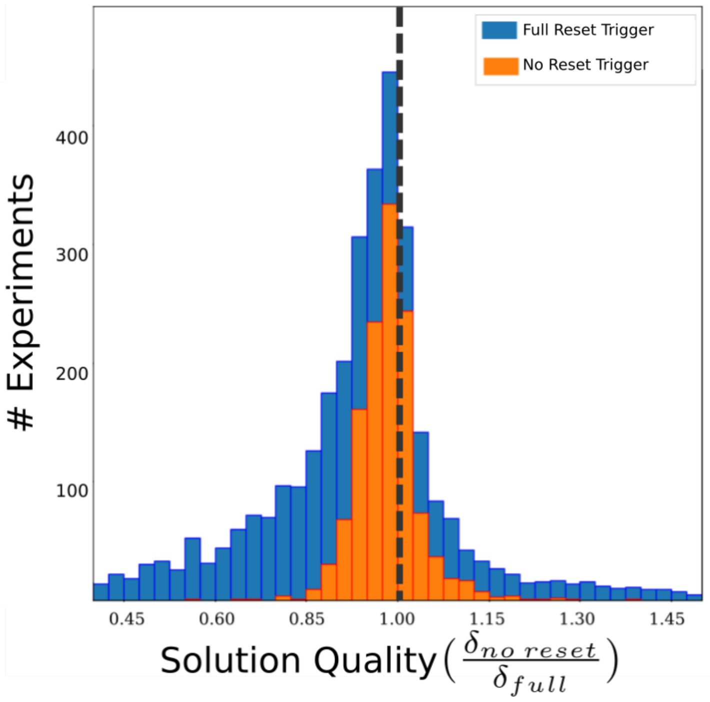


Figure 4-6: Solution quality of no reset strategy compared to full reset strategy for random placement of tasks and robots. Simulations with heuristic triggered for no-reset (orange, $\hat{H} \geq 0.75$) are centered about 0.95 performance with narrower distribution of solution quality. Simulations triggered for full-reset (blue) are skewed to the left, meaning most were simulations where no-reset would have performed poorly.

histogram reports the solution quality of the no-reset solution compared to the full replan. Note that there is a wide distribution of performance for the no-reset algorithm ranging from poor results due to limited replanning (50% solution quality compared to full replan) to performing 50% better than a full replan. The latter occurs in scenarios where greedy algorithms perform poorly, such as when the new task is very far from the original tasks, incurring a large penalty for greedily allocating the new task. The orange histogram reports the simulations classified by the heuristic, during which the team would decide *not* to execute a full replan. Note that in these results are centered at approximately 95% in solution quality compared to a full reset and results in a much smaller variance in performance for those triggered by the heuristic. The blue areas denote simulations that triggered a full reset by the heuristic. Note that this region is asymmetric, with the majority of cases triggered for replan being those were no-reset perform very poorly. This suggests that the decentralized heuristic mostly triggers replanning in scenarios where no-reset performs poorly. In addition, the overall variance in solution quality in situations where no-reset was triggered is reduced. This suggests that the heuristic leads to higher confidence in identifying settings where replanning is unnecessary, leading to faster and more efficient replanning decisions.

4.3 Task Discovery and Subteam Formation

As robot teams increase in membership size, so too its ability to operate in larger environments and service a larger number of tasks. However, with an increase in robots and coverage size comes an increase in network diameter. Any algorithm, such as CBBA, that relies on global consistency of information thus is linearly affected by an increase in agents within a team. In addition, as team size increases, the communication throughput must decrease to compensate for the limited bandwidth of communication for the team. As such, care must be taken to limit the effective size of the team for consensus, reducing the effective network diameter for running CBBA.

This section proposes a subteam formation algorithm to reduce the network diameter of the team, which consists of three primary phases: (1) discovery, (2) bidding and consensus, and (3) execution and replan. In phase 1, a discovery agent i_d discovers the new task $T^\star$ and must communicate both the existence of the task and important algorithm specific information to its teammates. In addition, it provides an initial bid that reflects its own ability to accomplish the task, as a benchmark that the rest of the team must beat, else the discovery agent will default to servicing the task. These important parameters (subteam size and consensus deadline) are calculated by the discovery agent and communicated to the rest of the team. The second phase, bidding and consensus phase, consists of a period of time during which teammates performs CBBA-like bidding and consensus to agree on an optimal agent to service the time-sensitive task, and finally, the winning agent begins executing the new task, while the rest of the team can replan the remaining tasks to optimize the allocation given the new $T^\star$ assignment.

4.3.1 Task Discovery

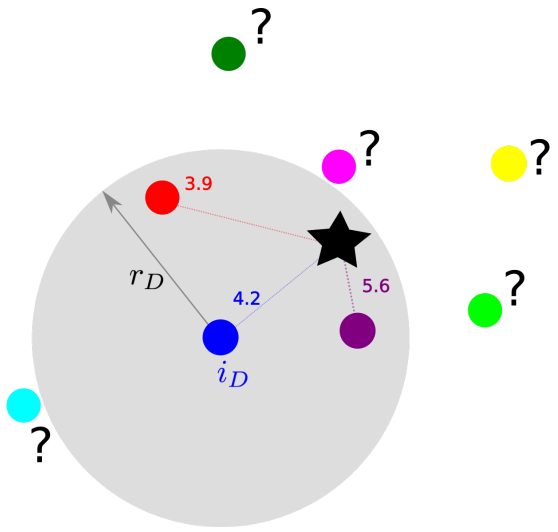


Figure 4-7: A discovery agent is the first agent to observe the new task and must decide which agents can participate in CBBA by choosing an effective network diameter

Unlike previous formulations, this section assumes that the new task is *not known* to the team, allowing for discretion by the discovery agent to choose which agents are allowed to bid on the new task (Figure 4-7). Specifically, missions where the team can take longer to replan, the discovery agent may decide to allow the entire team to participate in CBBA. If the new task is highly time-sensitive and mission critical, it may instead decide to only allow few agents to participate, to more quickly allocate the new task.

In the first phase, the discovery phase, the discovery agent (Figure 4-8) discovers the new task $T^\star$ which has a time-deadline $t^\star$. The discovery agent's goal in this stage is to calculate an effective subteam size that will be communicated to the team and guide a subteam formation. It does this by first calculating the minimum time it would need to service the task which represents the default behavior of i_d servicing $T^\star$. Any excess time, the difference between the deadline and the service time, will be utilized to coordinate with its teammates and will be denoted t_c, or consensus time. Now with a total time allocated for consensus, the agent calculates the size of the subteam that will ensure consensus within t_c. Given Δ_{comm}, the time-delay at each hop of communication (i.e., pairwise delay), and r_{id}, the communication radius of our

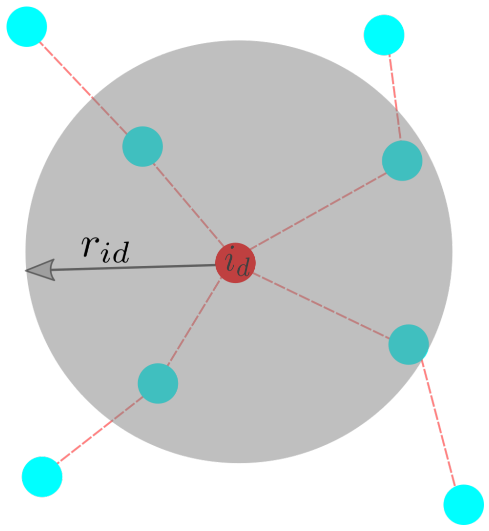


Figure 4-8: Discovery agent calculates a radius of consensus within which replanning is required for T^*

subteam, the team will need a total of $t = 3r_{id}\Delta_{comm}$ time to achieve consensus. Using this constraint, the discovery agent can calculate the maximum team radius centered from i_d. Those teammates within the radius will then participate in the subsequent phases of the algorithm. These important parameters (subteam size and consensus deadline) are calculated by the discovery agent and communicated to the rest of the team.

4.3.2 Self-Censoring

In the subsequent two stages, the agents that are within the subteam radius r_{id} participate in the bidding and consensus process, while agents beyond the radius will self-censor and not participate in the bidding. This (1) ensures that there is consensus across the sub-team, thus a conflict-free allocation; and (2) speeds up the communication exchange because less agents are contending for the medium. Once the bidding and consensus ends, the winning agent will "lock-in" the tasks in its bundle and begin executing its bundle which now includes $T^{\star}$. Now that the time sensitive task has been allocated, the remaining teammates and tasks can be allocated in the more optimal and slower process of CBBA. This is done by resetting the allocations of the remaining agents and doing a complete restart of CBBA.

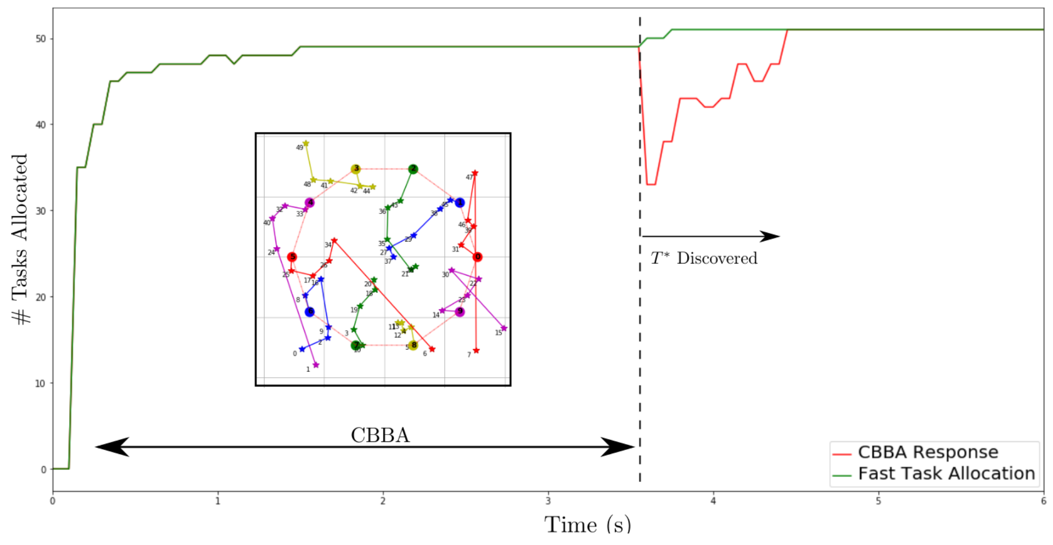


Figure 4-9: Convergence of full replan and fast CBBA after initial allocation

4.3.3 Results

To validate the algorithm, a network of 10 agents are placed in a ring formation such that the network diameter is 5 hops. Initially, the robots are presented with a large set of tasks and utilize CBBA to allocate the tasks across the team (Figure 4-9, left of the vertical dashed line). After all the tasks are allocated, a new time-sensitive task $T^{\star}$ is introduced into the system(right of dashed line). The red-line plots the number of tasks allocated (and agreed) by the team as a function of time using the static CBBA and the green line plot the response of the new, time-sensitive CBBA algorithm. With the original CBBA, there is a longer delay until convergence to a conflict-free allocation because it is doing a full reset, while the subteam formation allows for a much faster response, fulfilling the time requirement of $T^{\star}$.

4.4 Summary

In some scenarios, multi-agent teams may decide that quick assignment of a new task to an existing agent is the preferred method of dynamic allocation. This chapter presented three methods for improving such a strategy. First, a single reset bidding approach was presented that can prevent pathological solutions in the case of highly constrained scenarios. Second, a decentralized heuristic was shown to provide more reliable performance when used to trigger the no reset strategy. Finally, a subteam formation strategy was used for scenarios when a discovery agent must execute a new task before a specified deadline. While the subteam formation strategy helps to reduce the network diameter, and thus reduce the hops of communication, it may have very little effect on the reliability and speed of the channel itself. In the following chapter, the reliability and speed of the channel are explored in the context of multi-agent planning and the multi-agent effects on real communication hardware.

Chapter 5

Effects of Broadcasting Over Ad-hoc Networks on Team Consensus

Communication plays a key role in multi-agent planning and estimation, serving as a critical component of the robot system. Many multi-robot coordination schemes require some sort of consensus across the team such as in task allocation [17, 41, 60], consensus filtering [31, 61, 62], and coverage control [63]. In all of these planning algorithms, robots are able to estimate and plan in a distributed fashion with distributed communication. Communication is especially important in highly dynamic environments, when information can be changing very quickly and unpredictably, and thus team-wide communication is necessary to maintain a consistent team wide belief. In these dynamic scenarios it is imperative to have fast and reliable communication that still maintains the decentralized nature of most of these algorithms.

In order to implement distributed consensus algorithms on real robots, researchers utilize a few main modes of communication. A popular choice for hardware experiments is to utilize the 802.15.4 ZigBee protocol using the popular Digi XBee modules [64–66]. ZigBee provides mesh management and thus commonly used in multi-agent research, however, its limitations include small packet sizes and the necessity for central coordinators – and as such – is not fully decentralized. In addition, Time Division Medium Access (TDMA) is commonly used to coordinate communication and avoid contention [63]. However, TDMA has its own limitations including initial synchronization and inefficiency for dynamic teams where robots may enter or exit. Thus for static networks and low-bandwidth message passing such as small sensor values, ZigBee may be appropriate, however for higher bandwidth requirements, such as sending over full maps of the environment or entire belief spaces, other technologies are needed.

A common protocol for high-bandwidth communication is 802.11 or WiFi due to the commercial accessibility of the technology. While 802.11 provides a fully decentralized protocol that can be used in decentralized robotic applications, frequently the centralized, infrastructure mode of 802.11 is used in hardware implementations, with a multi-hop network imposed on top of the hardware. By doing so, researchers are able to simulate the network topology effects of decentralized communication on real hardware [17, 37]. However, the main limitation of this approach is that ad-hoc 802.11 performance is known to perform poorly compared to its centralized counterpart, especially for large multi-robot applications [67, 68].

The goal of this chapter is to explore the real-world reliability of decentralized communication, using 802.11's ad-hoc mode, as well test methods for increased communication reliability.

5.1 Contention and Effects on Delay

An additional motivation for studying ad-hoc communications is that many algorithms such as CBBA-PR and explicit subteam formation rely on a known network communication delay, Δ_{comm}. However, the delay through a network is rarely only dependent on the communication hardware itself. Rather, it will depend on external noises (such as multipath and path-fading) as well as the communication congestion created by the team itself, as they are contending a shared communication medium. This form of communication degradation, known as contention (Figure 5-1), is especially interesting from a multi-agent perspective because the agents themselves are causing the degradation of communication and the delays in the system. For example, if agents decide to communicate more often, it will degrade the network because

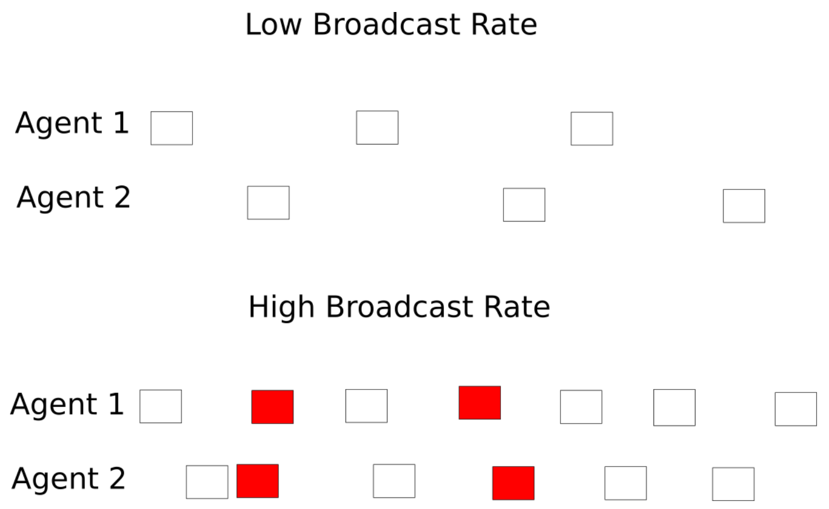


Figure 5-1: High rates of communication increases collisions in messages

there is a higher probability the messages with "collide" as agents attempt to receive messages.

While many of the external factors that lead to unreliable communication have been studied in length in both the communication and robotics community [34, 69–71], the effect of lossy communication due to contention on multi-agent planning has not been as throughly studied. While real experimentation on ad-hoc WiFi has been done on a few nodes as in [67], a larger network-wide analysis is lacking to understand the effect on the team as a whole.

5.2 802.11 Distributed Coordination Function

The 802.11 protocol [72] specifies two layers of the communication architecture: the physical layer (PHY) which designates the frequency and speed requirements of devices, and the medium access layer (MAC) which describes the policies for multiple devices communicating on a single channel. The MAC layer both describes the distributed coordination function (DCF) called Carrier Sense Medium Access which allows some coordination between users for sharing the medium and an optional personal coordination function (PCF) for centralized/infrastructure operation, where central access points receive the messages from each agents, coordinating the team communication for higher reliability and throughput.

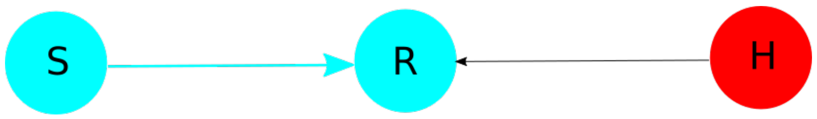


Figure 5-2: Hidden node that is unknown to sender may collide at receiving end

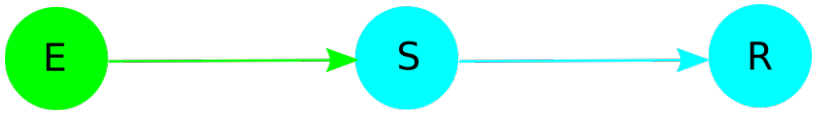


Figure 5-3: Node known to sender may delay or slow message sent even if exposed node has no effect on the receiving node

In Carrier Sense Medium Access (CSMA), a radio must first sense the medium for any existing communication traffic by measuring the electromagnetic power in the environment. If the channel is free, the radio is free to transmit its message. If the radio must wait (due to an existing broadcast), once the channel is free the radio waits an addition (random) amount of time to mitigate the possibility of multiple agents broadcasting at the simultaneously once the channel is available. An exponential back-off is used to increase the random wait time depending on the number of radios are utilizing the channel.

Because there is no explicit coordination of each radio, a single radio could flood the network with messages and prevent teammates from communicating. Even without adversarial flooding, if agents communicate at a fixed rate, as the team grows to larger numbers of agents, the delays due to the backoff function will increase delays. Conversely, the team as a whole could flood the network at broadcast rates below the physical rate limitations of the radio due to the number of agents attempting to communicate at the same time. Furthermore, CSMA does not mitigate all possible collisions in a multi-agent team. In what is commonly known as the hidden node problem (Figure 5-2), the sending node may not sense all the neighbors of the receiving node, and as a result, collisions may occur on the receiving end due to a hidden node's transmission. While hand-shaking protocols can be implemented, including the RTS/CTS handshaking option in 802.11's DCF, in a settings with highly dynamic topologies, RTS/CTS handshaking can prove to more burdensome than the hidden node problem itself. A complementary problem is the exposed node problem

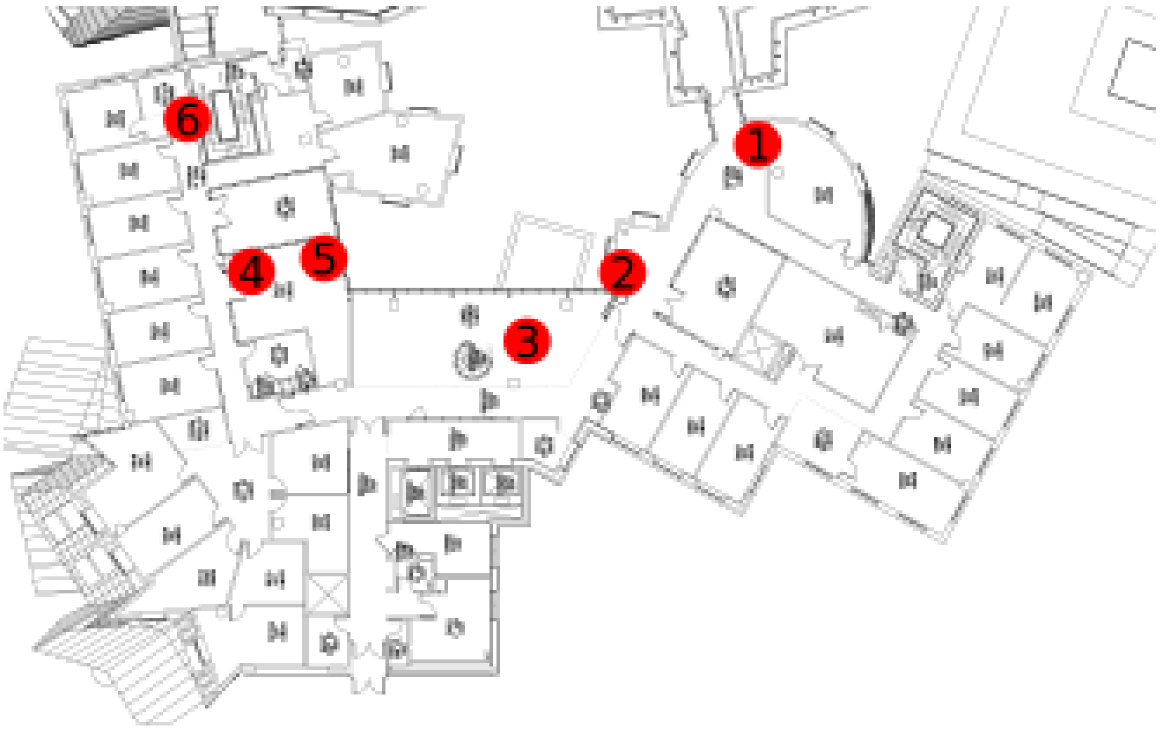


Figure 5-4: Raspberry Pi Placement in 6th Floor Building 32 at MIT

(Figure 5-3), where a neighbor of the sending node blocks the channel even though the receiving node is free to receive messages, adding delays to the communication network that are unnecessary.

5.3 Decentralized Broadcast Experiments

5.3.1 Measuring Link Level Performance

To investigate the real world effects of team wide communication on consensus algorithms, a leader election algorithm is tested on a network of Raspberry Pi nodes in the 6th floor of MIT's Stata building. Using Raspberry Pi 3's with state-of-the-art WiFi, a network of 6 agents was built that spanned the 6th floor of LIDS, featuring a wide multi-hop footprint, external disturbances (from random users) and physical obstacles such as walls and cabinets. Figure 5-4 shows the layout of the six Raspberry Pi 3's achieving a network diameter of three hops ($D = 3$) and each node has no more than 4 neighbors. Each Raspberry Pi is equipped with an onboard 802.11n Broadcom chip that is used to monitor the experiments while an external Canakit 802.11n WiFi adapter is used for the ad-hoc mesh communication.

First, the packet success rate (PSR) is measured for each node in the network to obtain a graph of the network, where PSR is defined as

$$p_{ij} = \frac{n_{i,received}}{n_{j,broadcasted}} \tag{5.1}$$

where $n_{i,received}$ are the total number of messages received and $n_{j,broadcasted}$ are the total number of messages sent by its neighbor j. The PSR is effectively the per message success rate for each link in the network. Each experiment begins by first by testing each individual link between the nodes to ensure that the links can obtain some baseline performance which is effectively $p_{i,j} \approx 1.0$. This ensures that any reduction in link quality is not due to the physical ability for the radios to communicate due to distance but rather a degradation due to congestion or the hidden node problem. A central computer collects a record of received messages from each node and computes the PSR for each node. The graphs of the network for two different broadcast rates are shown in Figure 5-5, where real-world conditions shows that links are asymmetric and varies by broadcast rate. One reason for the asymmetry in link PSR is that the in-degree will effect the ability for a node to hear from its neighbors. Thus two neighbors with varying in-degrees will lead to an asymmetric data link between the two nodes.

To further identify the effects of congestion, the PSR measurements are repeated for a simulated ad-hoc network, where the agents are physically within range but the network topology is enforced in software, and a coordinated experiment where the nodes coordinate which agent communicates. Figure 5-6 shows the median degradation over various broadcast speeds for the three types of network. The degradation of each link is calculated by comparing the PSR for each link to the baseline PSR of a centralized network at 1Hz communication rate. Note that the coordinated networks perform the best as they have no congestion or hidden node issues. The simulated network provides comparable performance to the real network, suggesting that most

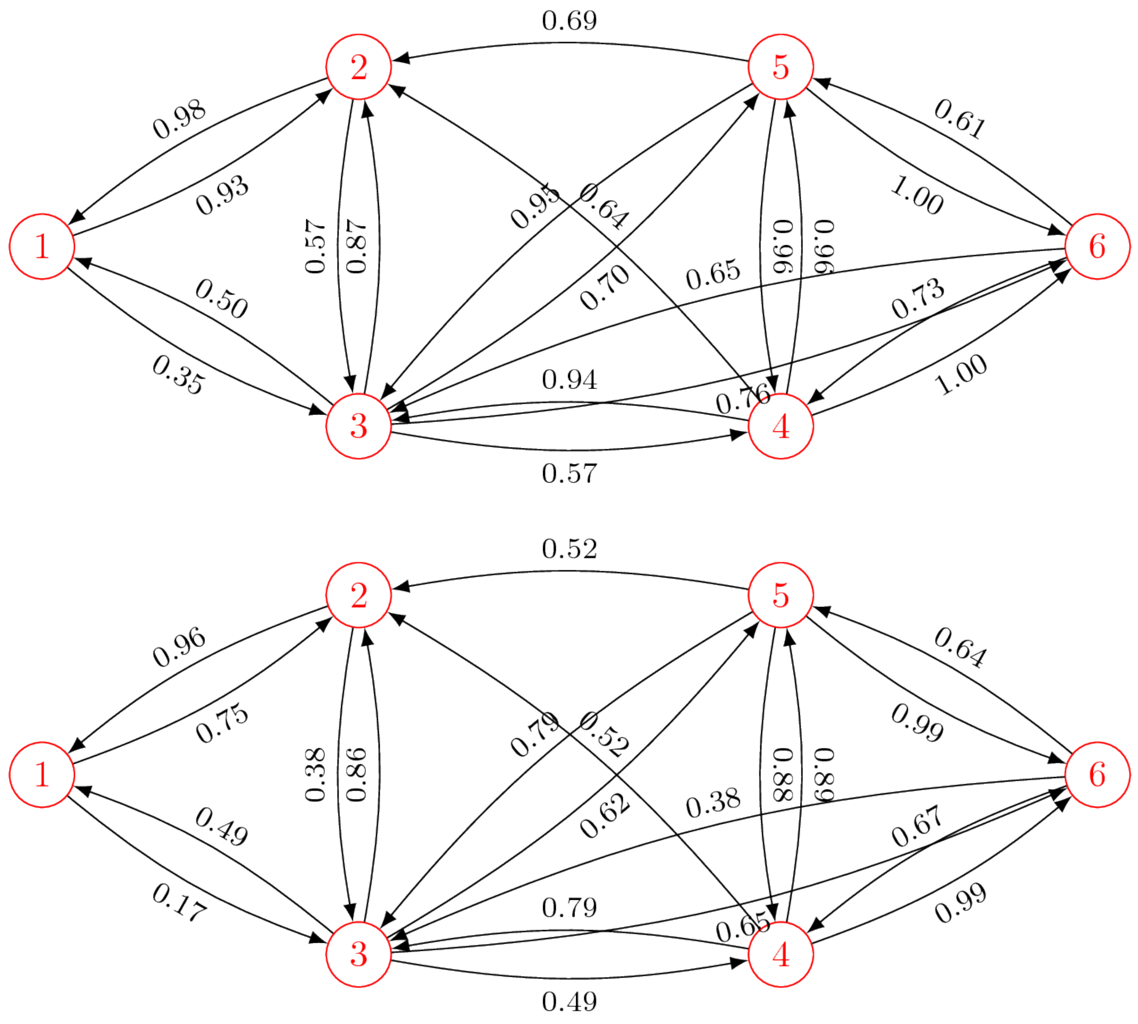


Figure 5-5: Network Topology for 1 Hz and 10 Hz broadcast rates. Edges show probability of message's successful reception (PSR).

of the network degradation is due to congestion effects and not the hidden node effect, since there are no hidden nodes in the simulated networks (as the node are all within close proximity of one another).

5.3.2 Rebroadcasting Messages for Increased Reliability

In a consensus problem, the team as a whole is not concerned with any individual link performance (PSR) but rather the speed and reliability of the network across the diameter. As such, the following experiments explore the network's ability to pass information across the entire diameter of the network. In this case, a source node

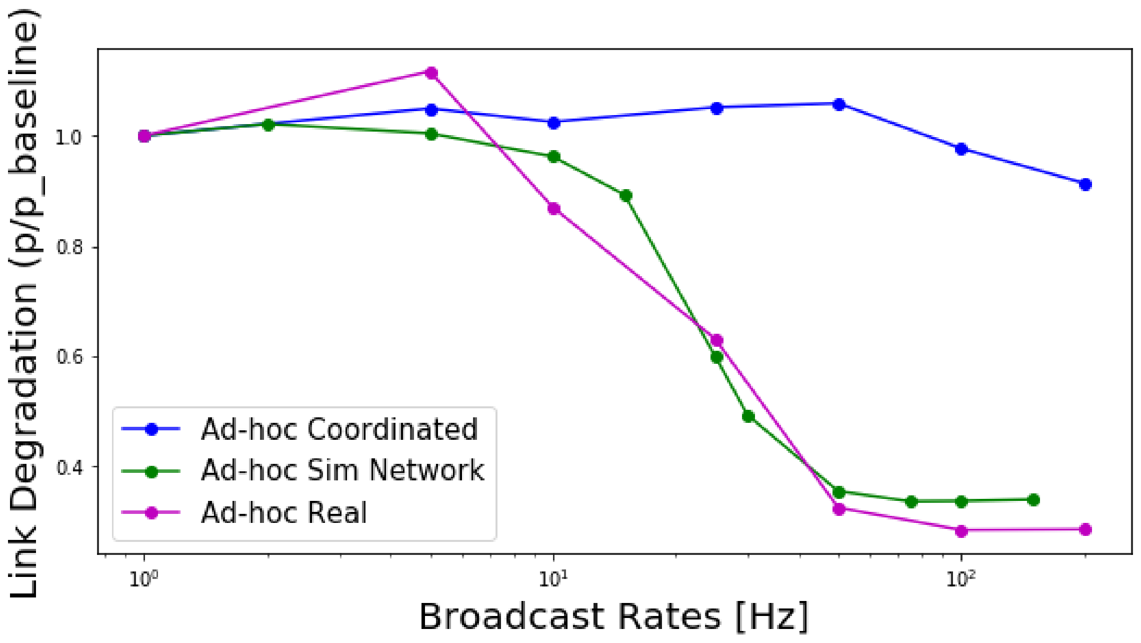


Figure 5-6: Average degradation of links as ratio of link transmission probability on baseline transmission probability (at 1 Hz broadcast rate)

(Node 1) passes information across the network to a sink node (Node 6). Since this is a purely distributed network, agents are not a priori aware of the source and sink nodes, and must run identical communication schemes.

Specifically, the reliability of the network is defined as the probability that the sink node (Node 6) receives the belief or information from the source (Node 1)

$$P = \frac{N_{6,received}}{N_{1,sent}}, \tag{5.2}$$

where now the team is no longer tracking the packets sent but rather the information itself, where $N_{6,received}$ is the total number of different beliefs that Node 6 receives and $N_{1,sent}$ is the total number of beliefs that are sent by Node 1. This distinction is important since one method for increasing reliability and throughput may be to send repeat beliefs in case there are links with a low PSR. At any given external information rate, team-wide communication strategies may have the freedom to rebroadcast information to ensure higher probability of reception from its neighbors. At lower information rates (e.g $\leq$ 1 Hz), agents may be able to resend messages 10-100 times

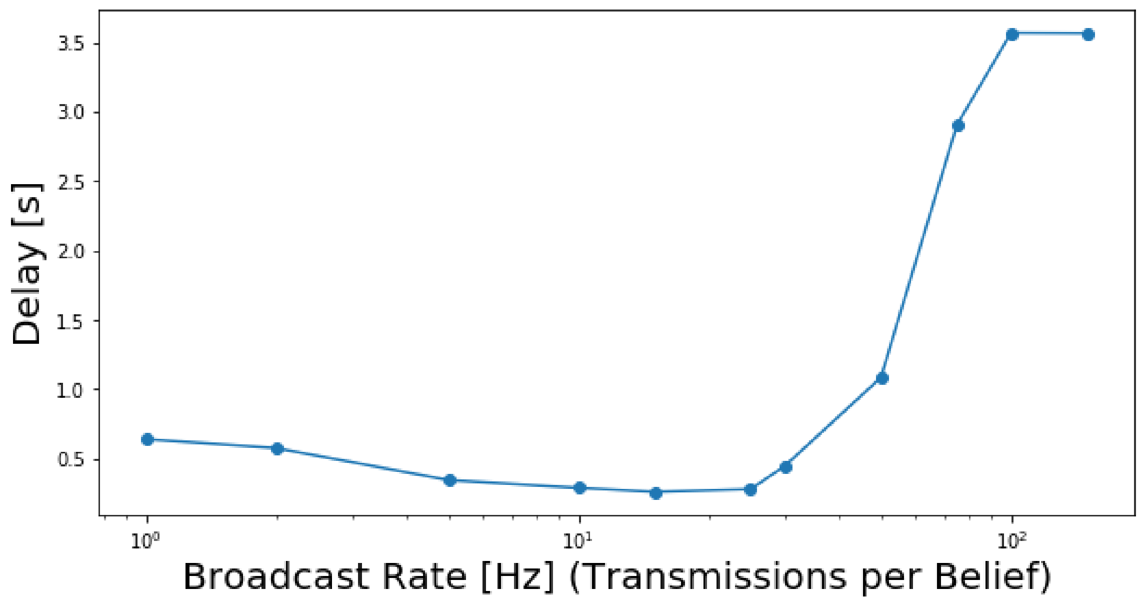


Figure 5-7: At higher rates, messages are re-sent more frequently, and thus delays decrease (500 ms). However, as network degrades, multiple retransmissions are needed until belief is received, increasing the overall delay for a belief to arrive across the network.

without reaching physical limitations of data transmission on the WiFi chip. The system designer must decide the trade-off however between increased reliability due to rebroadcasting and network degradation due to flooding. In addition, with unknown and dynamic networks, it will not be clear a priori the effect of rebroadcasts on the actual network. If, for example, the pair-wise communication quality is fixed across the links with PSR p then the probability of success of a belief repeated n times will be

$$P = 1 - (1 - p)^n \tag{5.3}$$

However, in reality p will be a function of the rate at which each agent is communicating, or in the case of a fixed rate at which beliefs are sent, then the PSR will be directly a function of n.

Figures 5-7 and 5-8 explore the teams ability to transmit information across the team. In this case, Agent 1 is attempting to communicate across the network (to Agent 6) external information that is changing at some update rate r_u. If Agent 1 is broadcasting at a rate higher than the information's update rate, Agent 1 can rebroadcast the information in subsequent transmissions by some rebroadcasting rate

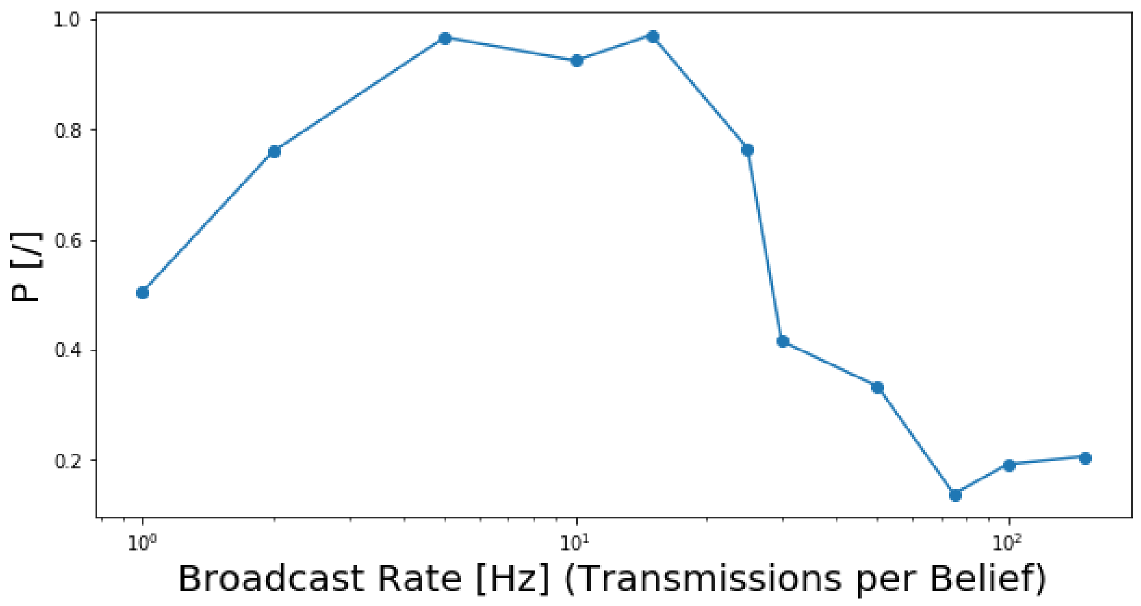


Figure 5-8: Probability that Node 6 receives belief increases as Pis rebroadcasts information, achieving near 100% success rate at n=10. At n>10, network degrades too much and overall belief success rate decreases to a low of 20%

n. These rebroadcasts could be advantageous in increasing the probability that any one of the messages is received by Agent 6. However, the increase in broadcasts may also destroy the network, adversarial affecting the teams ability to communicate, especially when every agent maintains that same broadcast speed. Figure 5-8 shows the team initially improving its reliability by rebroadcasting messages, however at $n = 10$, the network links degrade and outweigh any gains from rebroadcasting, leading to large losses in reliability. In Figure 5-7, the delays across the initially decrease as the communicate rate increases (as rebroadcasting requires higher communication rates), however at higher communication rates, the delays increase drastically due to the network degradation.

5.4 Summary

While team-wide communication is a necessary component to effective multi-agent coordination and planning, unrealistic modeling of the network itself may prove detrimental to algorithm design. In experiments, the ad-hoc network proves to be unreliable at high broadcast speeds due to contention. Increased rebroadcasting of messages provide extra reliability that important information is received the entire teams, however, network performance is not independent of broadcast speeds and must be considered when designing real distributed robot systems and task allocation algorithms such as CBBA.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Hardware and Simulation Experiments

To validate the algorithms presented in this thesis, both communication and planning algorithms were implemented on a combination of Python simulations and Raspberry Pi WiFi experiments. The Python simulations allowed multiple simulated scenarios consisting of large number of agents and tasks while allowing for motion and randomness in task values. In addition, simulations allow for a controlled environment for testing decentralized replanning algorithms by allowing synchronous and ideal communication considerations. The Raspberry Pi experiments provided the Python simulations with more realistic computation and communication conditions such as realistic communication delays and congestion-induced packet loss. In addition, the Raspberry Pi network was used to examine the effects of communication rates on the network reliability when using ad-hoc communication networks. For both the simulator and hardware experiments, code was written to enable execution on both a single central computer for simulations as well as separate robots with real hardware. To do so, wrappers were written in ROS to allow for easy execution on Raspberry Pi's running ROS nodes. Likewise, the simulator reuses classes used in the hardware for the task allocation planner, to allow for one code base for both simulations and hardware. The rest of this chapter describes the simulator and hardware architecture used in both of these experiment platforms.

6.1 Dynamic Task Simulator

The simulator (Figure 6-1) provides three main modes of simulation: (1) multiple access communication, (2) task allocation, and (3) the tasks and robots themselves. By implementing these three main class, each one can be independently tested and each class can have increased functionality and levels of complexity.

6.1.1 Multiple Access Communication

In most robot settings, the inter-robot communication is modeled as an erasure channel with probability p and a disk model, where messages can only be received within a range r. While these can accurately model physical layers limitations of communications, it does not effectively model the contention effects of multiple agents communicating on the same communication medium. Likewise, it can not model the losses due to the hidden node problem or delays due to exposed node problem. In addition, specific intricacies caused by 802.11 distributed coordination function (DCF) a.k.a WiFi, a communication medium that is accessible to most roboticists, is missing in a simple disk-model simulation.

To accurately simulate real-world communicate effects, each agent communicates via a simulated WiFi (802.11) communication module which implements the Distributed Coordination Function's: Carrier Sense Medium Access (CSMA). Before a robot attempts to communicate, it first senses the medium to determine if other agents are communicating, and if so, uses the back-off procedure of 802.11. To best match real-world conditions, the timing and back-off parameters are set to match normal operating conditions for WiFi hardware [67]. In addition, if two neighbors of a common receiving agent communicate at the same time, that receiving agents will not received either sent messages. Both the carrier sensing and collision erasure is managed by the WiFi Simulator which monitors the shared medium. At each iteration of the simulation, the Simulator checks each outgoing queue of messages from the agents to check for collisions. If there is an overlap in messaging, those messages are not sent to the receiving agent. However, if no collision occurs, the Simulator

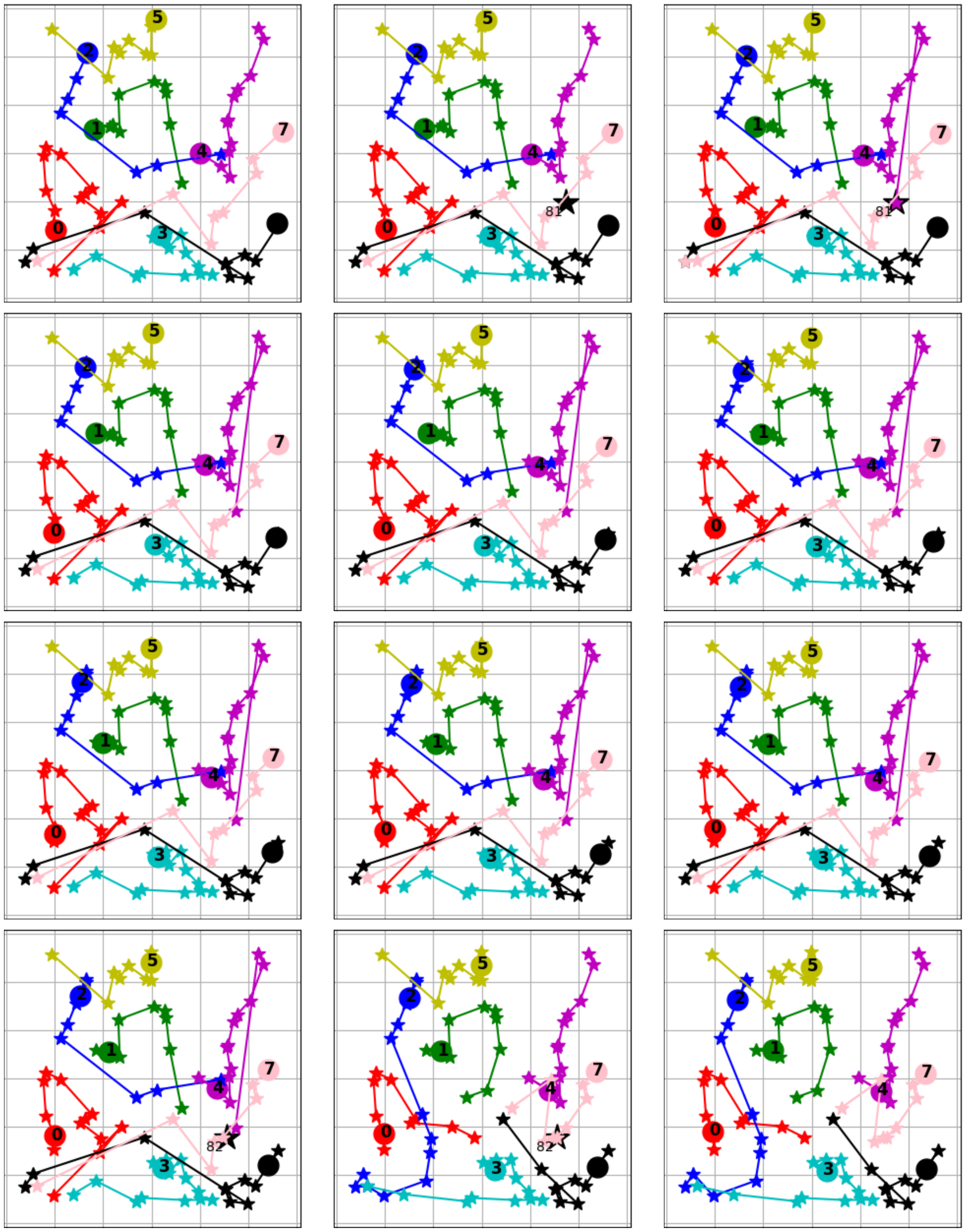


Figure 6-1: A new task (large star) appears to the agents after each agent has already allocated the initial tasks. When the agents converge on a solution, the new task is added to their allocation. Agents must lock-in the next task before starting to travel to the next task

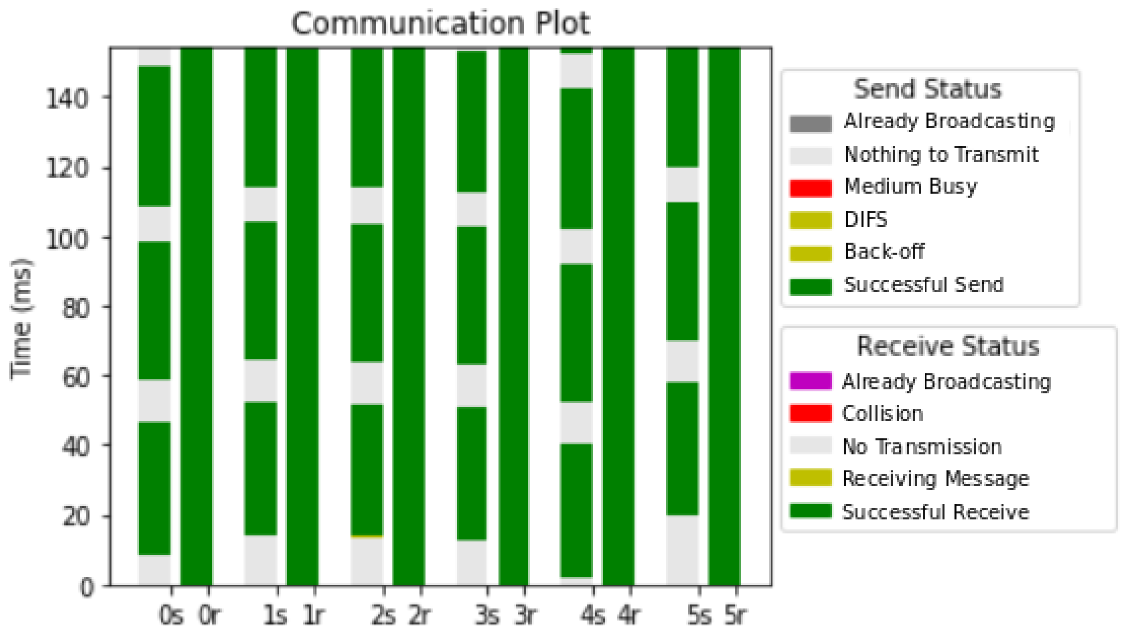


Figure 6-2: The communication status plot for the WiFi modules for six agents. Each bar represents the send and receive threads of the WiFi module. In the ideal communications shown above, messages are all received by the neighboring agents in a ring formation.

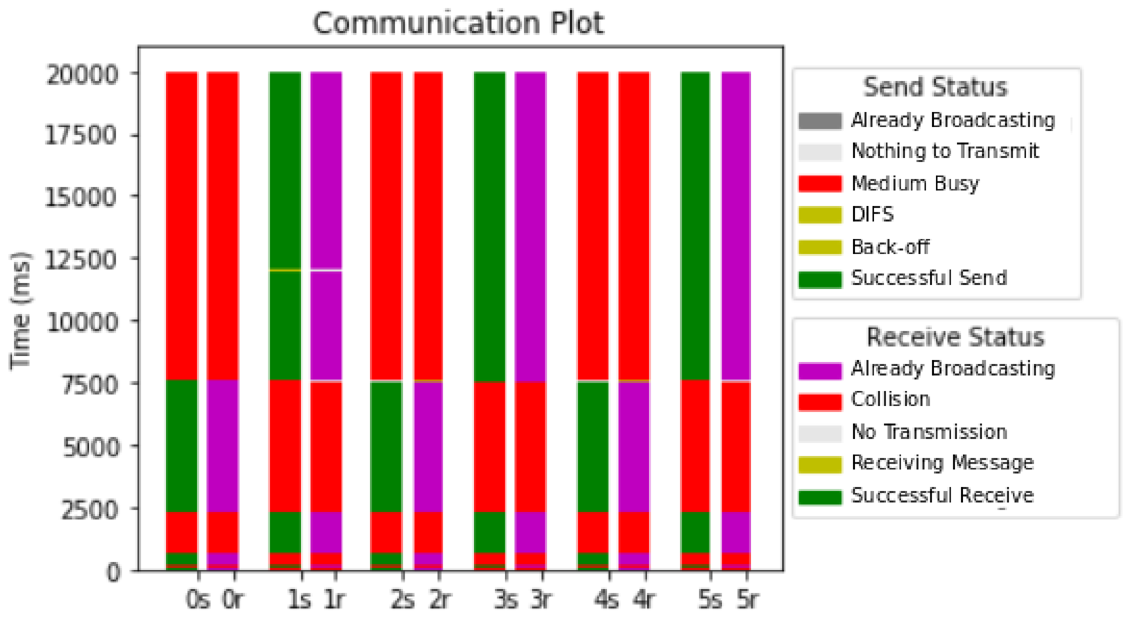


Figure 6-3: In a simulated environment with contention monitored, messages are commonly dropped by neighboring agents due to the hidden node problem or duplex issues, when agents can not receive messages while broadcasting.

passes the outgoing message to the intended recipient. Because the robot classes only have Send() and Receive() methods, the simulated communication handling can be easily replaced with real 802.11 hardware interfaces, allowing the planning code to be used in simulation and hardware experiments.

Figures 6-2 and 6-3 shows the status of each simulated WiFi module for a ring of six agents in ideal communication conditions, without contention and hidden-node issues, and in simulated real condition where contention is considered. In the ideal simulations, there is a high success rate of messages being sent and received as collisions are allowed. However, once the Simulator begins monitoring collisions and agents are restricted to only sending or receiving (half duplex communication), collisions begin to regularly occur.

The time scales of communication and planning are quite different and thus the time resolution of the overall simulation needs to change depending on the events occurring in the system. Initially, the simulator ran using a small time increment, however, simulations took hours to finish task allocation. A useful observation for implementing the simulator was that while communication conflict-resolution required short time scales to track message passing, the actual duration of the communication event was very short. An adaptive time resolution was implemented that would be triggered by the WiFi simulator to provide higher resolution time increment when a message was sent or received, but for any planning occurring between messages sent, the time scale could be lengthened to speed up the simulation time.

6.1.2 Task Allocation

Each Robot class contains a CBBAManager object which implements the original CBBA algorithms as well as the partial replan, subteam formation, and no reset algorithms. All information arrays are stored locally and after each round of consensus are sent to WiFi for inter-agent communication. The CBBAManager was implemented locally to ensure that no central information was being used in allocating the tasks. In addition, the code could easily be used on the Raspberry Pi's without any major modifications of cold.

When a new task arrives, the agents call a TaskDiscovery() subroutine to update its own information arrays and global list of tasks (setting $\mathcal{J} \rightarrow \mathcal{J}'$), and set initial replanning parameters such as number of tasks to reset or number of agents included in the replanning for CBBA-PR and subteam formation algorithms. In general, a replan parameter must be set before the onset of the planning that determines the amount of replanning required. For example, in a fixed bundle reset strategy, the team instantiates a *nBundleReset* parameter to determine the number of tasks that are removed (or kept) after each round of Bundle Build. In addition, in the team-wide replan, a τ_{replan} is set to determine the allowable amount of replanning to meet a response time requirement.

At each round of task allocation the sequence of calculations are: the agent checks for any incoming messages from neighbors, executes the conflict resolution procedure for each message, runs Bundle Build to arrive at a new allocation, and then broadcasts the allocation to the rest of the team. Since the algorithms presented in this paper are variations of the synchronous CBBA, each agent waits a fixed listening time to make sure that all agents have sent and received their allocations. This built in waiting is a cost of synchronous task allocation as each agent must ensure that the information has successfully been transmitted and received. For many of the task allocation simulations, perfect communication is assumed (no contention or edge loss), however, Chapter 5 explores the effects of imperfect communication due to congestion and multi-hop communication. To determine team-wide consensus, the agents wait for $2D$ rounds of unchanging allocations to terminate the algorithm. This $2D$ wait is to ensure that there is no delay of any disagreeing agents. However, if the agents all agree after $2D$ rounds of communication, the they will remain in agreement (unless a new task arrives) and the team can conclude that there is global agreement on a conflict-free allocation.

6.1.3 Dynamic Tasks and Robots

The reward function being optimized by the team is a time-sensitive reward function where each task has an intrinsic reward value R_j and time-discounted reward λ_j. Task reward and time-sensitivity values can be randomly generated to explore variety in relative value of new tasks. For most experiments, new tasks are designed to have value that is relatively large, such that $S(T^{*}) \approx S(\boldsymbol{p}_i)$. Agents are required to visit each task in their path for the task to be completed and receive the value of the reward, moving with constant velocity.

An issue with dynamic agents is that agents may service tasks while the bidding process is in progress. To deal with this, agents communicate already serviced tasks by bidding $y_{ij} = \infty$ and updating its information arrays to reflect the new value. In addition, the bundle of the agent must be updated, so that the task is at the beginning of the bundle by setting $\boldsymbol{b}_i(t) = j \oplus \boldsymbol{b}_i$. In this way, the bundle's bids are still monotonically decreasing and the rest of the team will not bid on the serviced task.

Another issue that may occur with robots moving while bidding is churning and dead-lock, with multiple agents servicing a task before CBBA has converged to a conflict-free solution. To deal with this issue, agents are required to lock-in any task (by bidding $y_{ij} = \infty$) before beginning its trajectory to the new task. If the team has not converged yet, the agent must wait until consensus is reached, before locking-in the next task. In practice, highly dynamic agents can cause for bids to be outdated and thus the speed of the agents are chosen to be slow compared to the rate of convergence of CBBA and that of new tasks.

6.2 Raspberry Pi Experiments

Raspberry Pi 3's (Figure 6-5 and 6-4) were used to experiment with 802.11 communication for consensus missions, testing high rate communication, and then implemented CBBA using the Raspberry Pi's to test dynamic task allocation on real hardware. In implementing synchronous algorithms, such as CBBA, and algorithms that depend on communication delays, such as the subteam formation algorithm, an understanding if required as to the realistic rates of communication for multi-agent teams. The challenge with determining these rates is that the measurement of interest is the rate

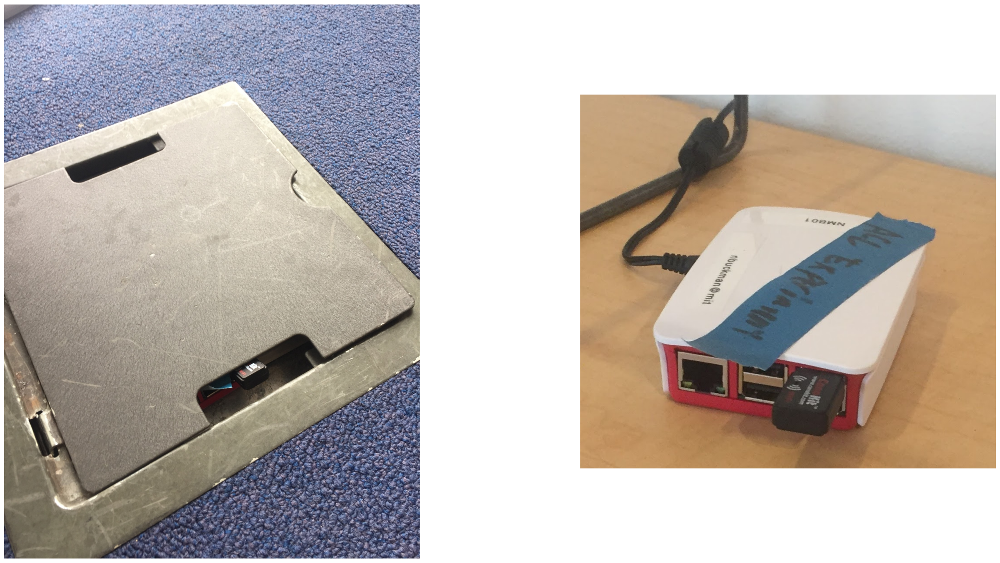


Figure 6-4: Raspberry Pis placed in Building 32 for consensus experiments

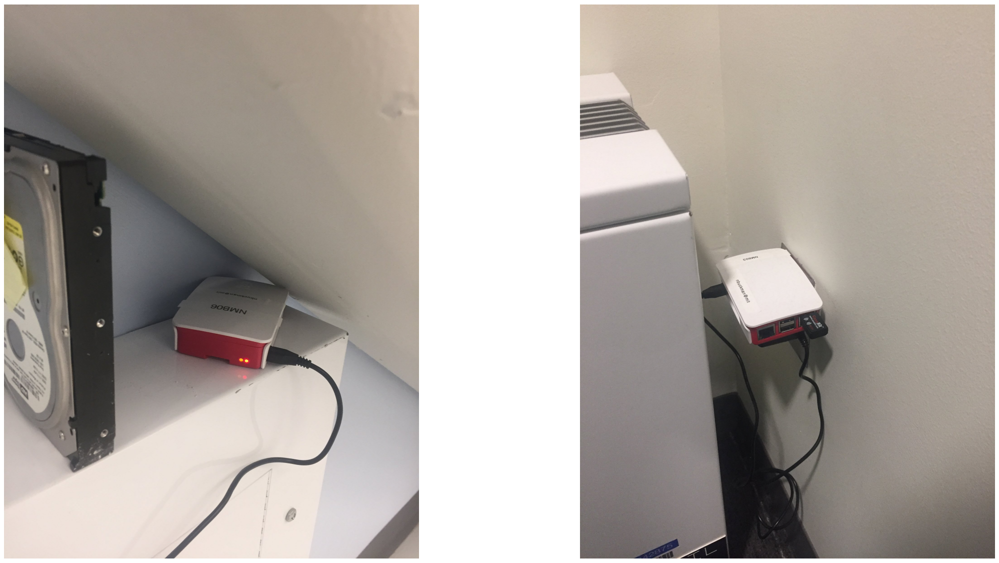


Figure 6-5: Raspberry Pis in Building 31 for CBBA experiment tests

at which information is reliably received by teammates. For example, if a radio can send out pulses at 1 Gbps but only 1% of those messages are received it is hard to judge whether that is a “faster” radio than one that communicates at 10Mbps with 100% reliability. Since consensus algorithms rely on information being received accurately by all agents, the quantity of interest for these algorithms is the rate at which agents can reliably receive information. Another difficulty is that the rates advertised on most commercial radios, such as WiFi radios, only advertise their physical communication rate, i.e., the rate at which they can communicate to another radio in ideal conditions. However, when radios are separated to further distances, obstacles placed in its way, and multiple agents sharing the channel, the actual rate of reliable communication greatly degrade. In addition, many of the rates advertised are rated for infrastructure mode, where a central router is assumed to collect the messages from each radio and then route the messages to the target recipient. However, ad-hoc operating conditions are rarely rated and usually use much slower rates. The goal of the Raspberry Pi experiments were to accurately measure the actual reliability of ad-hoc networks using WiFi and then the effects on consensus algorithms such as leader election/max-consensus and then subsequently task allocation algorithms that rely on consensus, such as CBBA.

6.2.1 Consensus Experiments

In the consensus experiments, eight Raspberry Pi nodes were placed along MIT Building 32's 6th floor in a formation such multiple hops of communication would be required to pass a message from one end of the team to the other. In these experiments, a leader election algorithm was used to track a dynamic state of a leader node. In this case, the leader state would increase its agent ID at a rate r such that:

$$\frac{dx_L}{dt} = r \tag{6.1}$$

the goal of the rest of the team was to track the ID of the leader by a simple max-consensus where:

$$x_i(t) = \max_{j \in N_i} x_j(t) \tag{6.2}$$

The speed of communication propagation through the network is measured by its ability to effectively track the changing state of the leader node.

Each Raspberry Pi runs a LeaderElection node which sends messages, receives incoming messages, and updates its own state. This is implemented using three threads to ensure that no process is slowing the other. The send and receive threads interface with the WiFi dongle using the Socket module, messaging using User Datagram Protocol (UDP). UDP was chosen for these algorithms because it does not require acknowledgment from the receiver, something that is impossible if attempting to broadcast a message without knowledge of the desired recipient, as is this case in many distributed robotics application. Instead, requiring agents to send and receive acknowledgment from each recipient increases the message delays and increases the overall message traffic. In addition, for teams with dynamic number of agents (agents entering or exiting the system), there is an advantage to having a communication system that is teammate agnostic, simply broadcasting state information without requiring knowledge of the actual teammates participating in consensus.

6.2.2 CBBA Experiments

CBBA with Partial Replan is implemented on the network of 4 Raspberry Pi's to confirm that CBBA-PR converges to a conflict free solution with a convergence speed related to the number of tasks reset. Since the simulator code is written to be reused on the Raspberry Pi's, little modification needed to be made to run on the Raspberry Pi's. One main consideration was that the agents must be synchronized, thus a waiting period is included to ensure that bids are received by each agent and conflict resolution is completed. In addition, the agents keep track of their previous allocation to count the number of rounds of CBBA that have passed without any changes. CBBA only terminates when $D+1$ rounds of CBBA occur without any changes. The number

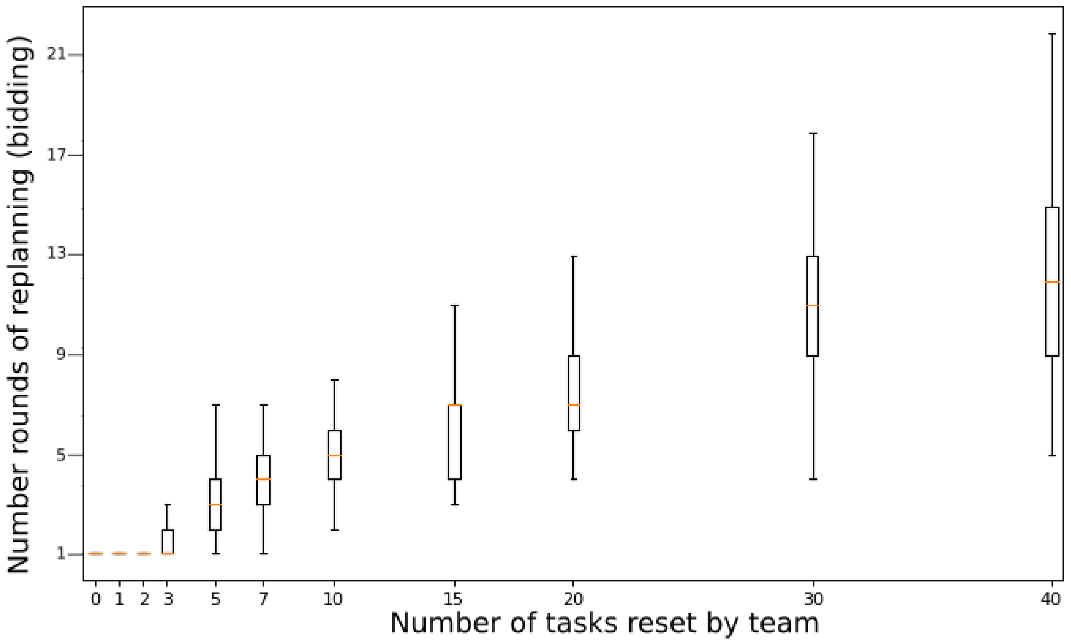


Figure 6-6: Box-and-whisker plot of various runs of CBBA-PR for a varying number of global tasks reset during the replan phase. The number of rounds bidding until the team converges on a new conflict-free allocation increased as the number of team tasks are reset.

of tasks were limited due to computation limitations for the agents, namely that the amount of time to compute all possible task allocations at a given Bundle Build lead to long auction rounds, and a lengthy CBBA convergence. In reality, faster computers can be used onboard to speed the computation of possible allocations. In addition, if tasks are known a priori, cost tables can be computed for all possible allocations, however the space requirement to store the cost matrix is exponential in number of tasks. Figure 6-6 shows the replan times for CBBA-PR in 100 task configurations. In each task configuration, CBBA-PR is run multiple times, each time the number of tasks reset at the time of task discovery is varied at each run of CBBA. As the number of tasks reset increases, so too the number of rounds required by the team to reach consensus. The results in these experiments both validate that CBBA-PR can converge to a conflict-free solution, and the that convergence is proportional to the number of tasks reset.

6.3 Summary

In this chapter, the software and hardware used throughout this thesis were described, including key implementation details such as time-resolution, medium access control, and dynamic agents for the simulator and details of the consensus module for the Raspberry Pi's. As both the Raspberry Pis and Python simulator were used interchangeably in testing the planning algorithms, it was very beneficial to have core set of code that could be tested easily in both simulation and hardware. The simulations allowed for rapid testing and large scaling of tasks and agents. While the hardware experiments, grounded the simulations in reality by providing realistic delays and constraints to the simulator. Future integration of these planning algorithms with a robot's perception, control, and motion modules will provide an even more complete testing environment for the efficacy of these replanning algorithms.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis extended the decentralized CBBA algorithm [17] to allow a team of agents to allocate new tasks that arrive after having initially allocated a set of time-sensitive tasks. Recent work has focused either on centralized dynamic routing algorithms or decentralized dynamic assignment problems where agents can independently assign tasks. This thesis specifically explored algorithms that can be applied to score functions that are combinatorial in nature. The main approach of CBBA with Partial Replan is to (1) allow variable amounts of resetting depending on the mission settings and (2) reusing the solution and bids from the original allocation problem solved by the agents. A proof of convergence was provided for the bundle resetting algorithm which then motivated a strategy of resetting the team-wide lowest bid tasks. This thesis also explored alternative strategies for allocating new tasks and investigated the impact that communication has on the team's ability to run consensus.

This work also demonstrated the algorithms and theory presented in simulation and hardware experiments to further validate the contributions of the work. Specifically, dynamic task allocation simulations were used to test replanning algorithms on a wide variety of task settings, while enabling simulations with a large number of agents and tasks. Raspberry Pi experiments confirmed the real-world usability of CBBA-PR and provided real-world communication data that reflected congestion and multi-hop conditions.

7.2 Future Work

While this work provided algorithms for dynamic tasks, additional levels of dynamics must be addressed by decentralized task allocation. For example, new agents may enter the team, allowing for drastically different allocations and team capabilities. Likewise, in hostile environments, it is common for agents to exit the mission or lose connectivity with the team. Dynamic task allocation algorithms should be able to handle these agent membership dynamics. Further dynamics that could be considered in decentralized task allocation include time-dependent task value functions, dynamic agent capabilities (such as time-varying heterogeneity), and the agent's own dynamics as it services tasks during the planning.

In addition, while communication connectivity was addressed in [**?**], the congestion issue and bandwidth-capacity is not explicitly considered in CBBA. Likewise, many decentralized algorithms assume that the main limitation of communication is the multi-hop connectivity, requiring a factor of D communications to allow information to propagate across the network. However, the degradation of networks due to congestion poses an additional constraint the reliability of consensus algorithms such as CBBA. As such, a future direction of this research includes a co-design of planning and communication algorithms, such that decentralized planning algorithms can explicitly consider the bandwidth of the channel, limiting bids and agent communication to best utilize the shared channel, and likewise, communication protocols can consider the functional use case of that channel by the team to optimize transmission of information.

Bibliography

[1] US Army. The us army robotic and autonomous systems strategy. *Internet) available at: www. arcic. army. mil/App_ Documents/RAS_ Strategy. pdf (viewed 2 November, 2017)*, 2017.

[2] Persistent aquatic living sensors (pals) proposers day. URL `https://www.darpa.mil/news-events/persistent-aquatic-living-sensors-proposers-day`.

[3] Future cities could run on shared fleets of electric self-driving cars. URL `http://fortune.com/2016/10/11/shared-electric-self-driving-cars/`.

[4] Amazon robotics screenshot. URL `https://www.amazonrobotics.com/#/vision`.

[5] Sajjad Yaghoubi, Negar Ali Akbarzadeh, Shadi Sadeghi Bazargani, Sama Sadeghi Bazargani, Marjan Bamizan, and Maryam Irani Asl. Autonomous robots for agricultural tasks and farm assignment and future trends in agro robots. *Int. J. Mech. Mechatronics Eng.*, 13(3):1–6, 2013.

[6] Todd Litman. Autonomous Vehicle Implementation Predictions: Implications for Transport Planning. *Transp. Res. Board Annu. Meet.*, (2014):36–42, 2014. doi: 10.1613/jair.301.

[7] M.R. Endsley. Autonomous Horizons: System Autonomy in the Air Force - A Path to the Future. *US Dep. Air Force, Washingt.*, 1:27, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. IEEE Int. Conf. Comput. Vis.*, volume 2015 Inter, pages 1026–1034, 2015.

[9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[10] Michael R Garey and David S Johnson. *A Guide to the Theory of NP-Completeness*. 1979. ISBN 00806234.

[11] J K Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[12] Dorit S Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., 1996.

[13] H.W. Kuhn. The hungarian method for the assigning problem. *Nav. Res. Logist. Quaterly*, pages 83–87, 1955.

[14] Luke B. Johnson, Han-lim Choi, and Jonathan P. How. The role of information assumptions in decentralized task allocation: A tutorial. *IEEE Control Syst.*, 36 (4):45–58, August 2016.

[15] J. Stipes, R. Hawthorne, David Scheidt, and D. Pacifico. Cooperative Localization and Mapping. In *2006 IEEE Int. Conf. Networking, Sens. Control*, volume 00, pages 596–601. IEEE, 2006.

[16] Defense Advanced Research Projects Agency (DARPA). DARPA-PS-17-01 Mobile Force Protection Solicitation. Technical report, 2016.

[17] Han-Lim Choi, Luc Brunet, and J.P. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Trans. Robot.*, 25(4):912–926, August 2009.

[18] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *Eur. J. Oper. Res.*, 225(1):1–11, February 2013.

[19] Ulrike Ritzinger, Jakob Puchinger, and Richard F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *Int. J. Prod. Res.*, 54(1):215–231, January 2016.

[20] Éric D. Taillard, Luca M. Gambardella, Michel Gendreau, and Jean Yves Potvin. Adaptive memory programming: A unified view of metaheuristics. *Eur. J. Oper. Res.*, 135(1):1–16, 2001.

[21] R Montemanni, L M Gambardella, A E Rizzoli, and A V Donati. Ant Colony System for a Dynamic Vehicle Routing Problem. *J. Comb. Optim.*, 10(4):327–343, December 2005.

[22] Éric Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transp. Sci.*, 31(2):170–186, May 1997.

[23] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Planned Route Optimization For Real-Time Vehicle Routing. In *Dyn. Fleet Manag.*, volume 38, pages 1–18. Springer US, Boston, MA, 2007.

[24] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. *Parallel Comput.*, 27(12):1641–1653, November 2001.

[25] Barrie M Baker and M.A. Ayechew. A genetic algorithm for the vehicle routing problem. *Comput. Oper. Res.*, 30(5):787–800, April 2003.

[26] Jano I van Hemert and J. A. La Poutré. Dynamic Routing Problems with Fruitful Regions: Models and Evolutionary Computation. In X. Yao, E. Burke, and J. Lozano et al., editors, *Parallel Probl. Solving from Nat. VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 692–701. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[27] Eugene Edison and Tal Shima. Genetic Algorithm for Cooperative UAV Task Assignment and Path Optimization. In *AIAA Guid. Navig. Control Conf. Exhib.*, number August, Reston, Virigina, August 2008. American Institute of Aeronautics and Astronautics.

[28] Xu Guangtong, Liu Li, Teng Long, Zhu Wang, and Ming Cai. Cooperative Multiple Task Assignment Considering Precedence Constraints Using Multi-Chromosome Encoded Genetic Algorithm. In *2018 AIAA Guid. Navig. Control Conf.*, number January, pages 1–9, Reston, Virginia, January 2018. American Institute of Aeronautics and Astronautics.

[29] J.A. Benediktsson and P.H. Swain. Consensus theoretic classification methods. *IEEE Trans. Syst. Man. Cybern.*, 22(4):688–704, 1992.

[30] Ali Jadbabaie, Jie Lin, and A.S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. Automat. Contr.*, 48(6):988–1001, June 2003.

[31] Brian J. Julian, Michael Angermann, Mac Schwager, and Daniela Rus. Distributed robotic sensor networks: An information-theoretic approach. *Int. J. Rob. Res.*, 31(10):1134–1154, September 2012.

[32] Ya Zhang and Yu-Ping Tian. Maximum Allowable Loss Probability for Consensus of Multi-Agent Systems Over Random Weighted Lossy Networks. *IEEE Trans. Automat. Contr.*, 57(8):2127–2132, August 2012.

[33] Stacy Patterson, Bassam Bamieh, and Amr El Abbadi. Convergence Rates of Distributed Average Consensus With Stochastic Link Failures. *IEEE Trans. Automat. Contr.*, 55(4):880–892, April 2010.

[34] Stacy Patterson, Bassam Bamieh, and Amr El Abbadi. Distributed average consensus with stochastic communication failures. *IEEE Conf. Decis. Control*, (0):4215–4220, 2007.

[35] Alireza Tahbaz-Salehi and Ali Jadbabaie. On consensus over random networks. In *Proc. 44th Annu. Allert. Conf. Commun. Control Comput.*, pages 1315–1321. University of Illinois at Urbana-Champaign, Coordinated Science Laboratory and Department of Computer and Electrical Engineering, 2006.

[36] Reza Olfati-Saber and R.M. Murray. Consensus Problems in Networks of Agents With Switching Topology and Time-Delays. *IEEE Trans. Automat. Contr.*, 49 (9):1520–1533, September 2004.

[37] Reza Olfati-Saber, J.A. Alexander Fax, and Richard M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proc. IEEE*, 95(1):215–233, January 2007.

[38] Angelia Nedic and Asuman Ozdaglar. Distributed Subgradient Methods for Multi-Agent Optimization. *IEEE Trans. Automat. Contr.*, 54(1):48–61, January 2009.

[39] Angelia Nedic, Asuman Ozdaglar, and P.A. Parrilo. Constrained Consensus and Optimization in Multi-Agent Networks. *IEEE Trans. Automat. Contr.*, 55(4): 922–938, April 2010.

[40] Alessandro Settimi and Lucia Pallottino. A subgradient based algorithm for distributed task assignment for heterogeneous mobile robots. In *52nd IEEE Conf. Decis. Control*, pages 3665–3670. IEEE, December 2013.

[41] Stefano Giordani, Marin Lujak, and Francesco Martinelli. A Distributed Algorithm for the Multi-Robot Task Allocation Problem. In *Trends Appl. Intell. Syst.*, pages 721–730. 2010.

[42] Smriti Chopra, Giuseppe Notarstefano, Matthew Rice, and Magnus Egerstedt. A Distributed Version of the Hungarian Method for Multirobot Assignment. *IEEE Trans. Robot.*, pages 1–16, 2017.

[43] D P Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Ann. Oper. Res.*, 14(1):105–123, December 1988.

[44] B.P. Gerkey and M.J. Mataric. Sold!: auction methods for multirobot coordination. *IEEE Trans. Robot. Autom.*, 18(5):758–768, October 2002.

[45] M.B. Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-Based Multirobot Coordination: A Survey and Analysis. *Proc. IEEE*, 94(7):1257–1270, July 2006.

[46] Dimitri P. Bertsekas. The Auction Algorithm for Assignment and Other Network Flow Problems: A Tutorial. *Interfaces (Providence).*, 20(4):133–149, August 1990.

[47] Ioannis Rekleitis, Ai Peng New, Edward Samuel Rankin, and Howie Choset. Efficient Boustrophedon Multi-Robot Coverage: an algorithmic approach. *Ann. Math. Artif. Intell.*, 52(2-4):109–142, April 2008.

[48] M.J. Mataric. Task-allocation and coordination of multiple robots for planetary exploration. In *Proc. 10th Int. Conf. Adv. Robot.*, pages 61–70, 2001.

[49] Bernardine Dias and Anthony Stentz. A Free Market Architecture for Distributed Control of a Multirobot System. *6th Int. Conf. Intell. Auton. Syst. IAS6*, 6:115–122, 2000.

[50] Yehuda Elmaliach, Noa Agmon, and Gal A. Kaminka. Multi-robot area patrol under frequency constraints. *Ann. Math. Artif. Intell.*, 57(3-4):293–320, December 2009.

[51] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and João Valente. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *J. F. Robot.*, 28(5):667–689, September 2011.

[52] Mehdi Alighanbari and J.P. How. Decentralized Task Assignment for Unmanned Aerial Vehicles. In *Proc. 44th IEEE Conf. Decis. Control*, pages 5668–5673. IEEE, 2005.

[53] Mehdi Alighanbari and Jonathan How. Robust Decentralized Task Assignment for Cooperative UAVs. In *AIAA Guid. Navig. Control Conf. Exhib.*, volume 5, pages 3232–3247, Reston, Virigina, August 2006. American Institute of Aeronautics and Astronautics.

[54] Luc Brunet, Han-Lim Choi, and Jonathan How. Consensus-Based Auction Approaches for Decentralized Task Assignment. In *AIAA Guid. Navig. Control Conf. Exhib.*, number August, Reston, Virigina, August 2008. American Institute of Aeronautics and Astronautics.

[55] Michael M Zavlanos, Leonid Spesivtsev, and George J Pappas. A distributed auction algorithm for the assignment problem. In *2008 47th IEEE Conf. Decis. Control*, pages 1212–1217. IEEE, 2008.

[56] Michael Otte, Michael Kuhlman, and Donald Sofge. Multi-robot task allocation with auctions in harsh communication environments. In *2017 Int. Symp. Multi-Robot Multi-Agent Syst.*, number Mrs, pages 32–39. IEEE, December 2017.

[57] Harilaos N. Psaraftis. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Transp. Sci.*, 14(2): 130–154, May 1980.

[58] Luke Johnson, Sameera Ponda, Han-Lim Choi, and Jonathan How. Asynchronous Decentralized Task Allocation for Dynamic Environments. In *Infotech@aerosp. 2011*, number March, pages 1–12, Reston, Virigina, March 2011. American Institute of Aeronautics and Astronautics.

[59] Luke Johnson, Han-Lim Choi, Sameera Ponda, and Jonathan P. How. Allowing non-submodular score functions in distributed task allocation. In *2012 IEEE 51st IEEE Conf. Decis. Control*, number 1, pages 4702–4708. IEEE, December 2012.

[60] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. Distributed algorithm design for multi-robot generalized task assignment problem. In *2013 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 4765–4771. IEEE, November 2013.

[61] Reza Olfati-Saber and J.S. Shamma. Consensus Filters for Sensor Networks and Distributed Sensor Fusion. In *Proc. 44th IEEE Conf. Decis. Control*, number 0, pages 6698–6703. IEEE, 2005.

[62] Saptarshi Bandyopadhyay and Soon-Jo Chung. Distributed estimation using Bayesian consensus filtering. In *2014 Am. Control Conf.*, pages 634–641. IEEE, June 2014.

[63] Mac Schwager, Brian J. Julian, Michael Angermann, and Daniela Rus. Eyes in the Sky: Decentralized Control for the Deployment of Robotic Camera Networks. *Proc. IEEE*, 99(9):1541–1561, September 2011.

[64] James Stephan, Jonathan Fink, Vijay Kumar, and Alejandro Ribeiro. Hybrid architecture for communication-aware multi-robot systems. In *2016 IEEE Int. Conf. Robot. Autom.*, pages 5269–5276. IEEE, May 2016.

[65] Stephanie Gil, Mac Schwager, Brian J. Julian, and Daniela Rus. Optimizing communication in air-ground robot networks using decentralized control. In *2010 IEEE Int. Conf. Robot. Autom.*, pages 1964–1971. IEEE, May 2010.

[66] Sameera Ponda, Luke Johnson, Han-Lim Choi, and Jonathan How. Ensuring Network Connectivity for Decentralized Planning in Dynamic Environments. In *Infotech@aerosp. 2011*, number March, pages 1–18, Reston, Virigina, March 2011. American Institute of Aeronautics and Astronautics.

[67] Giuseppe Anastasi, Marco Conti, and Enrico Gregori. IEEE 802.11 AD HOC Networks: Protocols, Performance, and Open Issues. In *Mob. Ad Hoc Netw.*, pages 69–116. John Wiley & Sons, Inc., Hoboken, NJ, USA, January 2005.

[68] Carlo Vallati, Victor Omwando, and Prasant Mohapatra. Experimental Work Versus Simulation in the Study of Mobile Ad Hoc Networks. In *Mob. Ad Hoc Netw.*, pages 191–238. John Wiley & Sons, Inc., Hoboken, NJ, USA, March 2013.

[69] Alireza Tahbaz-Salehi and Ali Jadbabaie. A Necessary and Sufficient Condition for Consensus Over Random Networks. *IEEE Trans. Automat. Contr.*, 53(3): 791–795, April 2008.

[70] T.C. Aysal, M.E. Yildiz, A.D. Sarwate, and Anna Scaglione. Broadcast Gossip Algorithms for Consensus. *IEEE Trans. Signal Process.*, 57(7):2748–2761, July 2009.

[71] Alex Olshevsky and John N. Tsitsiklis. Convergence Speed in Distributed Consensus and Averaging. *SIAM Rev.*, 53(4):747–772, January 2011.

[72] IEEE. *IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, volume 2012. 2012.