Distributed Multi-agent Decision-making for Task Assignment and Collision Avoidance

by

Mitchell Khoo

ORCID ID: 0000-0003-2938-5971

Submitted in partial fulfillment for the degree of Doctor of Philosophy

Department of Electrical and Electronic Engineering
The University of Melbourne

March 2022
Abstract

Multi-agent systems generally involve agents cooperating or coordinating to perform complex tasks. Typically, agents must do so by computing decisions amongst themselves without a centralised decision-maker. In this thesis, a team of agents with different initial locations is tasked with shifting their positions to a set of goal locations. There are two aspects of this type of multi-agent mission that are of interest. The first relates to deciding which agent to assign to which goal location. The second aspect relates to deciding how the agents should move to their goals without colliding with each other along the way. These two aspects can be investigated separately but they are also linked as assigning agents in particular ways can lead to intrinsic collision avoidance between agents. This thesis explores each of these ideas.

Assignment problems arise in multi-agent systems when there is a need to allocate a set of tasks to a set of agents. Two types of assignment problems are considered in this thesis. The Bottleneck Assignment Problem (BAP) is an assignment problem where the objective is to minimise the costliest allocation of a task to an agent, while the Lexicographic Bottleneck Assignment Problem (LexBAP) is a related problem with the objective of further minimising the allocation costs of the remaining agents and tasks. These two types of assignment problems are highly applicable in time-critical applications, where there is a need for agents to complete tasks such that the worst-case completion time is minimised. Hence, many centralised algorithms exist to solve them. However, it has become increasingly desirable to produce solutions without relying on a centralised decision-maker. In particular, there is a need for distributed algorithms to solve these problems that do not require a centralised decision-maker having access to all information from each agent.

Distributed algorithms for assignment problems with other objectives have been proposed, yet to date no such algorithms for either the BAP or the LexBAP exist. In order to address this gap, novel tools for analysing the BAP and LexBAP are introduced. The introduction of these tools precipitates an analysis of structure in the BAP, where in particular, an investigation into how two separate BAPs can be merged into a combined BAP is conducted. Then by applying these concepts, a distributed algorithm for the BAP is presented, a greedy distributed algorithm for the LexBAP is developed and conditions on exactness of the greedy approach are provided. Numerical results are presented for both distributed algorithms to benchmark them against existing approaches.
In specific applications where agents are mobile robots that move towards goal locations to complete tasks, it is subsequently prudent to guarantee inter-agent collision avoidance once the assignment problem is computed. Two approaches for providing collision avoidance guarantees are explored. The first approach shows collision avoidance as an intrinsic property of assigning agents according to the LexBAP, which leads to time-varying safe sets of positions for agents. The second approach considers the use of Control Barrier Functions for collision avoidance, which alternatively leads to time-varying safe sets of accelerations for mobile agents modelled as double integrators.
This page intentionally left blank.
Declaration

This is to certify that

• the thesis comprises only my original work towards the PhD except where indicated in the Preface,

• due acknowledgement has been made in the text to all other material used, and

• the thesis is fewer than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Mitchell Khoo, March 2022
This page intentionally left blank.
Preface

The work in this thesis was carried out during my PhD study at the University of Melbourne, Australia in collaboration with my supervisors, Prof. Iman Shames, Prof. Chris Manzie and Dr. Tony A. Wood. The following is a list of publications that are associated with the contents of this thesis. Where indicated, some publications are currently under review at the time of writing.

Journal papers:


For the third item on the above list, I am not listed as the first author. However, I contributed to greater than 50% of the work in terms of conception of theory, design of mathematical proofs and development of methodology.

Conference proceedings:


This research is funded by the following sources.
• Research Training Program (RTP), the University of Melbourne.

• Defence Science and Technology Group, MyIP: 7558, 7562, 9156.
Acknowledgements

I would like to thank my supervisor Professor Iman Shames. All the way through my PhD journey, he has challenged me to think critically and to back my claims with appropriate reasoning. His ideas and thoughts have been insightful and have consistently provided me with inspiration and things to ponder. He taught me the value of being precise in my explanations and meticulous in my work. Without his guidance, I would have definitely struggled to find my footing progressing through my PhD study. I want to thank him for his effort and for always providing me with his time to discuss my ideas in detail.

I would also like to thank my co-supervisor Professor Chris Manzie. His help has been crucial in developing my work from initial ideas into mature concepts. Much of my research has taken shape on the whiteboard in his office. He has also kept me on track throughout my PhD by reminding me to think about the bigger picture and to keep a focus on the milestones. On that note, I am also grateful for the support from Lyn Buchanan and her efficient help in resolving any and all matters related to administration.

I am very grateful for all the help I received from Dr. Tony Wood. He was the first person I turned to discuss new ideas, the last person helping me polish manuscript drafts and was present in every technical discussion in between. This still does not adequately sum up his support throughout my PhD and I want to express my profound gratitude for his assistance.

I will also take this opportunity to thank my parents, Khoo Koon Loong and Ng Lai Yan. Their love, nurturing and support have made me the person I am today and I am eternally grateful to them for raising me with such patience. I want to also thank my sister, Samantha Kho. We have not always agreed on everything, but I know I can always count on her to be there for me.
To my parents, my sister and the rest of my family...
This page intentionally left blank.
Contents

Nomenclature xxii

1 Introduction 1
   1.1 Motivation and Literature Review ........................................... 1
   1.2 Contributions ........................................................................ 9

2 Background 12
   2.1 Graph Theory ......................................................................... 12
      2.1.1 Undirected Graph ............................................................... 12
      2.1.2 Tree Graph ....................................................................... 13
      2.1.3 Weighted Bipartite Graph ................................................... 14
   2.2 Common Assignment Problem Formulations ................................. 15
      2.2.1 The Sum Assignment Problem ............................................. 15
      2.2.2 The Bottleneck Assignment Problem .................................... 15
      2.2.3 The Lexicographic Bottleneck Assignment Problem ................ 16
   2.3 Distributed Computation .......................................................... 16

3 Structure in The Bottleneck Assignment Problem 18
   3.1 Introduction ............................................................................. 18
   3.2 Relationship Between an Augmenting Path and an MCM ................ 18
   3.3 Structure for Solving the BAP ..................................................... 19
      3.3.1 Properties of a Critical Edge ............................................... 20
   3.4 Structure for Solving the LexBAP ............................................... 21
      3.4.1 Properties of an Edge with Positive Price of Absence ............... 23
   3.5 Clustering Agents ..................................................................... 26
      3.5.1 Investigating the Role of Augmenting Paths in the BAP .......... 28
   3.6 Conclusion .............................................................................. 31

4 A Distributable Approach for Solving the Bottleneck Assignment Problem 32
   4.1 Introduction .............................................................................. 32
   4.2 Centralised PruneBAP .............................................................. 33
      4.2.1 Demonstration of PruneBAP ............................................... 34
   4.3 Distributed PruneBAP .............................................................. 35
# Contents

4.3.1 Distributed Maximum Consensus ........................................ 36  
4.3.2 Distributed Edge Removal ............................................... 36  
4.3.3 Distributed Augmenting Path Search ................................. 37  
4.3.4 Alternative Method for Distributed Augmenting Path Search .... 41  
4.3.5 Comparing Distributed Search Methods ............................. 43  
4.4 Warm-starting Property .................................................... 44  
4.5 Numerical Analysis ......................................................... 46  
4.5.1 Complexity of PruneBAP ................................................ 47  
4.5.2 Size of Messages Exchanged Between Agents .................... 49  
4.5.3 Comparison to a Greedy Method .......................... 49  
4.6 Conclusion ................................................................. 50  

5 A Distributable Approach for Solving the Lexicographic Bottleneck Assignment Problem 52  
5.1 Introduction ............................................................. 52  
5.2 Greedy Approach to the LexBAP ........................................ 53  
5.3 Conditions for Correct Solutions from Greedy Approach .......... 56  
5.4 An Algorithm for Solving the LexBAP ................................ 59  
5.5 Case Study ............................................................... 63  
5.6 Conclusion ................................................................. 66  

6 Collision Avoidance for Carrying Out Tasks with Mobile Robots 67  
6.1 Introduction ............................................................. 67  
6.2 Collision Avoidance Guarantee via SeqBAP .......................... 68  
6.2.1 Sufficient Conditions for Collision Avoidance ................. 71  
6.2.2 Local Constraints for Guaranteed Collision Avoidance ....... 73  
6.3 Collision Avoidance Guarantee via CBF ............................ 76  
6.3.1 Existing CBF Methods .............................................. 77  
6.3.2 CBF Method to Address Feasibility and Decoupling Issues ... 82  
6.3.3 Agent and Obstacle Collision Avoidance ....................... 84  
6.4 Conclusion ................................................................. 85  

7 Conclusion and Future Works ............................................. 87  

Bibliography ................................................................. 92
This page intentionally left blank.
# List of Figures

1.1 Illustration of the asset protection scenario. The green square represents the asset, the blue circles represent the decoys and the red triangles represent the incoming threats. ................................................................. 3

2.1 Illustration of an undirected graph. The circles represent vertices and the lines represent edges. ................................................................. 12

2.2 Illustration of an alternating path. The circles represent vertices and the lines represent edges. The solid lines represent an edge in $\mathcal{M}$ and the dashed lines represent an edge not in $\mathcal{M}$ ................................................................. 13

2.3 Illustration of a tree. The circles represent vertices and the lines represent edges. The blue circle represents the root vertex. ................................................................. 14

2.4 Illustration of a weighted bipartite graph. The circles represent vertices and the lines represent edges. White vertices represent agents $\mathcal{A}$ and black nodes represent tasks $\mathcal{B}$. ................................................................. 15

3.1 Comparison of a pruned edge set and a matching-sublevel set for an example graph. The circles represent vertices and the lines represent edges. White vertices represent agents $\mathcal{A}$ and black nodes represent tasks $\mathcal{B}$. The solid lines represent an MCM $\mathcal{M}$, the edge highlighted in red is an edge with largest weight in $\mathcal{M}$. ................................................................. 22

3.2 Illustration of Proposition 3.2 and Corollary 3.3. The circles represent vertices and the lines represent edges. White vertices represent agents $\mathcal{A}$ and black nodes represent tasks $\mathcal{B}$. The solid lines represent an MCM $\mathcal{M}$, the edge highlighted in red is an edge with largest weight in $\mathcal{M}$. ................................................................. 25

3.3 Illustration of a bottleneck cluster with respect to a critical edge. The dotted lines represent edges not in the matching $\mathcal{M}$, the solid lines represent edges in $\mathcal{M}$. The set of agents is $\{a_1, a_2, ..., a_7\}$ and the set of tasks is $\{b_1, b_2, ..., b_7\}$. The edge $e_c = \{a_1, b_1\}$ is a critical edge. Only edges in $\phi(\mathcal{G}_B, \mathcal{M})$ are displayed on the figure. ................................................................. 27
3.4 Illustration of Theorem 3.2 in $\mathbb{R}^2$. The vertices are partitioned into two sets $V_1 = \{a_1, a_2, a_3\} \cup \{b_1, b_2, b_3\}$ and $V_2 = \{\alpha_1, \alpha_2\} \cup \{\beta_1, \beta_2\}$. Graph $G_1$ is a bottleneck cluster with respect to $e_1 = \{a_1, b_1\}$, and graph $G_2$ is a bottleneck cluster with respect to $e_2 = \{\alpha_2, \beta_2\}$. The length of each line corresponds to the weight of that edge. Solid lines show edges in $M_1 \cup M_2$. The set of edges with dashed lines is an MCM with smaller bottleneck weight than $W(e_1)$.

4.1 Demonstration of PruneBAP with four agents $A = \{a_1, a_2, a_3, a_4\}$ and four tasks $B = \{b_1, b_2, b_3, b_4\}$. The set of edges $E_B$ are arranged in order of ascending weight, where $e_{pq}$ is the edge between agent $a_p$ and task $b_q$. At Iteration 1, an initial arbitrary MCM is chosen, denoted by the four circled edges. Edges to the right of the dashed lines have been pruned from $E_B$ and edges to the left form the set $\phi(G_B, M)$. Note, $W(e_{44}) \geq W(e_{11}) \geq W(e_{21})$, i.e., with each iteration the weight of the largest edge in the current MCM is non-increasing. The algorithm terminates when it is not possible to form a matching of cardinality 4 from the remaining edges to the left of the dashed line.

4.2 Comparison of alternating trees constructed by AugDFS and AugBFS. The solid lines represent an MCM $\bar{M}$, the edge highlighted in red is the edge to be removed, $\bar{e} = \{\bar{a}, \bar{b}\}$, and $b_1$ is the root vertex.

4.3 Demonstration of applying a warm-start to PruneBAP to solve Problem 3.1. The sets of agents and tasks are $A_1 = \{a_1, a_2\}$, $A_2 = \{a_3, a_4\}$, $B_1 = \{b_1, b_2\}$, and $B_2 = \{b_3, b_4\}$. Then, $A_3$ and $B_3$ are the same as $A$ and $B$ in Fig. 4.1. In this example, warm-starting pruneBAP allows a bottleneck assignment of $G_3$ to be found in 1 iteration.

4.4 Sample configuration of agents and tasks. The solid lines show the bottleneck edges $e_1$ and $e_2$ for the subproblems. The black dash-dotted line shows the bottleneck edge $e_3$ for the combined problem.

4.5 Average number of iterations of the while-loop of PruneBAP for completion versus the number of tasks, $n$.

4.6 Average number of time steps for completion of PruneBAP versus the number of tasks, $n$.

4.7 Average maximum number of explored agents and overall average number of explored agents per $D$ time steps for AugBFS versus number of tasks, $n$. For AugDFS, one agent is explored per $D$ time steps.

4.8 Weight of largest edge in MCM versus number of time steps.

4.9 Average number of time steps for CBAA to converge and average number of time steps for PruneBAP to find an MCM with largest weight lower than the largest weight from the CBAA assignment versus the number of tasks $n$. 

---

List of Figures xvii
5.1 Venn diagrams of the sets of the solutions to the BAP, the SeqBAP, and the LexBAP. ............................................................... 58
5.2 Demonstration of PruneSeq with four agents $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ and four tasks $\mathcal{B} = \{b_1, b_2, b_3, b_4\}$. The set of edges $\mathcal{E}_B$ are arranged in order of ascending weight, where $e_{pq}$ is the edge between agent $a_p$ and task $b_q$. The circled edges represent edges in $\mathcal{M}$. At each iteration, a warm-start of PruneBAP is applied since we already know the circled edges from the previous iteration. ........................................... 63
5.3 Comparison of the runtime of an exact LexBAP algorithm, a naive greedy approach, and PruneSeq. .......................................................... 64
5.4 Demonstration of PruneSeq in a distributed setting. The shaded circles show the communication range of each agent. Agents are only able to communicate with other agents within their circles. .................................................. 65

6.1 Illustration of empty set of allowable acceleration in $\mathbb{R}^2$. The blue shaded region represents the halfspace of possible accelerations agent $a \in \mathcal{A}$ can take to maintain the safety distance with agent $a'' \in \mathcal{A}$. The orange shaded region represents the halfspace of possible accelerations agent $a$ can take to maintain the safety distance with agent $a' \in \mathcal{A}$. ............................................................................. 79
6.2 Demonstration of the hybrid controller in (6.15) with the condition in (6.12) resulting in the violation of the safety distance when there are three agents. The braking of agent $a'$ is inadvertently an acceleration in the direction of agent $a$ .................................. 81
6.3 Illustration of apparent relative velocity being an upper bound on the sum of individual velocities of two agents. Using apparent relative velocity forces agents to maintain larger seperation than in (6.16). ................................................. 83
This page intentionally left blank.
List of Tables

4.1 Comparison of AugDFS and AugBFS. .............................................. 43
This page intentionally left blank.
Nomenclature

Acronyms

BAP Bottleneck Assignment Problem
BFS Breadth-First Search
CBF Control Barrier Function
DFS Depth-First Search
LexBAP Lexicographic Bottleneck Assignment Problem
MCM Maximum Cardinality Matching
SAP Sum Assignment Problem
SeqBAP Sequential Bottleneck Assignment Problem

Notation

$\mathbb{R}$ The set of real numbers
$\mathbb{Z}$ The set of integers
$\mathbb{Z}^+$ The set of strictly positive integers
$\|v\|_p$ The $p$-norm of a vector $v$
$\mathcal{G}$ An undirected graph
$\mathcal{V}$ The vertex set of $\mathcal{G}$
$\mathcal{E}$ The edge set of $\mathcal{G}$
$\mathcal{G}_B$ A bipartite graph
$\mathcal{V}_B$ The vertex set of $\mathcal{G}_B$
$\mathcal{E}_B$ The edge set of $\mathcal{G}_B$
$I$ The identity matrix
$0$ The zero vector
$|A|$ The cardinality of a set $A$
$A \cup B$ The union of sets $A$ and $B$
$A \cap B$ The intersection of sets $A$ and $B$
$A \setminus B$ The set difference of sets $A$ and $B$
$A \oplus B$  The symmetric difference of sets $A$ and $B$
Chapter 1

Introduction

This thesis focuses on two aspects of distributed decision-making in multi-agent systems. Of particular interest are assignment problems, which involve agents making decisions on how to allocate a given collection of tasks amongst themselves. Distributed decision-making in this context refers to the fact that agents compute this assignment without the help of a centralised decision-maker. Once the agents have been assigned to their tasks, subsequent decisions are required for ensuring each agent can successfully carry out their tasks. Specifically, this thesis explores how mobile robot agents can place constraints on their set of allowable actions that guarantee collision avoidance for the team of agents as they move to complete their tasks, again without the aid of a centralised decision-maker to coordinate the actions of all the agents. In other words, each individual agent must decide its own actions as it moves to complete its assigned task while still guaranteeing collisions with every other agent are avoided. In Section 1.1, a scenario is presented to motivate the types of assignment problems and collision avoidance approaches covered in this thesis and a review of related work is conducted. In Section 1.2, a breakdown of the upcoming chapters and their contributions is provided.

1.1 Motivation and Literature Review

Consider the problem of protecting an asset from incoming threats using a team of agents [1]. The agents must position themselves along the trajectory of the threats in order to intercept them before they reach the asset. In this case, a “task” for an agent is to move from its initial position to a position where it can intercept a threat. This leads to an assignment problem since we have a set of agents and a set of tasks and we need to pair up the agents to the tasks so that each threat is intercepted.
Once in position, each agent can then seduce its corresponding threat away from the asset by causing the threat to change its radar track from the asset to the agent, i.e., breaking the tracking lock of the threat on the asset, hence the name “break-lock” decoys or seduction decoys [2]. In order to do this, these decoys must reach the appropriate position along the trajectory of the threat in a timely fashion. Therefore, a suitable objective for this problem is to minimise the worst-case positioning time of the decoys. This scenario provides motivation for the particular types of assignment problems discussed in this thesis, which aim to fulfill this objective.

Furthermore, as agents move to position themselves to intercept the threats, they move about within the same environment so there is potential for their paths to intersect. This demonstrates the need for collision avoidance protocols between agents. In fact, it may also be desirable for agents to have the ability to avoid obstacles in the environment in addition to other agents. As such, this scenario also provides motivation for the collision avoidance approaches investigated in this thesis.

For both the assignment phase and the task execution with collision avoidance phase, agents must coordinate without the help of a centralised decision-maker. Note that a centralised decision-maker is not without its own benefits. For example, the heavy computation can be handled from a fixed location with larger processing capability, leaving the agents with a smaller computational burden. However, having all computation carried out by a centralised decision-maker would mean agents must provide all local information to this centralised decision-maker, some of which may not be necessary. This is one of the draws of distributing computation amongst agents. With distributed decision-making, agents make their own local decisions and instead only communicate information to other agents that is necessary for converging to a global decision. This does not imply that less communication is required in a distributed approach, only that the centralised decision-maker requires aggregation of all information upfront. In this asset protection scenario, it is desirable that agents distribute the computation amongst themselves and not rely on centralised computation.

Figure 1.1 shows an example of the asset protection scenario. Depending on where the decoys are initially located, they take different amounts of time to move into a position along the trajectory of a given threat from which they can effectively seduce that threat away from the asset. This scenario is provided as motivation but the concepts investigated in this thesis can be employed in a wide variety of other applications, especially time-critical ones where there is a need to minimise the worst-case completion time. For example, a similar type of problem
arises in a ride-sharing service, where the agents are cars and they need to pick up customers at different locations in a city. Another example might be a drone delivery service, where the agents are delivery drones and they need to deliver packages to different houses. The similarity here is that we again aim to minimise the worst-case waiting time of the customers.

These example applications are typically approached from the direction of coordination of distributed robots or vehicles, most commonly Unmanned Aerial Vehicles (UAV)s [3–6]. Although there is a wide variety of applications for UAVs, task assignment in this domain is largely focused on minimising the sum of costs as an objective as opposed to the time-based objective in the asset protection scenario. This is despite the fact that minimising the sum of costs may not be the reasonable objective in every UAV mission. The asset protection scenario and the other application examples expose a gap in the robotics literature for carrying out task assignment with a focus on this time-critical aspect. Before discussing assignment objectives in more detail, we note that this gap is not surprising as cooperative robotics is a broad area with many other focuses. Some common focuses in the literature on cooperation and coordination of robots are discussed below.

The task assignment problem is just one of several topics in cooperative multi-robot systems. Other topics include consensus, formation control, swarming and estimation [4, 7–10]. Consensus or agreement is the problem of agents reaching global agreement on a state. Formation control concerns moving agents while maintaining them in geometric formations. Swarming refers to having agents mimic the behaviour of animals in nature such as flocking in birds. Estimation refers to coordinating a group of local distributed sensors to estimate some global information.
The authors in [11–13] classify task assignment problems according to the task type, robot
type and allocation type. Since we are viewing this from the perspective of cooperative robotics,
the agents in these problems are specifically referred to as robots. The task type refers to the
number of robots required per task, i.e., in some cases a single task may require cooperation
from multiple robots. The robot type refers to the number of simultaneous tasks a robot can
complete, i.e., in some cases a robot may have the ability to perform multiple tasks at a time.
The allocation type refers to the scheduling of tasks, i.e., in some cases a robot may perform
multiple consecutive tasks. For example in [14], the authors consider two types of scheduling
problems. The first problem is referred to as a cycle problem where robots must start at a
common depot, move to complete their allocated tasks and return to the depot. The second
is referred to as a path problem where the robots start at arbitrary locations and do not need
to return to their start locations after completing their tasks. For both types of scheduling
problems, the authors develop approximation algorithms to generate the robots’ schedules.

In this thesis, we limit discussion to problems where tasks require only one agent to be com-
pleted and agents each carry out only one task, which is simply known as an assignment prob-
lem in combinatorial optimisation. We note the distinction in wording here between a “task
assignment problem” in cooperative robotics and an “assignment problem” in combinatorial
optimisation. Hence, the motivating scenario lies at the intersection of combinatorial optimisa-
tion and cooperative robotics and it shows a gap in literature for task assignment that caters
specifically to a timing objective. In order to delve into objectives and optimality, we next con-
sider literature related specifically to assignment problems from the combinatorial optimisation
perspective.

An assignment problem arises when multiple tasks are to be allocated to multiple agents. The
asset protection scenario illustrates one application that can be cast as an assignment problem.
Assignment problems are a type of combinatorial optimisation problem [15, 16] and the optimal
assignment would therefore depend on the objective function, which is chosen depending on
the situation or application. Reviews on the different kinds of objective functions for assign-
ment problems are found in [17–19]. In this thesis, we consider linear assignment problems [20],
where the objective functions are linear in the costs, as opposed to so-called quadratic assign-
ment problems [21–25], which take the form of quadratic programs with quadratic costs in the
objective function.

In the context of the asset protection scenario, the objective there is to minimise the worst-case
positioning time of the decoys. This type of assignment problem where we wish to minimise the bottleneck or most expensive agent-task pairing is aptly known as the Bottleneck Assignment Problem (BAP) [26–31]. Typically, such an assignment is necessary in time-critical problems where tasks are carried out by agents simultaneously and the time taken to complete all tasks must be minimised. We contrast the objective of the BAP to that of the Sum Assignment Problem (SAP) [32–37], where the goal is to instead minimise the sum of costs. The SAP may be useful, for instance, in minimising the total fuel consumption of a team of agents as they move to complete their tasks. However, minimising the sum of costs may result in some agents consuming very little fuel at the expense of other agents consuming a lot of fuel, i.e., the SAP may result in an assignment that is less equitable than the BAP in terms of spreading the costs across a team of agents. A special case of the BAP is the Lexicographic Bottleneck Assignment Problem (LexBAP) [38–40]. This type of assignment problem focuses not only on minimising the most expensive agent-task pairing, but also the second costliest, and the third costliest, etc. There may be different reasons to additionally focus on the subsequent pairings in the LexBAP, i.e., the LexBAP has several desirable properties. In this thesis, we focus on obtaining inter-agent collision avoidance guarantees from solving the LexBAP; we review collision avoidance in the later paragraphs of this section.

Primarily, the BAP and LexBAP are the two types of assignment problems that we explore in the coming chapters. As mentioned previously, task assignment from the perspective of cooperative robotics focuses heavily on the SAP while neglecting the BAP and the LexBAP. We address this gap by analysing these two assignment problems and developing distributed algorithms for solving them.

Recall from the asset protection scenario above that we are interested in solving assignment problems with distributed computation since, as discussed previously, distributed algorithms remove the need for a central decision-maker to carry out all computations and aggregate all the necessary information. There are many distributed algorithms for assigning tasks to agents for the SAP. However, there is a gap in literature for distributed algorithms that solve the BAP and the LexBAP. The Hungarian Method in [32] is a well-studied algorithm that solves the SAP in a centralised manner and in [41], a distributed method of executing the Hungarian Method is developed. The Consensus-Based Auction Algorithm (CBAA) in [6] is an example of a distributed greedy SAP algorithm, which greedily and myopically chooses agent-task pairings one after another. We discuss greedy algorithms in more detail in a later paragraph. Another
distributable greedy SAP algorithm for an assignment problem using submodular utility functions is presented in [42]. Furthermore, there exist distributed algorithms in [6, 43, 44] for assigning multiple tasks per agent, i.e., the type of assignment problems classified in [11–13] as time-extended assignment problems. The availability of literature on distributed algorithms for the SAP and other types of assignment problems contrasts the lack of literature on distributed algorithms that solve the BAP and the LexBAP.

Existing centralised BAP algorithms are not immediately amenable to being translated into distributed algorithms. The threshold algorithm in [26] involves a central decision-maker calculating an initial threshold cost and checking if an assignment can be created using only allocations with costs smaller than the threshold. The threshold is iteratively increased until such an assignment is found, and the first such assignment corresponds to the optimal assignment for the BAP. The threshold algorithm requires the central decision-maker to form a matrix with elements being the costs of every allocation, with each row corresponding to an agent and each column corresponding to a task. In [27, 28], improvements on the completion time of the threshold algorithm are made by sorting the costs and moving the threshold according to a binary search pattern rather than incrementally. In [29], an algorithm solves the BAP over a subset of the total agents and tasks, iteratively increasing the size of the subset until it contains all the agents and tasks. The solution to the BAP over a given subset is found using the solution to the BAP over the previous subset. The algorithms in [30, 31] are based on finding so-called augmenting paths and are more closely related to the distributed BAP algorithm discussed in this thesis. Finding an augmenting path with distributed agents poses a challenge as it requires information from multiple agents. The distributed BAP algorithm developed in this thesis also makes use of augmenting paths and finding an augmenting path in a bipartite graph in a distributed manner is one main contributions of Chapter 4.

One method of solving assignment problems is through greedy algorithms [6, 45–49]. A greedy algorithm sequentially pairs up one task and one agent from the remaining available choices, where each pair is chosen short-sightedly with no regard to the affect on subsequent choices. As such, the use of greedy methods generally results in suboptimal solutions with respect to the objective function of the intended assignment problem. However, the trade-off is that greedy approaches typically are lower in computational complexity than ones that produce exact solutions to assignment problems. In this thesis, we consider a distributed greedy approach for solving the LexBAP and discuss the conditions in which this greedy approach produces an
exact solution to the LexBAP.

Furthermore, the asset protection scenario demonstrates the need for collision avoidance guarantees between agents as they move to position themselves to intercept the threats. This thesis covers two different approaches to achieve this. The first approach considers how collision avoidance can be guaranteed as an outcome of assigning agents in a particular way during the task assignment phase, i.e., we investigate collision avoidance as a feature that emerges as a result of solving the LexBAP. The second approach considers how collisions can be avoided through the use of Control Barrier Functions (CBF)s [50–56], which are typically used for guaranteeing safety. Safety is a more general concept than inter-agent collision avoidance, involving guarantees or verification that a system remains within a safe set of states [57–60].

Motion control with inter-agent collision avoidance is a heavily researched problem and many different strategies exist, e.g., reciprocal collision avoidance [61, 62], CBFs [63–67], buffered Voronoi cells [68], and multi-agent rapidly-exploring random trees [69]. In [70] independent trajectories for all possible agent-destination pairs are designed, tasks are allocated by selecting an assignment that minimises the sum of costs of the chosen trajectories, and collision are avoided by subsequently delaying the starting times of certain agents. Collision avoidance awareness criteria are included in a task assignment problem in [71] by designing a quadratic assignment cost function that incentivises collision-free paths. In this case, collision avoidance is not guaranteed, it is only penalised in the assignment stage.

Similar to [72–74], we investigate intrinsic properties of assignment problems that provide conditions for which inter-agent collisions are avoided. In [72] it was first shown that straight lines connecting agents to allocated destinations do not intersect if the allocation corresponds to the solution of the SAP, with Euclidean distances between initial agent positions and target destinations as assignment costs. Conditioned on sufficiently large distances between initial positions and target locations, a time parametrisation of straight-line trajectories is derived in [74] such that agents with finite extent do not collide when following an assignment that solves an SAP, with squared distances as assignment costs. The non-intersection properties of the SAP do not hold for the BAP. However, assuming agents are point-masses that travel with constant velocity on straight paths towards their allocated destinations, it is shown in [73] that no collisions occur if the task allocation is based on either the LexBAP or a designated assignment problem which combines the BAP with the SAP. In this thesis, we extend this work by assuming that agents have finite extent, consider arbitrary distance metrics, and do not limit
the trajectories to straight lines. In particular, we show sufficient conditions for producing a unique optimising solution of the LexBAP and if these conditions are satisfied, this LexBAP solution leads to time-varying safe sets of positions for agents as opposed to just straight line trajectories.

In the context of the asset protection scenario that we are using as a motivating example, intrinsic collision avoidance from solving assignment problems is desirable as we can tackle two issues at the same time. Not only do we assign agents and tasks such that the objective of minimising the worst-case positioning time of an agent is met, but we also guarantee that the agents do not collide with each other as they move to their goal positions by solving the LexBAP. However, we also wish to explore collision avoidance separate to the assignment problem as this would allow flexibility of guaranteeing collision avoidance regardless of the choice of assignment objective. Moreover, we also need to consider the case when the sufficient conditions for deriving the safe position sets via the LexBAP are not satisfied. In other words, we do not wish to rely solely on the intrinsic assignment problem properties to achieve collision avoidance. Therefore, we now consider the gaps in existing literature on collision avoidance to develop a suitable approach for the motivating asset protection scenario. In [61, 62, 68], collision avoidance guarantees are provided for agents with velocity as an input. This means the guarantees do not extend to higher order dynamics for agents, e.g., agents with acceleration input. In [69], we require additional computation in the path planning stage, where agents first generate paths to their goal positions and then agents must modify their paths until all agents have collision-free paths to their goals. Similarly in [70], the agents are given a priority ordering and agents with lower priority delay their starting times to avoid collisions with agents of higher priority. This approach requires lower priority agents to have full knowledge of the trajectories of the higher priority agents in order to compute the necessary start time delay. With this in mind, we are interested in approaches that guarantee collision avoidance for agents with three criteria. Firstly, as has been the overarching goal, we wish to develop an approach that is amenable to having computation distributed across agents. Secondly, we wish to consider agents with acceleration input dynamics, not the lower-order velocity input model. Thirdly, we wish to constrain the time-dependent acceleration inputs of the distributed agents to achieve collision avoidance rather than requiring the explicit construction of collision-free paths beforehand. This last criterion relates to the fact that it requires additional computational overhead to compute collision-free paths for all agents, as opposed to an ad-hoc approach where agents make changes to their trajectories while in motion and only when collisions are imminent.
To that extent, we consider the work in [63, 64], which use CBFs to derive safe sets of accelerations for agents modelled as double integrators to avoid collisions. We discuss the issues facing the methods in [63, 64] regarding feasibility and coupling between distributed agents and develop a novel CBF approach that addresses these issues. This approach is in turn extended to incorporate collision avoidance guarantees for obstacles modelled as convex polytopes, i.e., the safe sets of acceleration for agents takes into account both collisions with other agents as well as collisions with convex polytopic obstacles. In [75, 76], repulsive potential field approaches are applied for obstacle avoidance and as such do not offer collision avoidance guarantees. In [77], obstacle avoidance is cast as an optimisation problem that rewards tracking the desired trajectory and penalises collisions with obstacles. Reviews of obstacle avoidance in the context of the broader path planning and navigation problem are found in [78, 79].

The following summarises the gaps in literature this thesis will address and the corresponding research aim undertaken in this thesis.

- **Gap**: Lack of multi-agent task assignment with BAP objective.
  - **Aim**: Identify exploitable structure for solving the BAP both efficiently and with distributed computation.

- **Gap**: Non-existing distributed BAP algorithm.
  - **Aim**: Develop a distributed algorithm for the BAP.

- **Gap**: Non-existing distributed LexBAP algorithm.
  - **Aim**: Develop a distributed algorithm for the LexBAP.

- **Gap**: Intrinsic collision avoidance guarantees from assignment problems are only for straight line trajectories.
  - **Aim**: Derive time-varying safe sets of agent positions from solving the LexBAP.

- **Gap**: Issues with methods for guaranteeing interagent collision avoidance and obstacle avoidance in agents with acceleration input without explicit trajectory generation.
  - **Aim**: Address issues in existing CBF methods.

### 1.2 Contributions

The following is an outline of the contributions in each of the upcoming chapters.
Chapter 2:

This chapter introduces preliminaries and background concepts, but it does not contain any novel contributions. It covers the graph theory, assignment problem formulations and distributed setting that are necessary for the rest of the thesis.

Chapter 3:

This chapter contains novel concepts on the structure of the BAP that are exploited in the following chapters. In particular, two important tools are introduced, i.e., the critical edge and the price of absence. The former is vital in developing the algorithm for solving the BAP in Chapter 4, while the latter is needed in the greedy algorithm for solving the LexBAP in Chapter 5. We derive important properties of these two tools that are exploited in the later chapters and explore the correlation between them, showing how both rely on the existence of so-called augmenting paths. To cap off the structural analysis of the BAP, we demonstrate the role of augmenting paths in merging two separate BAPs into a single combined problem, i.e., we establish the significance of augmenting paths by exploring the ideas of clustering agents and forming alternating trees. The idea of merging BAPs is related to an important warm-starting property of the BAP algorithm in Chapter 4, which is in turn crucial for the complexity analysis of the LexBAP algorithm in Chapter 5, while the alternating tree is a useful pedagogical tool for explaining parts of the BAP algorithm. These results appear in [80, 81].

Chapter 4:

This chapter presents an algorithm to solve the BAP that is amenable to having computation distributed across agents. We leverage the concepts developed in Chapter 3 to first discuss the algorithm in its centralised form. Then, we examine how each component of the algorithm can be executed with distributed computation. Given the importance of augmenting paths in the BAP, we focus on one component in particular, i.e., the component related to searching for augmenting paths and compare two methods of conducting such a search. One noteworthy property of the algorithm is highlighted, i.e., the warm-starting property, which allows the algorithm to exploit prior information when converging to a solution of the BAP. A numerical analysis of the BAP algorithm is presented, where we investigate the empirical complexity of the algorithm and benchmark it against an existing distributed algorithm. These results appear in [82, 83].
Chapter 5:

This chapter presents a novel greedy and distributable algorithm to solve the LexBAP with conditions on exactness of the solution. Once again, we leverage the concepts in Chapter 3 to discuss this algorithm. Since it is a greedy approach, we also explore conditions for when the algorithm produces an exact solution to the LexBAP. The numerical complexity of the LexBAP algorithm is investigated via a case study. These results appear in [81].

Chapter 6:

This chapter explores two different approaches to guarantee collision avoidance of mobile robot agents. The first approach involves characterising a set of time-varying safe positions for agents, which can be obtained as a result of solving the LexBAP. The conditions in which such constraints can be derived from solving the LexBAP are shown to be related to the conditions for when the greedy algorithm in Chapter 5 produces an exact solution to the LexBAP. The second approach involves characterising a set of time-varying safe acceleration inputs for agents, which are derived via CBFs. As long as agents choose accelerations from within the safe set, they are guaranteed to avoid collisions with other agents. We extend this and also provide safe accelerations for agents to avoid collisions with obstacles modelled as convex polytopes. These results appear in [47, 81].

Chapter 7:

This chapter consists of the conclusion and future works.
Chapter 2

Background

In this chapter, we consider the background concepts that are used in the following chapters, namely the preliminary graph theory, the assignment problem formulations and the distributed setting. The graph theory in Section 2.1 is not novel and can be found, for example, in [7, 82, 84–86].

2.1 Graph Theory

2.1.1 Undirected Graph

Consider an undirected graph $G = (V, E)$, where $V$ is a set of vertices and $E$ is a set of edges. We represent edges as a set of two distinct vertices, e.g., an edge $e = \{v, v'\} \in E$ with vertices $v, v' \in V$. Figure 2.1 shows an undirected graph. Given $G$, we consider the following definitions.

Figure 2.1: Illustration of an undirected graph. The circles represent vertices and the lines represent edges.
Definition 2.1 (Maximum Cardinality Matching). A matching $\mathcal{M}$ of $\mathcal{G}$ is a subset of edges $\mathcal{M} \subseteq \mathcal{E}$ such that no vertex $v \in \mathcal{V}$ is incident to more than one edge in $\mathcal{M}$. A Maximum Cardinality Matching (MCM) is a matching of $\mathcal{G}$ with maximum cardinality.

Definition 2.2 (Neighbours). The set of neighbours of vertex $v \in \mathcal{V}$ in $\mathcal{G}$ is $N(\mathcal{G}, v) := \{v' | \{v, v'\} \in \mathcal{E}\}$.

Definition 2.3 (Path). Given distinct vertices $v_1, v_2, ..., v_{l+1} \in \mathcal{V}$ such that for all $k \in \{1, 2, ..., l\}$, $v_{k+1} \in N(\mathcal{G}, v_k)$, the set of edges $P(\mathcal{E}, v_1, v_{l+1}) := \{\{v_k, v_{k+1}\} \in \mathcal{E}| k \in \{1, 2, ..., l\}\}$ is a path between $v_1$ and $v_{l+1}$ of length $l$.

Definition 2.4 (Diameter). The diameter of $\mathcal{G}$ is $D(\mathcal{G}) := \ max_{v,v' \in \mathcal{V}} \delta(v, v')$, where $\delta(v, v')$ is the length of the shortest path between vertices $v, v' \in \mathcal{V}$.

Definition 2.5 (Alternating path). A path $P$ is an alternating path relative to a matching $\mathcal{M}$ if and only if each vertex that is incident to an edge in $P$ is incident to no more than one edge in $P \cap \mathcal{M}$ and no more than one edge in $P \setminus \mathcal{M}$.

Definition 2.6 (Free vertex). A vertex $v \in \mathcal{V}$ is free relative to a matching $\mathcal{M}$ if and only if for all $v' \in \mathcal{V}$, $\{v, v'\} \notin \mathcal{M}$.

Definition 2.7 (Augmenting path). A path $P = P(\mathcal{E}, v, v')$ between vertices $v, v' \in \mathcal{V}$ is an augmenting path relative to a matching $\mathcal{M}$ if and only if $P$ is an alternating path relative to $\mathcal{M}$ and $v$ and $v'$ are both free vertices relative to $\mathcal{M}$.

2.1.2 Tree Graph

Definition 2.8 (Tree). A tree is a graph in which any two vertices are connected by exactly one path.
Figure 2.3 shows a tree. Given a tree $G_T = (V_T, E_T)$, we can designate one vertex $v_r \in V_T$ as the root vertex of $G_T$. We define the length of the path of a vertex to the root vertex as the level of the vertex.

**Definition 2.9 (Level).** The level of a vertex $v \in V_T$ in $G_T$ is $L(G_T, v) := |P(E_T, v, v_r)|$.

**Definition 2.10 (Parent and child).** The parent of vertex $v \in V_T$ in $G_T$ is the vertex $v' \in V_T$ such that $P(E_T, v, v_r) = P(E_T, v', v_r) \cup \{\{v, v'\}\}$. A vertex is a child of its parent.

The root itself does not have a parent vertex. We can extend Definition 2.10 to a grandparent, grandchild, ancestor, or descendant of a vertex in $G_T$.

**Definition 2.11 (Grandparent and grandchild).** The grandparent of vertex $v \in V_T$ in $G_T$ is the vertex $v'' \in V_T$ such that $P(E_T, v, v_r) = P(E_T, v', v_r) \cup \{\{v, v'\}, \{v', v''\}\}$. A vertex is a grandchild of its grandparent.

**Definition 2.12 (Ancester and descendant).** An ancester of vertex $v \in V_T$ in $G_T$ is a vertex $v' \in V_T$ such that $P(E_T, v', v_r) \subset P(E_T, v, v_r)$. A vertex is a descendant of its ancester.

### 2.1.3 Weighted Bipartite Graph

Consider a bipartite graph $G_B = (V_B, E_B)$, with vertex set $V_B = A \cup B$ and edge set $E_B \subseteq \{(a, b) | a \in A, b \in B\}$, where $A$ is a set of agents and $B$ is a set of tasks such that $A \cap B = \emptyset$. Let $X(G_B)$ be the set of all MCMs of $G_B$. Let the function $W : E_B \rightarrow \mathbb{R}$ map the edges of $G_B$ to real-valued weights. Figure 2.4 shows a weighted bipartite graph.
2.2 Common Assignment Problem Formulations

2.2.1 The Sum Assignment Problem

The SAP for a bipartite graph $G_B$ is formulated as

$$\text{SAP} : \arg \min_{\mathcal{M} \in X(G_B)} \sum_{e \in \mathcal{M}} W(e).$$

2.2.2 The Bottleneck Assignment Problem

The BAP for a bipartite graph $G_B$ is formulated as

$$\text{BAP} : \arg \min_{\mathcal{M} \in X(G_B)} \max_{e \in \mathcal{M}} W(e). \quad (2.1)$$

We define the bottleneck weight, bottleneck assignment and the bottleneck edge of $G_B$.

**Definition 2.13** (Bottleneck weight). The bottleneck weight of $G_B$ is $Y(G_B) := \min_{\mathcal{M} \in X(G_B)} \max_{e \in \mathcal{M}} W(e)$. 

Figure 2.4: Illustration of a weighted bipartite graph. The circles represent vertices and the lines represent edges. White vertices represent agents $A$ and black nodes represent tasks $B$. 

---

*Background*
Definition 2.14 (Bottleneck assignment). The set of bottleneck assignments of $G_B$ is $S(G_B) := \arg \min_{M \in X(G_B)} \max_{e \in M} W(e)$.

Definition 2.15 (Bottleneck edge). A bottleneck edge of $G_B$ is an edge $e_b \in E_B$ such that there exists a bottleneck assignment $M \in S(G_B)$ for which $e_b \in \arg \max_{e \in M} W(e)$. The set of all bottleneck edges of $G_B$ is $Z(G_B) := \arg \max_{e \in \bigcup_{M' \in S(G_B)} M'} W(e)$.

2.2.3 The Lexicographic Bottleneck Assignment Problem

Consider an MCM $M \in X(G_B)$. Let $n = |M|$. Without loss of generality, we denote the edges in $M$ as $M := \{\{a_1, b_1\}, \{a_2, b_2\}, ..., \{a_n, b_n\}\}$ and assume $W(\{a_1, b_1\}) \geq W(\{a_2, b_2\}) \geq ... \geq W(\{a_n, b_n\})$. We define the ordered tuple of weights $T(M) := (W(\{a_1, b_1\}), W(\{a_2, b_2\}), ..., W(\{a_n, b_n\}))$ and the $k$th element of the tuple $T_k(M) := W(\{a_k, b_k\})$, for $k \in \{1, 2, ..., n\}$. Let the function $K : X(G_B) \rightarrow \mathbb{R}$ map the MCMs of $G_B$ to their lexicographic order, where the lexicographic order for two MCMs $M, M' \in X(G_B)$ is defined as follows.

Definition 2.16 (Lexicographic order). An MCM $M$ is lexicographically smaller than $M'$, i.e., $K(M) < K(M')$, if and only if there exists $k \in \{1, 2, ..., n\}$ such that $T_k(M) < T_k(M')$ and for all $l \in \{z \in \mathbb{Z}^+ | z < k\}$, $T_l(M) = T_l(M')$. An MCM $M$ is lexicographically equal to $M'$, i.e., $K(M) = K(M')$, if and only if for all $k \in \{1, 2, ..., n\}$, $T_k(M) = T_k(M')$.

For example, an MCM $M$ with tuple of weights $T(M) = (5, 3, 3, 3)$ is lexicographically smaller than an MCM $M'$ with tuple $T(M') = (5, 4, 1, 1)$ because its second element is smaller and their first elements have equal weight. The LexBAP for a bipartite graph $G_B$ is formulated as

$$\text{LexBAP} : \arg \min_{M \in X(G_B)} K(M).$$

2.3 Distributed Computation

The terms distributed and decentralised have a wide variety of connotations in literature [87–91], including the computer science domain [92, 93]. Broadly, it involves restricting agents from accessing information so that no single agent has knowledge of all relevant information in the problem. In this thesis, we use the following assumptions to describe distributed computation.
Assumption 2.1. Assume an agent \( a \in \mathcal{A} \) has access to the set of incident edges \( \mathcal{E}_a := \{\{a, b\} | b \in N(G_B, a)\} \) and access to the weight of each edge in \( \mathcal{E}_a \), i.e., the set \( \mathcal{W}_a := \{W(e) | e \in \mathcal{E}_a\} \).

All the edges in \( \mathcal{E}_B \) are stored collectively by agents, i.e., \( \mathcal{E}_B = \bigcup_{a \in \mathcal{A}} \mathcal{E}_a \). However, no agent has access to edges stored by another agent, i.e., \( \mathcal{E}_a \cap \mathcal{E}_{a'} = \emptyset \) for \( a, a' \in \mathcal{A}, a \neq a' \).

Assumption 2.2. Assume communication between agents is modelled by a time invariant, undirected and connected graph \( G_C = (\mathcal{A}, \mathcal{E}_C) \) with vertex set \( \mathcal{A} \), edge set \( \mathcal{E}_C \) and diameter \( D \). Assume all agents \( a \in \mathcal{A} \) communicate synchronously according to a global clock.

Remark 2.1. An MCM \( \mathcal{M} \) of \( G_B \) can be jointly stored by agents. For each agent \( a \in \mathcal{A} \) that is not free according to Definition 2.6, let that agent store its matched task \( \mu_a \) in \( \mathcal{M} \), where task \( \mu_a \in \mathcal{B} \) is the task for which \( \{a, \mu_a\} \in \mathcal{M} \). For free agent \( a' \in \mathcal{A} \), we henceforth use \( \mu_{a'} = \hat{b} \) to signify agent \( a' \) is not matched to a task, where \( \hat{b} \notin \mathcal{B} \) is a “dummy” task. Let the function \( Q : X(G_B) \times \mathcal{A} \rightarrow \mathcal{B} \cup \{\hat{b}\} \) map an agent to the task that it is matched to in a given MCM of \( G_B \).
Chapter 3

Structure in The Bottleneck Assignment Problem

3.1 Introduction

In this chapter, we discuss structure in the BAP that is exploited in the following chapters. We introduce novel graph theoretic concepts that provide insight into the development of a BAP algorithm that is amenable to distributed computation as well as concepts that are useful for developing a greedy distributable LexBAP algorithm. Furthermore, we demonstrate the vital role of augmenting paths in the BAP by considering the problem of merging two separate BAPs.

3.2 Relationship Between an Augmenting Path and an MCM

We begin with general results for a graph, i.e., the following concepts are for an undirected graph $G = (V, E)$, as introduced in Section 2.1.1. Lemmas 3.1 and 3.2 are not novel but are used to prove Proposition 3.1, which equates the existence of an augmenting path to the existence of a new MCM within a set of edges.

**Lemma 3.1** (Proof in [84]). If $M$ is a matching of $G$ and $P$ is an augmenting path relative to $M$, then their symmetric difference $M ⊕ P$ is also a matching of $G$ and $|M ⊕ P| = |M| + 1$.

**Lemma 3.2** (Proof in [84, 94], Berge’s Theorem). A matching $M$ of $G$ is an MCM of $G$ if and only if there is no augmenting path relative to $M$ in the edge set $E$.

**Proposition 3.1.** Let $M$ be an arbitrary MCM of $G$. Let edge set $E'$ be such that $M ⊆ E' ⊆ E$. Consider an edge $e ∈ M$. An augmenting path $P ⊆ E' \{e\}$ exists relative to $M \{e\}$ if and only if there exists an MCM $M'$ of $G$ such that $M' ⊆ E' \{e\}$.
Proof. First, we prove necessity. Assume there exists an augmenting path $P \subseteq E' \setminus \{e\}$ relative to $M \setminus \{e\}$. By Lemma 3.1, $M' = M \setminus \{e\} \oplus P$ is an MCM of $G$. Since both $P$ and $M \setminus \{e\}$ are subsets of $E' \setminus \{e\}$, their symmetric difference is also a subset of $E' \setminus \{e\}$.

Next, we prove sufficiency. Assume there does not exist an augmenting path $P \subseteq E' \setminus \{e\}$ relative to $M \setminus \{e\}$. By Lemma 3.2, the matching $M \setminus \{e\}$ is an MCM of the graph $(V, E' \setminus \{e\})$. Since $M \setminus \{e\}$ has cardinality $|M| - 1$, it is not an MCM of $G$; we know that $M$ is an MCM of $G$ and it has cardinality $|M|$. Thus, there does exist an MCM of $G$ within the set $E' \setminus \{e\}$. 

Proposition 3.1 is a property of augmenting paths that can be exploited to determine if an MCM is a solution to the BAP. To achieve this, we combine the concept of reducing the cardinality of a set of edges in Proposition 3.1 with the concept of a pruned edge set introduced in the following section. In other words, the existence of an augmenting path within a pruned edge set is a crucial condition that we exploit in the algorithm to solve the BAP in Chapter 4.

3.3 Structure for Solving the BAP

The following concepts are defined using a weighted bipartite graph $G_B = (V_B, E_B)$, as introduced in Section 2.1.3. A pruned edge set is a tool that allows partitioning of the edge set $E$ using the edge with largest weight in an MCM as the partitioning criterion.

Definition 3.1 (Pruned edge set). Given an MCM $M$ of $G_B$, a pruned edge set of $G_B$ relative to $M$ is $\phi(G_B, M) := M \cup \{e \in E_B | W(e) < \max_{e' \in M} W(e')\}$.

With respect to its applications for the BAP, every pruned edge set of a weighted bipartite graph must contain at least one bottleneck edge, which is a vital observation for this set of edges. With that in mind, we focus on the relationship between an augmenting path and one edge of an MCM in particular.

Definition 3.2 (Critical edge). An edge $e_c$ is a critical edge of $G_B$ relative to an MCM $M$ of $G_B$ if and only if $e_c \in \arg \max_{e \in M} W(e)$ and $\phi(G_B, M) \setminus \{e_c\}$ does not contain an augmenting path relative to $M \setminus \{e_c\}$. 
3.3.1 Properties of a Critical Edge

Corollary 3.1 follows from combining Definition 3.2 and Proposition 3.1. We apply Proposition 3.1 to show that a particular edge in an MCM having such properties in Definition 3.2 means that the edge must be one of the bottleneck edges contained within a pruned edge set. This relationship is exploited in the development of the BAP algorithm in Chapter 4 as Definition 3.2 shows specific criteria for determining if an edge is a critical edge. Hence, we have specific criteria to determine if an MCM is a solution to the BAP. A proof for this corollary is provided as some steps are not self-evident.

**Corollary 3.1.** A critical edge of $G_B$ is a bottleneck edge of $G_B$.

**Proof.** We first apply Definition 3.2 for a critical edge. Assume $e_c$ is a critical edge relative to an MCM $M$ of $G_B$. This implies $e_c \in \arg \max_{e \in M} W(e)$ and $\phi(G_B, M) \setminus \{e_c\}$ does not contain an augmenting path relative to $M \setminus \{e_c\}$. By applying Proposition 3.1 to $G_B$ with the pruned edge set $\phi(G_B, M) = E'$, there does not exist an MCM of $G_B$ in $\phi(G_B, M) \setminus \{e_c\}$, i.e., all MCMs of $G_B$ must either contain $e_c$ or must contain edges with equal or larger weight than $W(e_c)$. Thus, $M$ is a bottleneck assignment and $e_c$ is a bottleneck edge of $G_B$.

We summarise the relationship between a critical edge, a pruned edge set and an augmenting path by focusing on the agent and task vertices of a critical edge.

**Proposition 3.2.** Given a bottleneck assignment $M \in S(G_B)$, if an edge $e_c = \{a_c, b_c\}$ is a critical edge of $G_B$ relative to $M$, then the path $P = \{e_c\}$ is the unique alternating path in the pruned edge set $\phi(G_B, M)$ relative to $M$ between $a_c$ and $b_c$.

**Proof.** Assume edge $e_c$ is a critical edge of $G_B$ relative to $M$. Path $P = \{e_c\}$ is trivially an alternating path relative to $M$ and edge $e_c \in M$, therefore $P \subseteq \phi(G_B, M)$. It remains to show that there does not exist another alternating path $P' \subseteq \phi(G_B, M)$ relative to $M$ between $a_c$ and $b_c$. To this end, assume for contradiction that aside from the path $P = \{e_c\}$ there exists another alternating path $P'$ relative to $M$ between $a_c$ and $b_c$, where $P' \neq P$ and $P' \subseteq \phi(G_B, M)$. From Definition 2.3, a path is constructed from a sequence of distinct vertices, and $a_c$ and $b_c$ are the first and last vertices of the sequence so $P'$ cannot contain $e_c$ without repeating $a_c$ or $b_c$ in the sequence. Since $P' \neq P$, it is guaranteed that $e_c \notin P'$ and $P' \subseteq \phi(G_B, M) \setminus \{e_c\}$. It follows
that $\mathcal{P}'$ is an augmenting path relative to $\mathcal{M}\setminus\{e_c\}$. This contradicts the assumption that $e_c$ is a critical edge of $G_B$ relative to $\mathcal{M}$.

In the following section, we introduce concepts that are applied in Chapter 5 to solve the LexBAP. Furthermore, we show how these concepts are related to the concepts of a pruned edge set and a critical edge that are used for the BAP.

### 3.4 Structure for Solving the LexBAP

For the BAP, we are only concerned with one edge in an MCM, i.e., the edge in the MCM with largest weight. For the LexBAP, we are also concerned with the remaining edges within the MCM as they also determine the lexicographic order of an MCM, not just the one with largest weight. The tools in this section relate to exploring structure of the BAP and LexBAP for efficiently determining these remaining edges. Once again, the following concepts are defined using a weighted bipartite graph $G_B = (V_B, E_B)$, as introduced in Section 2.1.3.

A matching-sublevel set is related to a pruned edge set, both of which are tools that allow partitioning of the edge set $E_B$ according to the edge with largest weight in an MCM. Given an example graph, Figure 3.1 shows the differences between these two tools.

**Definition 3.3 (Matching-sublevel set).** Given an MCM $\mathcal{M}$ of $G_B$, a matching-sublevel set of $G_B$ relative to $\mathcal{M}$ is $\psi(G_B, \mathcal{M}) := \{e \in E_B | W(e) \leq \max_{e' \in \mathcal{M}} W(e')\}$.

We introduce the notion of the price of absence of an edge. The price of absence is a concept that we apply in Chapter 5 to determine if an edge belongs to a an MCM that is a solution of the LexBAP. To this end let $\kappa(G_B)$ be the cardinality of an MCM of $G_B$, i.e., given any MCM $\mathcal{M} \in X(G_B)$, $\kappa(G_B) = |\mathcal{M}|$. The price of absence of an edge is defined as follows.

**Definition 3.4 (Price of Absence).** Given an edge $e \in E_B$ such that $\kappa((V_B, E_B)) = \kappa((V_B, E_B\setminus\{e\}))$, the price of absence of an edge $e$ is

$$\zeta(G_B, e) := Y((V_B, E_B\setminus\{e\})) - Y((V_B, E_B)),$$

where $Y(\cdot)$ is the bottleneck weight as introduced in Definition 2.13. If the removal of an edge $e \in E_B$ changes the cardinality of an MCM, i.e., $\kappa((V_B, E_B)) \neq \kappa((V_B, E_B\setminus\{e\}))$, then the price of absence of $e$ is defined to be $+\infty$. 

(a) An example weighted bipartite graph $G_B = (V_B, E_B)$.

(b) Subgraph $(V_B, \phi(G_B, \mathcal{M}))$.

(c) Subgraph $(V_B, \psi(G_B, \mathcal{M}))$.

Figure 3.1: Comparison of a pruned edge set and a matching-sublevel set for an example graph. The circles represent vertices and the lines represent edges. White vertices represent agents $A$ and black nodes represent tasks $B$. The solid lines represent an MCM $\mathcal{M}$, the edge highlighted in red is an edge with largest weight in $\mathcal{M}$.

The price of absence is always non-negative. It is a measure of how much the bottleneck weight of a graph increases in the absence of a given edge. The so-called robustness margin first seen in $[47]$ is a special case of the price of absence. Its value can be used to quantify the sensitivity of an assignment solution to perturbations of edge weights as first studied for the SAP in $[34]$ and for the BAP in $[95]$. This type of sensitivity information is exploited in Chapter 6 to guarantee collision avoidance of mobile agents that are assigned to different destinations.
3.4.1 Properties of an Edge with Positive Price of Absence

To explore the concept of an edge with positive price of absence, we first observe that the price of absence is related to a critical edge in the following proposition.

**Proposition 3.3.** Given a bottleneck assignment \( M \in \mathcal{S}(G_B) \) and a bottleneck edge \( e_b \in M \) of \( G_B \), if \( e_b \) has a positive price of absence, then it is a critical edge of \( G_B \) relative to \( M \).

**Proof.** Assume bottleneck edge \( e_b \) is not a critical edge of \( G_B \) relative to \( M \). By Definition 3.2, \( \phi(G_B,M) \setminus \{e_b\} \) contains an augmenting path \( P \) relative to \( M \setminus \{e_b\} \). By Berge’s Theorem [94], \( M' := M \setminus \{e_b\} \oplus P \) is an MCM of \( G_B \), where the operator \( \oplus \) denotes symmetric difference. The weight of all edges in \( M' \) must be smaller than or equal to \( W(e_b) \) because \( M' \subseteq \phi(G_B,M) \setminus \{e_b\} \). This holds as \( M' \) is formed from the symmetric difference of \( M \setminus \{e_b\} \subseteq \phi(G_B,M) \setminus \{e_b\} \) and \( P \subseteq \phi(G_B,M) \setminus \{e_b\} \). However, \( \phi(G_B,M) \setminus \{e_b\} \subseteq \mathcal{E}_B \setminus \{e_b\} \) implies that \( \mathcal{E}_B \setminus \{e_b\} \) must also contain the MCM \( M' \), i.e., \( W((\mathcal{V}_B,\mathcal{E}_B \setminus \{e_b\})) \neq W((\mathcal{V}_B,\mathcal{E}_B)) \).

The following proposition establishes that an edge with positive price of absence appears in all solutions to the BAP. This is a key property of an edge with positive price of absence.

**Proposition 3.4.** If an edge \( e_p \in \mathcal{E}_B \) has a positive price of absence, then \( e_p \) is an element of every bottleneck assignment of \( G_B \), i.e., for all \( M \in \mathcal{S}(G_B), e_p \in M \).

**Proof.** Consider an arbitrary edge \( e_p \in \mathcal{E}_B \) with positive price of absence, i.e., \( \zeta(G_B,e_p) = Y((\mathcal{V}_B,\mathcal{E}_B \setminus \{e_p\})) - Y((\mathcal{V}_B,\mathcal{E}_B)) > 0 \). Assume for the sake of contradiction that there exists an MCM \( M \in \mathcal{S}(G_B) \) such that \( e_p \notin M \). However, this implies that \( Y((\mathcal{V}_B,\mathcal{E}_B \setminus \{e_p\})) = Y((\mathcal{V}_B,\mathcal{E}_B)) \), which contradicts the assumption that \( \zeta(G_B,e_p) > 0 \).

The following corollary follows directly from Proposition 3.4.

**Corollary 3.2.** Given a bottleneck assignment \( M \in \mathcal{S}(G_B) \), the set of edges with positive price of absence \( \mathcal{E}_{\zeta^+} := \{ e \in M | \zeta(G_B,e) > 0 \} \) is a subset of every bottleneck assignment of \( G_B \), i.e., for all \( M' \in \mathcal{S}(G_B), \mathcal{E}_{\zeta^+} \subseteq M' \).

Corollary 3.2 establishes the motivation for introducing edges with positive price, as these edges must appear in all bottleneck assignments. Therefore, it is desirable to find the set of
edges $E^+_{e^+}$ in an efficient way. The following theorem provides a property of an edge $e_p$ that can be exploited to identify whether $e_p$ has positive price of absence.

**Theorem 3.1.** Given a bottleneck assignment $M \in S(G_B)$, an edge $e_p \in E_B$ has positive price of absence if and only if there does not exist an augmenting path in $\psi(G_B, M) \setminus \{e_p\}$ relative to $M \setminus \{e_p\}$.

**Proof.** First, we prove necessity. Assume $e_p$ has positive price of absence. Assume for the sake of contradiction that there exists an augmenting path $P' \subseteq \psi(G_B, M) \setminus \{e_p\}$ relative to $M \setminus \{e_p\}$. By Berge’s Theorem [94], $M' = P' \oplus M \setminus \{e_p\}$ is another MCM of $G_B$, where the operator $\oplus$ denotes symmetric difference. However, $M'$ does not contain $e_p$ because it is the symmetric difference of two sets of edges that both do not contain $e_p$. This contradicts Proposition 3.4 because $e_p$ is an edge with positive price of absence and all bottleneck assignments of $G_B$ must contain $e_p$.

Next, we prove sufficiency. Assume there does not exist an augmenting path $P' \subseteq \psi(G_B, M) \setminus \{e_p\}$ relative to $M \setminus \{e_p\}$. Then, it is not possible to construct an MCM of $G_B$ from $\psi(G_B, M) \setminus \{e_p\}$. In other words, without edge $e_p$, an MCM of $G_B$ must contain at least one edge with weight larger than the weights of the edges in $M$. Thus, $e_p$ has positive price of absence.

The following remark is an observation of the significant role that augmenting paths play in the BAP and LexBAP. It also highlights the similarities between a critical edge and an edge with positive price of absence.

**Remark 3.1.** Corollary 3.1 connects the non-existence of an augmenting path within a pruned edge set to finding a bottleneck edge. Theorem 3.1 connects the non-existence of an augmenting path within a matching-sublevel set to finding an edge with positive price of absence.

The following corollary follows from Theorem 3.1 and further illustrates the above properties of an edge with positive price of absence and its role within the structure of a matching-sublevel set.

**Corollary 3.3.** Given a bottleneck assignment $M \in S(G_B)$, an edge $e_p = \{a_p, b_p\} \in E_B$ has positive price of absence if and only if the path $P = \{e_p\}$ is a unique alternating path in the matching-sublevel set $\psi(G_B, M)$ relative to $M$ between $a_p$ and $b_p$. 

Fig. 3.2 illustrates the statements in Proposition 3.2 and Corollary 3.3 with a simple example. In this example, the edge \( \{a_1, b_1\} \) is a critical edge. There exists only one alternating path between \( a_1 \) and \( b_1 \) in the pruned edge set \( \phi(G_B, M) \), i.e., the path \( P = \{\{a_1, b_1\}\} \). If this edge were removed from the pruned edge set, there would not exist an augmenting path between \( a_1 \) and \( b_1 \). No MCM of \( G_B \) can be constructed using edges contained in \( \phi(G_B, M) \setminus \{\{a_1, b_1\}\} \). On the other hand, the edge \( \{a_1, b_1\} \) does not have positive price of absence. There exist two alternating paths between \( a_1 \) and \( b_1 \) in the matching-sublevel set \( \psi(G_B, M) \). If this edge were removed from the matching-sublevel set, there would exist an augmenting path between \( a_1 \) and \( b_1 \). In fact, another MCM \( M' = \{\{a_1, b_3\}, \{a_3, b_1\}, \{a_2, b_2\}\} \) of \( G_B \) exists within the matching-sublevel set, which confirms the statement that the price of absence of \( \{a_1, b_1\} \) is zero and not positive.

![Illustration of Proposition 3.2 and Corollary 3.3](image-url)
Up to this point, we discussed concepts aimed at developing algorithms for solving the BAP and LexBAP. For the remainder this chapter, we focus on extending the analysis of the structure of the BAP that is hinted at in Figure 3.2. This representation of the bipartite graph that centres around the critical edge and pruned edge set leads to a novel way of clustering a team of agents. We discuss properties of the BAP that result from this structure in the following section.

3.5 Clustering Agents

We introduce with the following definitions related to clustering agents. Then, using the concept of partitioning a graph according to a pruned edge set, we explore the vital role of augmenting paths in the BAP by considering the problem of merging two BAPs. There is existing literature on clustering agents and forming teams of agents that coordinate to carry out complex tasks, e.g., [96, 97]. A bottleneck cluster, defined below, is used to derive conditions for efficiently solving two separate BAPs as a combined problem. In the context of team formation, it provides a novel way to form a team of agents based on knowledge of the bottleneck edge. Once again, the following concepts are defined using a weighted bipartite graph $G_B = (V_B, E_B)$, as introduced in Section 2.1.3.

Definition 3.5 (Bottleneck cluster). Consider a bottleneck assignment $\mathcal{M} \in S(G_B)$ and a bottleneck edge $e \in \mathcal{M}$ of $G_B$. Graph $G_B$ is a bottleneck cluster relative to $e$ if and only if for any vertex $v \in V_B$, there exists an alternating path $P$ relative to $\mathcal{M}$ between $v$ and a vertex $v' \in e$ such that $P \subseteq \phi(G_B, \mathcal{M})$.

Definition 3.6 (Alternating tree). Given a matching $\mathcal{M}$ of $G_B$, $G_B$ is an alternating tree relative to $\mathcal{M}$ if and only if $G_B$ is a tree with root vertex $v_r \in V_B$, and for all vertices $v \in V_B \setminus \{v_r\}$, the path between $v$ and $v_r$ is an alternating path relative to $\mathcal{M}$.

We construct two particular subgraphs of a bottleneck cluster by partitioning its vertices and edges, i.e., we partition the bottleneck cluster so that there is no intersection between the vertices in the agent tree and the vertices in the task tree.

Definition 3.7 (Agent and task trees). Consider a bottleneck assignment $\mathcal{M} \in S(G_B)$ and a critical edge $e_c = \{a_c, b_c\}$ of $G_B$ relative to $\mathcal{M}$. Let $G_B$ be a bottleneck cluster with respect to $e_c$. The agent and task trees of $G_B$ are $\theta_A(G_B) := (V_A, E_A)$ and $\theta_B(G_B) := (V_B, E_B)$ respectively, where $\theta_A(G_B)$ and $\theta_B(G_B)$ are alternating trees with the following properties. The sets of edges
satisfy $\mathcal{E}_A \cup \mathcal{E}_B \subseteq \phi(G_B, \mathcal{M})$, $\mathcal{E}_A \cap \mathcal{E}_B = \emptyset$, the sets of vertices satisfy $\mathcal{V}_A \cup \mathcal{V}_B = \mathcal{V}_B$, $\mathcal{V}_A \cap \mathcal{V}_B = \emptyset$, $\mathcal{V}_A$ contains the bottleneck agent $a_c = e_c \cap \mathcal{A}$, and $\mathcal{V}_B$ contains the bottleneck task $b_c = e_c \cap \mathcal{B}$.

![Figure 3.3](image)

Figure 3.3: Illustration of a bottleneck cluster with respect to a critical edge. The dotted lines represent edges not in the matching $\mathcal{M}$, the solid lines represent edges in $\mathcal{M}$. The set of agents is $\{a_1, a_2, ..., a_7\}$ and the set of tasks is $\{b_1, b_2, ..., b_7\}$. The edge $e_c = \{a_1, b_1\}$ is a critical edge. Only edges in $\phi(G_B, \mathcal{M})$ are displayed on the figure.

Figure 3.3 illustrates the agent and task trees for $G_B$ and a bottleneck assignment $\mathcal{M}$. In this figure, the task tree $\theta_B(G_B)$ is on the left and has root $b_1$ and the agent tree $\theta_A(G_B)$ is on the right and has root $a_1$.

When the particular bottleneck edge of a bottleneck cluster is also a critical edge, we observe the emergence of the following structural properties in the BAP, i.e., we discuss structural properties of the BAP when combining the concepts of a critical edge, and the agent and task trees of a bottleneck cluster. The following corollary describes the structure of a bottleneck cluster $G_B$ relative to a critical edge $e_c$ based on Proposition 3.2. Given a bottleneck cluster $G_B$ relative to a critical edge $e_c = \{a_c, b_c\}$, the agent and task trees of $G_B$ can only be connected by edges that preserve the property that $e_c$ is the unique alternating path between $a_c$ and $b_c$.

**Corollary 3.4.** Consider a bottleneck assignment $\mathcal{M} \in S(G_B)$ and a critical edge $e_c = \{a_c, b_c\}$ of $G_B$ relative to $\mathcal{M}$. Let $G_B$ be a bottleneck cluster with respect to $e_c$. For all agents $a \in \mathcal{V}_A \cap \mathcal{A}$ in the agent tree $\theta_A(G_B)$ and for all tasks $b \in \mathcal{V}_B \cap \mathcal{B}$ in the task tree $\theta_B(G_B)$, we have that $\{a, b\} \notin \phi(G_B, \mathcal{M})$.

Corollary 3.4 states that due to the structure imposed by the bottleneck cluster and critical edge, particular agent and task vertices are never connected by an edge belonging to $\phi(G_B, \mathcal{M})$. Figure 3.3 illustrates this corollary; all edges displayed in the figure belong to $\phi(G_B, \mathcal{M})$ and there does not exist an edge between a task vertex $b \in \mathcal{V}_B \cap \mathcal{B}$ and an agent vertex $a \in \mathcal{V}_A \cap \mathcal{A}$.
3.5.1 Investigating the Role of Augmenting Paths in the BAP

We explore structure of the BAP by considering the problem of merging two separate BAPs into one combined BAP. Partitioning, clustering and merging graphs is an idea used in data processing, e.g., the authors in [98] consider data clustering by partitioning a bipartite graph while the authors in [99] consider the problem of merging cluster ensembles. We instead apply the problem of merging BAPs to highlight the integral role of an augmenting path in the BAP. Although in this thesis we consider two subproblems, we note that the results can be generalised for merging multiple subproblems. We formalise the pedagogical example in Problem 3.1 below.

**Problem 3.1.** Consider two sets of agents $A_1 = \{a_1, a_2, ..., a_m\}$ and $A_2 = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ and two sets of tasks $B_1 = \{b_1, b_2, ..., b_n\}$ and $B_2 = \{\beta_1, \beta_2, ..., \beta_n\}$. Define the sets $A_3 = A_1 \cup A_2$ and $B_3 = B_1 \cup B_2$. Assume $m_3 = m_1 + m_2$ and $n_3 = n_1 + n_2$ and that $m_1 \geq n_1$ and $m_2 \geq n_2$.

For all $k \in \{1, 2, 3\}$, define graph $G_k = (V_k, E_k)$, with vertex set $V_k = A_k \cup B_k$ and edge set $E_k = \{(i, j) | i \in A_k, j \in B_k\}$. Define the function $W : E_3 \rightarrow \mathbb{R}$ that maps all edges to their weights. For $k \in \{1, 2\}$, assume we are given a bottleneck assignment $M_k \in S(G_k)$ and a bottleneck edge $e_k \in M_k$ of graph $G_k$. Find a bottleneck assignment of $G_3$, i.e., find an MCM $M_3 \in S(G_3)$.

We first introduce an upper bound on the weight of a bottleneck edge of $G_3$, in terms of the bottleneck edges $e_1$ and $e_2$ of $G_1$ and $G_2$. Then in Section 3.5.1.2, we discuss bottleneck clusters and provide conditions when the solution to Problem 3.1 is found by merging the matchings $M_1$ and $M_2$.

3.5.1.1 A Bound on the BAP Solution

We derive an upper bound on the weight of a bottleneck edge of $G_3$ that depends on the weights of the bottleneck edges $e_1$ and $e_2$ of $G_1$ and $G_2$.

**Lemma 3.3.** Consider the definitions given in Problem 3.1. Given any bottleneck edge $e_3$ of graph $G_3$, the weight $W(e_3) \leq \max\{W(e_1), W(e_2)\}$.

**Proof.** By definition, $W(e_3) = \max_{e \in M_3} W(e) \leq \max_{e \in M} W(e)$ for any arbitrary MCM $M \in X(G_3)$. Since $M' = M_1 \cup M_2 \in X(G_3)$, i.e., $M'$ is an MCM of $G_3$, we have $W(e_3) \leq \max_{e \in M'} W(e) = \max\{W(e_1), W(e_2)\}$. □
The set \( M_1 \cup M_2 \) is an MCM of \( G_3 \). This MCM is not necessarily a bottleneck assignment of \( G_3 \). However, the largest weight of any edge in \( M_1 \) and \( M_2 \) provides an upper bound on the bottleneck weight of \( G_3 \) according to Lemma 3.3. Lemma 3.3 demonstrates structure of the BAP that allows the concept of warm-starting, i.e., exploiting this lemma in a BAP algorithm allows prior information to be used to solve the BAP if it is available rather than solving from scratch. We explore this idea further in Chapter 4 when a particular BAP algorithm is introduced.

3.5.1.2 Conditions for Merging Two BAPs Efficiently

We apply properties of a bottleneck cluster, a critical edge, and the agent and task trees of a bottleneck cluster to provide conditions for merging two BAPs to obtain a solution to the full problem. From Lemma 3.3, the weights of the bottleneck edges of \( G_1 \) and \( G_2 \) provide an upper bound on the weight of the bottleneck edge of \( G_3 \). Therefore, \( M_3 = M_1 \cup M_2 \) is not necessarily a bottleneck assignment of \( G_3 \). We discuss conditions for determining if \( M_3 \) is a bottleneck assignment of \( G_3 \) when \( G_1 \) and \( G_2 \) are both bottleneck clusters.

**Lemma 3.4.** Consider the definitions given in Problem 3.1. Let \( G_1 \) and \( G_2 \) be bottleneck clusters with respect to \( e_1 \) and \( e_2 \), respectively. Let \( e_1 \) be a critical edge of \( G_1 \) relative to \( M_1 \) and \( e_2 \) be a critical edge of \( G_2 \) relative to \( M_2 \). Moreover assume \( W(e_1) \geq W(e_2) \). If \( W(e_3) < W(e_1) \), then the following conditions all hold:

i. there exists an edge in \( E_3 \) with weight less than \( W(e_1) \) between an agent \( a' \in A_2 \) and a task \( b \in V_{B_1} \cap B_1 \), where \( V_{B_1} \) is the vertex set of the task tree \( \theta_B(G_1) = (V_{B_1}, E_{B_1}) \) of \( G_1 \);

ii. there exists an edge in \( E_3 \) with weight less than \( W(e_1) \) between a task \( b' \in B_2 \) and an agent \( a \in V_{A_1} \cap A_1 \), where \( V_{A_1} \) is the vertex set of the agent tree \( \theta_A(G_1) = (V_{A_1}, E_{A_1}) \) of \( G_1 \);

iii. there exists an alternating path \( P \) between \( a' \) and \( b' \) containing only edges with weight less than \( W(e_1) \), and \( |P \cap M_2| > |P \setminus M_2| \).

**Proof.** Without loss of generality, let \( e_1 = \{a_1, b_1\} \). By Proposition 3.2, \( e_1 \) is the only alternating path between \( a_1 \) and \( b_1 \) in edge set \( \phi(G_1, M_1) \). Assume there does not exist vertices \( a' \) and \( b' \) such that all i., ii., and iii. are true. Thus, \( e_1 \) is the only alternating path between \( a_1 \) and \( b_1 \) in \( \phi(G_3, M_1 \cup M_2) \). By Definition 3.2, \( e_1 \) is also a critical edge of \( G_3 \), since \( e_1 \in \arg \max_{e \in M_1 \cup M_2} W(e) \) and there does not exist an augmenting path in \( \phi(G_3, M_1 \cup M_2) \setminus \{e_1\} \) relative to \( (M_1 \cup M_2) \setminus \{e_1\} \). Thus, \( W(e_3) = W(e_1) \). \( \square \)
Taking the contrapositive of Lemma 3.4, if the conditions i., ii., and iii. do not all hold, then \( W(e_3) = W(e_1) \) because \( W(e_3) \) can only be less than or equal to \( W(e_1) \) by Lemma 3.3. Therefore, if the conditions i., ii., and iii. do not all hold, then \( M_3 = M_1 \cup M_2 \) is a bottleneck assignment of \( G_3 \). Lemma 3.4 argues that for \( M_3 \) to be a bottleneck assignment of \( G_3 \), \( e_1 \) must remain as the unique alternating path described in Proposition 3.2.

In general, the converse of Lemma 3.4 does not hold when either \( W(e_1) = W(e_2) \), or \( M_1 \) contains more than one edge with weight equal to \( W(e_1) \). We include these assumptions in the following theorem.

**Theorem 3.2.** *Adopt the hypothesis of Lemma 3.4. Moreover, assume \( W(e_1) > W(e_2) \) and \( \arg \max_{e \in M_1} W(e) \) is a singleton. It holds that \( W(e_3) < W(e_1) \) if and only if conditions i., ii., and iii. from Lemma 3.4 hold.*

**Proof.** The necessary condition for \( W(e_3) < W(e_1) \) holds from Lemma 3.4. We now prove the sufficient condition. Assume conditions i., ii., and iii. hold. Then, aside from path \( P = \{e_1\} \) containing a single edge \( e_1 = \{a_1, b_1\} \), there exists another alternating path \( P' \) between \( a_1 \) and \( b_1 \), which does not contain the edge \( e_1 \). Namely, the alternating path \( P' \) constructed from the union of the alternating paths between \( a_1 \) and \( a \), \( a \) and \( b' \), \( b' \) and \( a' \), \( a' \) and \( b \), and \( b \) and \( b_1 \). Thus, there exists an augmenting path \( P' \subseteq \phi(G_3, M_1 \cup M_2) \setminus \{e_1\} \) relative to \( (M_1 \cup M_2) \setminus \{e_1\} \). From Proposition 3.1, there exists an MCM \( M' \) of \( G_3 \) such that \( M' \subseteq \phi(G_3, M_1 \cup M_2) \setminus \{e_1\} \). By the assumptions on \( e_1 \), \( \phi(G_3, M_1 \cup M_2) \setminus \{e_1\} \) contains only edges with weights strictly smaller than \( W(e_1) \). Thus, there exists an MCM of \( G_3 \) with all edges having weight smaller than \( W(e_1) \), therefore, \( W(e_3) \) must be smaller than \( W(e_1) \). \( \square \)

Figure 3.4 illustrates a BAP where the conditions provided in Theorem 3.2 are satisfied. The orange dashed line is an edge that satisfies condition i. since \( \alpha_2 \in V_2 \) and there exists edge \( \{\alpha_2, b_3\} \in V_3 \), where \( b_3 \) is a task in the vertex set of \( \theta_B(G_1) \). The blue dashed line satisfies condition ii. since \( \beta_1 \in V_2 \) and there exists edge \( \{\beta_1, a_2\} \in V_3 \), where \( a_2 \) is an agent in the vertex set of \( \theta_A(G_1) \). Condition iii. is satisfied since there is an alternating path \( P = \{\{\alpha_1, \beta_1\}, \{\alpha_2, \beta_2\}, \{\alpha_1, \beta_2\}\} \) between \( \beta_1 \) and \( \alpha_2 \), and \( |\{\alpha_1, \beta_1\}| + |\{\alpha_2, \beta_2\}| > |\{\alpha_1, \beta_2\}| \), i.e., \( P \) starts with a solid line and ends with a solid line.

We summarise the role that an augmenting path plays in the BAP with the following corollary. Corollary 3.5 follows from Lemma 3.3 and Theorem 3.2.
Corollary 3.5. If one or more of conditions i., ii., or iii. in Theorem 3.2 do not hold, then $M_3 = M_1 \cup M_2$ is a solution to Problem 3.1.

If the merging of two BAPs does not result in the existence of an augmenting path within the set $\phi(G_3, M_1 \cup M_2) \setminus \{e_1\}$, then the solution for the merged problem is the union of the bottleneck assignments of the two separate problems.

3.6 Conclusion

We introduced tools, notably a pruned edge set, a critical edge and a bottleneck cluster, to analyse structure of the BAP. We also introduced the concept of price of absence and a matching-sublevel for solving the LexBAP. We illustrated the critical role that an augmenting path plays in the BAP by exploring a situation where two BAPs are merged into a combined BAP. In that situation, the conditions on whether or not reassignment is necessary are related to the existence of a particular augmenting path between the two bottleneck clusters. In the following chapters, we rely heavily on the concepts discussed in this chapter.
Chapter 4

A Distributable Approach for Solving the Bottleneck Assignment Problem

4.1 Introduction

We apply the concepts discussed in the previous chapter to develop an algorithm to solve the BAP that is amenable to distributed computation. The problem of solving the BAP in a distributed manner is formally stated as follows.

Problem 4.1. Given Assumptions 2.1 and 2.2, obtain a solution to the BAP given by (2.1).

We first leverage the concepts of a pruned edge set and a critical edge from the previous chapter to introduce the algorithm in its centralised form before showing how each component can be carried out with distributed computation. In the previous chapter, we discussed the importance of augmenting paths in the BAP. In fact, the challenging part of developing this distributed algorithm lies in searching for augmenting paths when information and computation is distributed over agents. Therefore, we focus on one component of the BAP algorithm in particular, i.e., the component related to the distributed search for augmenting paths and compare two methods of conducting such a search. In order to do this, we make use of alternating trees to represent the searches and discuss their differences. Furthermore, we explore the warm-starting property of the algorithm, which allows the exploitation of prior information when converging to a solution of the BAP. Then, a numerical analysis of the BAP algorithm is presented to compare the empirical complexities of the two different distributed augmenting path searches and benchmark the BAP algorithm as a whole against an existing distributed greedy algorithm.
4.2 Centralised PruneBAP

We first present PruneBAP for solving the BAP. Then, we discuss properties of this algorithm, namely how each subroutine of PruneBAP can be implemented in the distributed setting and how the algorithm is amenable to warm-starting. To initialise PruneBAP, we require an arbitrary MCM \( M_0 \in X(G_B) \) of \( G_B \), e.g., agents and tasks can be initially matched based on arbitrary indexing, \( M_0 = \\{\{a_p, b_q\}| a_p \in A, b_q \in B, p = q\} \). We do not require \( G_B \) to be complete as it is always possible if necessary to transform an incomplete bipartite graph into complete one for the purposes of PruneBAP by assuming any missing edges have weight of infinity.

Algorithm PruneBAP

Input: Graph \( G_B = (V_B, E_B) \), and an MCM \( M_0 \).
Output: An MCM \( M \) of \( G_B \) that is solution to the BAP.

1: \( M \leftarrow M_0 \)
2: matching_exists \leftarrow \text{True}
3: while matching_exists do
4: \( (\bar{e}, W(\bar{e})) \leftarrow \text{MaxEdge}(M) \) \hspace{1cm} \triangleright \text{Find edge with largest weight}
5: \( \bar{E} \leftarrow \phi(G_B, M) \setminus \{\bar{e}\} \) \hspace{1cm} \triangleright \text{Shrink } \bar{E} \text{ by pruning edges}
6: \( \bar{M} \leftarrow M \setminus \{\bar{e}\} \)
7: \( M_\nu \leftarrow \text{AugPath}(\bar{e}, \bar{M}, (V_B, \bar{E})) \) \hspace{1cm} \triangleright \text{Check condition for critical edge}
8: if \( M_\nu \neq \bar{M} \) then
9: \( M \leftarrow M_\nu \) \hspace{1cm} \triangleright \text{New MCM within } \bar{E} \text{ identified}
10: else
11: matching_exists \leftarrow \text{False} \hspace{1cm} \triangleright \text{ } \bar{e} \text{ is a critical edge}
12: end if
13: end while
14: return \( M \)

In Section 4.3.1 and Section 4.3.3, we verify that the following two assumptions hold and the functions can be implemented in a distributed setting.

Assumption 4.1. Given an MCM \( M \) of \( G_B \), assume that there exists a function \( \text{MaxEdge}(M) \) that returns the tuple \( (\bar{e}, W(\bar{e})) \), where edge \( \bar{e} \in \arg \max_{e \in M} W(e) \).

Assumption 4.2. Given a graph \( (V_B, E') \), an edge \( \bar{e} \in E' \), an MCM \( M \), and a matching \( \bar{M} = M \setminus \{\bar{e}\} \), assume there exists function \( \text{AugPath}(\bar{e}, \bar{M}, (V_B, E' \setminus \{\bar{e}\})) \) that checks if there exists an augmenting path \( P \) relative to \( \bar{M} \) in \( E' \setminus \{\bar{e}\} \). If \( P \) exists, the function returns an MCM \( M_\nu = \bar{M} \oplus P \). If \( P \) does not exist, the function returns \( M_\nu = \bar{M} \).
We prove that PruneBAP converges to a solution of the BAP as long as the above assumptions hold. As mentioned in Chapter 3, Proposition 3.1 is vital for proving the convergence of PruneBAP as it provides the criteria for finding a critical edge, which in turn allows us to find a bottleneck assignment.

**Proposition 4.1.** If Assumptions 4.1 and 4.2 hold, then PruneBAP returns a bottleneck assignment of $G_B$.

*Proof.* Given Assumptions 4.1 and 4.2, we observe that Lines 5, 6, and 7 of PruneBAP test the sufficient and necessary conditions for the existence of the MCM $M'$ based on Proposition 3.1, with $e = \bar{e}$ and $\mathcal{E}' = \phi(G_B, M)$.

Now, let $k \in \{1, 2, \ldots, f\}$ denote the iterations of the while-loop of PruneBAP, where without loss of generality, $f$ is the final iteration. Let $\bar{E}_k$ denote the set $\bar{E}$ in Line 5 at iteration $k$ of the while-loop. By Proposition 3.1, we have that for all iterations $k < f$, there exists an MCM $M_k$ of $G_B$ such that $M_k \subseteq \bar{E}_k$. At the final iteration $f$, an augmenting path is not found so PruneBAP returns $M = M_{f-1}$. It remains to show that the MCM $M_{f-1} \subseteq \bar{E}_{f-1}$ is a bottleneck assignment of $G_B$. Assume for contradiction there exists an MCM $M_f$ of $G_B$ with all edges having weights strictly less than the weight of edge $e = \arg \max_{e' \in M_{f-1}} W(e')$. This implies that there exists an MCM $M_f$ of $G_B$ such that $M_f \subseteq \phi(G_B, M_{f-1}) \setminus \{e\} = \bar{E}_f$, which contradicts the assumption that $f$ is the final iteration. Following the above arguments, $e$ is a critical edge of $G_B$ and consequently $M_{f-1}$ is a bottleneck assignment. \(\square\)

### 4.2.1 Demonstration of PruneBAP

We summarise PruneBAP in three steps. The algorithm is initialised with an arbitrary initial MCM $M_0$. Then, the following steps are repeated until a solution to the BAP is found.

**Step 1** Find the pruned edge set $\bar{E} = \phi(G_B, M)$, i.e., prune the edge set $E_B$ of $G_B$ by removing edges whose weights are too large.

**Step 2** Remove one edge $e_l$ with largest weight from $\phi(G_B, M)$. Based on Definition 3.1, this edge must also be an element of the current MCM $M$. 
Step 3 Find a new MCM of $G_B$ from the remaining edges in $\phi(G_B, M) \setminus \{e_l\}$, by making use of an augmenting path and $M \setminus \{e_l\}$. See Proposition 3.1 for the connection between the existence of an augmenting path and the existence of this new MCM.

When Step 3 does not yield an MCM, the algorithm terminates and the last valid MCM is a solution to the BAP. Fig. 4.1 is an illustration of this algorithm.

Iteration 1:

\[
\begin{align*}
& e_{24} & e_{42} & e_{43} & e_{12} & e_{21} & e_{13} & e_{22} & e_{33} & e_{23} & e_{14} & e_{31} & e_{11} & e_{41} & e_{32} & e_{44} \\
\end{align*}
\]

Iteration 2:

\[
\begin{align*}
& e_{24} & e_{42} & e_{43} & e_{12} & e_{21} & e_{13} & e_{22} & e_{33} & e_{23} & e_{14} & e_{31} & e_{11} & e_{41} & e_{32} & e_{44} \\
\end{align*}
\]

Iteration 3:

\[
\begin{align*}
& e_{24} & e_{42} & e_{43} & e_{12} & e_{21} & e_{13} & e_{22} & e_{33} & e_{23} & e_{14} & e_{31} & e_{11} & e_{41} & e_{32} & e_{44} \\
\end{align*}
\]

Figure 4.1: Demonstration of PruneBAP with four agents $A = \{a_1, a_2, a_3, a_4\}$ and four tasks $B = \{b_1, b_2, b_3, b_4\}$. The set of edges $E_B$ are arranged in order of ascending weight, where $e_{pq}$ is the edge between agent $a_p$ and task $b_q$. At Iteration 1, an initial arbitrary MCM is chosen, denoted by the four circled edges. Edges to the right of the dashed line have been pruned from $E_B$ and edges to the left form the set $\phi(G_B, M)$. Note, $W(e_{44}) \geq W(e_{11}) \geq W(e_{21})$, i.e., with each iteration the weight of the largest edge in the current MCM is non-increasing. The algorithm terminates when it is not possible to form a matching of cardinality 4 from the remaining edges to the left of the dashed line.

### 4.3 Distributed PruneBAP

We now discuss how the individual components of PruneBAP can be implemented with the distributed setting outlined in Assumptions 2.1 and 2.2 and Remark 2.1. There are three main components to consider, i.e., finding the largest edge within a set of edges, removal of edges to form a pruned edge set, and searching for an augmenting path.
4.3.1 Distributed Maximum Consensus

We discuss a function that allows Assumption 4.1 to be satisfied in the distributed setting given by Assumptions 2.1 and 2.2. Given an edge set \( \hat{\mathcal{E}} \subseteq \mathcal{E}_B \), agents must find an edge

\[
\bar{e} = \arg \max_{e \in \hat{\mathcal{E}}} W(e),
\]

without conflict, i.e., agents reach consensus on the next edge to test as a critical edge. See Remark 4.1 about resolving ties should they arise. The problem in (4.1) is known as the max-consensus problem [100, 101].

**Lemma 4.1.** Given Assumptions 2.1 and 2.2, the max-consensus algorithm in [101, eq. (3)] solves the problem given by (4.1) and guarantees that Assumption 4.1 is satisfied.

*Proof.* The max-consensus algorithm updates the local estimate of the maximum weight for all agents \( a \in \mathcal{A} \) according to the law

\[
x_a(k) = \max_{a' \in N(G_C, a)} x_{a'}(k - 1),
\]

where \( k \) denotes the \( k \)th communication instance and \( x_a(0) = \max_{e \in \mathcal{E}_a} W(e) \). Agents \( a \in \mathcal{A} \) converge to

\[
x_a(D) = \max_{a' \in \mathcal{A}} \max_{e \in \mathcal{E}_{a'}} W(e) = \max_{e \in \hat{\mathcal{E}}} W(e)
\]

after \( D \) communication instances, where \( D \) is the diameter of \( G_C \). \( \square \)

**Remark 4.1.** Given an arbitrary set of edges \( \hat{\mathcal{E}} \subseteq \mathcal{E}_B \), let the set of edges with maximum weight be

\[
\pi = \arg \max_{e \in \hat{\mathcal{E}}} W(e).
\]

If \( \pi \) is not a singleton, a deterministic method is required for selecting one edge to satisfy Assumption 4.1. For instance, given indexed agents, the edge incident to the agent with the lower index number can be selected.

Note that to satisfy Assumption 4.1, both the edge \( \bar{e} \) and the corresponding weight \( W(\bar{e}) \) are required from the max-consensus algorithm. This corresponds to agents communicating and keeping track of the particular edge that is associated with their local estimate of the maximum weight throughout the max-consensus procedure.

4.3.2 Distributed Edge Removal

Given the tuple \((\bar{e}, W(\bar{e}))\) from Assumption 4.1 and given that each agent \( a \in \mathcal{A} \) has access to \( \mathcal{E}_a \) and \( \mu_a \) from Assumption 2.1 and Remark 2.1, each agent \( a \) can locally determine the set

\[
\tilde{\mathcal{E}}_a = \{a, \mu_a\} \cup \{e \in \mathcal{E}_a | W(e) < W(\bar{e})\}.
\]

Thus, we have a distributed representation of the pruned edge
set, since \( \phi(G_B, M) = \bigcup_{a \in A} \tilde{E}_a \). To represent \( \tilde{E} = \phi(G_B, M) \setminus \{ \tilde{e} \} \) it remains for the particular agent \( a^* = A \cap \tilde{e} \) to additionally prune \( \tilde{e} = \{ a^*, \mu_{a^*} \} \), i.e., \( \tilde{E}_{a^*} = \{ e \in \tilde{E}_{a^*} | W(e) < W(\tilde{e}) \} \).

Similarly edge \( \tilde{e} \) is locally removed from the matching \( M \) in Line 6 of PruneBAP. To complete the distributed edge removal according to Remark 2.1, agent \( a^* \) is labelled as free, i.e., \( \mu_{a^*} = \hat{b} \).

### 4.3.3 Distributed Augmenting Path Search

For the following analyses on distributed augmenting path searches, we make the following assumption on the number of agents and tasks.

**Assumption 4.3.** Assume that the bipartite graph \( G_B = (A \cup B, \tilde{E}_B) \) has cardinalities of agent and task sets satisfying \( |A| \geq |B| = n \), where \( n \) is the cardinality of any MCM of \( G_B \), i.e., the number of agents is greater than or equal to the number of tasks.

This assumption is not vital and Definition 4.1 and Proposition 4.2 below can be generalised for bipartite graphs that do not satisfy this rule, i.e., in the case where there are strictly fewer agents than tasks. However, this particular case does not add to the results in this thesis. Instead, we limit the discussion of the consequences when tasks outnumber agents to a brief analysis in Remark 4.2.

We present a general framework for an augmenting path search and show that regardless of the order in which agents are explored, this framework guarantees all agents with an alternating path to the root are explored.

**Definition 4.1** (Alternating search). Consider a subgraph \( (A \cup B, \tilde{E}) \), where \( \tilde{E} \subseteq \tilde{E}_B \) and an MCM of \( G_B \), \( \tilde{M} = \{ \{a_1, \mu_{a_1}\}, \{a_2, \mu_{a_2}\}, ..., \{a_n, \mu_{a_n}\} \} \), such that \( \tilde{M} \setminus \{ \{a_n, \mu_{a_n}\} \} \subseteq \tilde{E} \) and \( \{a_n, \mu_{a_n}\} \notin \tilde{E} \). An alternating search is defined using the following vertex and edge set constructions.

Consider a tree \( (V_1, E_1) \) consisting of only the free root task vertex \( \mu_{a_n} \in B \), i.e., \( V_1 = \{ \mu_{a_n} \} \) and \( E_1 = \emptyset \). For all \( k \in \{1, 2, ..., f - 1\} \), we construct vertex and edge sets \( V_{k+1} = V_k \cup \{i_k, \mu_{i_k}\} \) and \( E_{k+1} = E_k \cup \{ \{i_k, j_k\}, \{i_k, \mu_{i_k}\} \} \), satisfying the condition that agent \( i_k \in A \setminus V_k \) and task \( j_k \in B \cap V_k \) are neighbours in the graph \( (A \cup B, \tilde{E}) \), i.e., \( \{i_k, j_k\} \in \tilde{E} \). Let \( f \) be the iteration at which for all tasks \( j_f \in B \cap V_f \), there does not exist an agent \( i_f \in A \setminus V_f \) with \( \{i_f, j_f\} \in \tilde{E} \).
We observe that the alternating search results in the construction of an alternating tree from the explored vertices and edges. Furthermore, we highlight the connection between an alternating search and the conditions on merging two BAPs in Theorem 3.2, where the purpose of an alternating search is to test for the uniqueness of the alternating path described in Proposition 3.2.

**Proposition 4.2.** Given the tree \((V_f, E_f)\) constructed in Definition 4.1, the following statement holds. For any agent \(a \in A\), if there exists an alternating path between \(a\) and free task \(\mu_{an}\) containing only elements of \(\bar{E}\), then the agent is in the tree, i.e., \(a \in V_f\).

**Proof.** By construction, if task \(b \in B\) is in \(V_f\), then for all agents \(a \in A\) with \(\{a, b\} \in \bar{E}\), we have \(a \in V_f\). By contrapositive, if an agent \(a \in A\) with \(\{a, b\} \in \bar{E}\) is not in \(V_f\), then task \(b \in B\) is not in \(V_f\). Similarly, if an agent \(a \in A\) is in \(V_f\), then \(\mu_{an} \in V_f\), and by contrapositive if \(\mu_{an} \notin V_f\), then agent \(a \in A\) is not in \(V_f\).

Assume for contradiction that there exists \(a \in A\) such that there exists an alternating path \(P = P(\bar{E}, a, \mu_{an}) = \{v_k, v_{k+1}\} \in \bar{E} | k = 0, 1, \ldots, k, v_0 = a, v_{k+1} = \mu_{an}\) and that \(a \notin V_f\). We have that \(v_0 = a \notin V_f\). From the two implications above, if \(v_k \notin V_f\), then \(v_{k+1} \notin V_f\). By applying this to all edges in \(P\), we have that \(v_{k+1} = \mu_{an} \notin V_f\). This is a contradiction, as \(\mu_{an}\) is the root of the tree. Therefore, \(a\) must be in \(V_f\).

**Remark 4.2.** Without Assumption 4.3, the root vertex \(\mu_{an}\) in Definition 4.1 is not necessarily a unique free task vertex relative to \(\bar{M}\). The following is a discussion for bipartite graphs where \(|A| < |B|\). First, we assume agents have access to the set of free task vertices relative to \(\bar{M}\), which we denote as \(B_F\). Instead of constructing a single alternating tree in an alternating search, there exists multiple alternating trees \((V_1, E_1), (V_2, E_2), \ldots, (V_q, E_q)\), one for each free task vertex \(b_1, b_2, \ldots, b_q \in B_F\). The important condition for these trees is that \(V_k \cap V_l = \emptyset\), for all \(k, l \in \{1, 2, \ldots, q\}, k \neq l\), i.e., a vertex is explored at most one time. Consequently, a generalisation of Proposition 4.2 is the statement that for any agent \(a' \in A\), if there exists an alternating path between \(a'\) and any free task in \(b' \in B_F\) containing only elements of \(\bar{E}\), then the agent is in one of the trees, i.e., \(a' \in V_k \cup \cdots \cup V_q\). This can be proven using the same arguments as before. Pedagogically, the case where there are more tasks than agents only serves to increase the burden of additional notation without providing significant conceptual insight.

AugDFS is a distributed function satisfying Assumption 4.2 in the distributed setting given by Assumptions 2.1 and 2.2. AugDFS implements a depth-first search (DFS) [102–104] that
adheres to the pattern of agent exploration in Definition 4.1. The goal of the search is to find a free agent and the root $\hat{b}$ is a free vertex. Therefore, by construction the path between that free agent and the root is an augmenting path. On the other hand, Proposition 4.2 guarantees that failure to find a free agent means an augmenting path does not exist.

**Function AugDFS**

Input: Edge $\{\hat{a}, \hat{b}\}$, matching $\mathcal{M}$, graph $(\mathcal{V}_B, \mathcal{E})$.

Output: New MCM $\mathcal{M}_\nu$.

1. **function AugDFS**({$\hat{a}, \hat{b}$}, $\mathcal{M}$, $(\mathcal{V}_B, \mathcal{E})$)  
   2. $\mathcal{F} \leftarrow \emptyset$  
   3. $\mu_a \leftarrow Q(\mathcal{M}, a), \forall a \in \mathcal{A}$  
   4. $\nu_a \leftarrow \mu_a, \forall a \in \mathcal{A}$  
   5. search_complete $\leftarrow$ False  
   6. $t \leftarrow \hat{b}$  
   7. while ¬search_complete do  
   8. $\mathcal{N}_t \leftarrow \{a \in \mathcal{A} | a \notin \mathcal{F}, \{a, t\} \in \mathcal{E}\}$  
   9. $a^* \leftarrow a \in \mathcal{N}_t$  
   10. if $\mathcal{N}_t = \emptyset$ and $t = \hat{b}$ then  
   11. search_complete $\leftarrow$ True  
   12. else if $\mathcal{N}_t = \emptyset$ and $t \neq \hat{b}$ then  
   13. $a^* \leftarrow k$, s.t. $\mu_k = t$  
   14. $t \leftarrow \nu_{a^*}$  
   15. $\nu_{a^*} \leftarrow \mu_{a^*}$  
   16. else if $\mu_{a^*} = \hat{b}$ then  
   17. $\nu_{a^*} \leftarrow t$  
   18. search_complete $\leftarrow$ True  
   19. else  
   20. $\nu_{a^*} \leftarrow t$  
   21. $\mathcal{F} \leftarrow \mathcal{F} \cup \{a^*\}$  
   22. $t \leftarrow \mu_{a^*}$  
   23. end if  
   24. end while  
   25. $\mathcal{M}_\nu \leftarrow \{(a, \nu_a) | a \in \mathcal{A}, \nu_a \neq \hat{b}\}$  
   26. **return** $\mathcal{M}_\nu$  
   27. end function

**Remark 4.3.** AugDFS requires storage of the alternating path between the root and the current vertex $t$. The following is a suggested method to achieve this with distributed storage. All agents keep track of the current vertex $t$ by storing a First-In/Last-Out stack of tasks; a task is added to the stack when the search proceeds in Line 19 and removed when the search backtracks in Line 12. Since an agent and its matched task always appear as a pair in the alternating search, it is not necessary to add agent vertices to the stack.
Remark 4.4. An approach for choosing the next agent to explore in Line 9 in AugDFS is to have \( a^* \leftarrow \arg \min_{a \in \mathcal{N}_t} W(\{a, t\}) \). This corresponds to the min-consensus problem and greedily explores edges with smaller weights first.

Lemma 4.2. Under Assumptions 2.1 and 2.2, AugDFS fulfills the requirements for \texttt{AugPath()} and therefore guarantees that Assumption 4.2 is satisfied.

Proof. At every iteration of the while-loop in AugDFS, there exists an alternating path \( P \) between the current vertex \( t \) and root \( \overline{b} \). Consider the set of agents incident to edges in \( P \), i.e., \( \epsilon_P := \{a \in \mathcal{A}|\{a, b\} \in P\} \), and consider function \( \gamma : \mathcal{A} \rightarrow \mathcal{B} \) mapping an agent in the tree to its parent vertex. From Lines 17 and 20, for all agents \( a \in \epsilon_P \), \( \nu_a = \gamma(a) \). From Lines 4 and 15, we have that \( \nu_a = \mu_a \), for all \( a \notin \epsilon_P \). Thus, at every iteration of the while-loop we have \( \{(a, \nu_a) | a \in \mathcal{A}\} = \{(a, \mu_a) | a \notin \epsilon_P\} \cup \{(a, \gamma(a)) | a \in \epsilon_P\} \).

From Line 25, we get \( \{(a, \nu_a) | a \in \mathcal{A}, \nu_a \neq \overline{b}\} = \{(a, \mu_a) | a \notin \epsilon_P, \mu_a \neq \overline{b}\} \cup \{(a, \gamma(a)) | a \in \epsilon_P\} = \overline{M} \oplus P \). Thus, if \( P \) is an augmenting path, then the function returns \( \mathcal{M}_v = \overline{M} \oplus P \) as required. On the other hand, if no augmenting path exists, the search terminates at \( t = \overline{b} \), i.e., \( \mathcal{P} = \emptyset, \epsilon_P = \emptyset \), and \( \{(a, \nu_a) | a \in \mathcal{A}, \nu_a \neq \overline{b}\} = \{(a, \mu_a) | a \in \mathcal{A}, \mu_a \neq \overline{b}\} = \overline{M} \).

PruneBAP returns a bottleneck assignment. Each component of PruneBAP can be implemented in a distributed setting provided by Assumptions 2.1 and 2.2.

Theorem 4.1. PruneBAP solves Problem 4.1.

Proof. From Proposition 4.1, PruneBAP returns a bottleneck assignment of \( \mathcal{G}_B \). From Lemmas 4.1 and 4.2, the functions \texttt{MaxEdge()} and \texttt{AugPath()} in PruneBAP can be implemented such that they satisfy Assumptions 2.1 and 2.2. Edge removal satisfying Assumptions 2.1 and 2.2 can be implemented as shown in Section 4.3.2.

We derive a bound on the complexity of PruneBAP.

Definition 4.2 (Time step). Given Assumption 2.2, one time step refers to one time step or tick of the global clock shared by all agents.

Proposition 4.3. The worst-case complexity, in terms of time steps, of the distributed implementation of PruneBAP is order \( \mathcal{O}(mn^2D) \), where \( m \) is the number of agents, \( n \) is the number of tasks and \( D \) is the diameter of the communication graph \( \mathcal{G}_C \).
Proof. In each iteration of the while-loop of PruneBAP, at least one edge in $E_B$ is removed. There are therefore at most $|E_B| = mn$ iterations of the while-loop. The while-loop of PruneBAP itself contains two components that depend on time steps. The function $\text{MaxEdge}()$ requires at most $D$ time steps for completion. The distributed search for an augmenting path has order $O(nD)$, where each iteration of the while-loop in AugDFS requires $D$ time steps and there are at most $2n - 1$ iterations. \hfill \Box

If $m = n$, then $|E_B| = n^2$ and PruneBAP has a worst-case complexity of order $O(n^3D)$. The number of iterations of the while-loop in AugDFS is at most $2n - 1$, where in the worst-case the search explores $n$ matched agents and backtracks $n - 1$ times before terminating. The root $\bar{b}$ is known by all agents to be the free task vertex in the graph, which narrows down the search for an augmenting path. Without exploiting the fact that $\bar{b}$ known, finding an augmenting path in a bipartite graph in general has complexity $O(n^2)$, from [17, 84]. We discuss the complexity of AugDFS in more detail in the following section, where we make a comparison with an alternative method to implement the distributed search for an augmenting path.

4.3.4 Alternative Method for Distributed Augmenting Path Search

AugDFS conducts a distributed DFS that allows Assumption 4.2 to be satisfied. However, this is not the only function that fulfills the requirements of Assumption 4.2. We introduce an approach based on a breadth-first search (BFS) [105, 106]. AugBFS does not implement a standard BFS. Multiple agents $a \in A$ are explored simultaneously rather than only one agent at a time.

Each line in AugBFS can be implemented with distributed computation, i.e., in the distributed setting given by Assumptions 2.1 and 2.2. AugBFS has two main steps. The first step is to determine the set of agents to explore next, shown in Line 8. To achieve this in a distributed setting, each unexplored agent individually checks for membership to the set of next agents to explore $N$. The second step is to store all the alternating paths in the alternating search, shown in Line 11. The following remark outlines a method to implement this step in the distributed setting.

Remark 4.5. Each time an agent $a \in A$ is explored, agent $a$ stores its parent and child vertices, $\nu_a$ and $\mu_a$, respectively. This pair of tasks $(\nu_a, \mu_a)$ is then communicated to all other agents. All explored agents $a' \in F$ receive the pair of tasks and determine if task $\mu_a$ is to be added to their
### Function AugBFS

Input: Edge \(\{\bar{a}, \bar{b}\}\), matching \(\mathcal{M}\), graph \((V_B, \bar{E})\).
Output: New MCM \(\mathcal{M}_\nu\).

1: \textbf{function} AugBFS\((\{\bar{a}, \bar{b}\}, \mathcal{M}, (V_B, \bar{E}))\) \hfill \triangleright \text{Set of explored agents}
2: \(\mathcal{F} \leftarrow \emptyset\) \hfill \triangleright \text{Current matched tasks, see Remark 2.1}
3: \(\mu_a \leftarrow Q(\mathcal{M}, a), \forall a \in \mathcal{A}\) \hfill \triangleright \text{Alternating paths to the root vertex}
4: \(\mathcal{R} \leftarrow \emptyset\)
5: \(\text{search\_complete} \leftarrow \text{False}\)
6: \(\mathcal{J} \leftarrow \{\bar{b}\}\) \hfill \triangleright \text{Set of explored tasks}
7: \textbf{while} \text{¬search\_complete} \textbf{do}
8: \(\mathcal{N} \leftarrow \{a \in \mathcal{A}|a \notin \mathcal{F}, b \in \mathcal{J}, \{a, b\} \in \bar{E}\}\) \hfill \triangleright \text{Set of next agents to explore}
9: \(\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{N}\) \hfill \triangleright \text{Flag all chosen agents as explored}
10: \(\nu_a \leftarrow t \in \{b \in \mathcal{J}|\{a, b\} \in \bar{E}, \forall a \in \mathcal{N}\}\) \hfill \triangleright \text{Store parent as new matched task}
11: \(\mathcal{R} \leftarrow \mathcal{R} \cup \bigcup_{a \in \mathcal{N}}\{\{a, \mu_a\}, \{a, \nu_a\}\}\) \hfill \triangleright \text{Store alternating paths}
12: \(\mathcal{J} \leftarrow \bigcup_{a \in \mathcal{N}}\mu_a\) \hfill \triangleright \text{Update set of explored tasks}
13: \textbf{if} \(\mathcal{N} = \emptyset\) \textbf{then}
14: \(\text{search\_complete} \leftarrow \text{True}\) \hfill \triangleright \text{No remaining agents to explore}
15: \(\mathcal{M}_\nu \leftarrow \mathcal{M}\) \hfill \triangleright \text{New MCM unchanged from input MCM}
16: \textbf{else if} \\exists a \in \mathcal{N}, \mu_a = \bar{b}\ \textbf{then}
17: \(\text{search\_complete} \leftarrow \text{True}\) \hfill \triangleright \text{Free agent identified}
18: \(a^* \in \{a \in \mathcal{N}|\mu_a = \bar{b}\}\) \hfill \triangleright \text{Choose one free agent}
19: \(\mathcal{P} \leftarrow P(\mathcal{R}, a^*, \bar{b})\) \hfill \triangleright \text{Find all ancestors of } a^*, \text{ see Remark 4.5}
20: \(\mathcal{M}_\nu \leftarrow \mathcal{M} \oplus \mathcal{P}\) \hfill \triangleright \text{Construct new MCM from new matched tasks}
21: \textbf{end if}
22: \textbf{end while}
23: \textbf{return} \(\mathcal{M}_\nu\)
24: \textbf{end function}

individual sets of descendents. For example, given that an agent \(a' \in \mathcal{F}\) receives a pair of tasks \((\nu_a, \mu_a)\), if \(\nu_a\) is a descendant of \(a'\), then it follows that \(\mu_a\) is also a descendant of \(a'\).

**Lemma 4.3.** Under Assumptions 2.1 and 2.2, AugBFS fulfills the requirements for AugPath() and therefore guarantees that Assumption 4.2 is satisfied.

*Proof.* At every iteration of the while loop in AugBFS, for all \(t \in \mathcal{J}\), there exists an alternating path \(\mathcal{P}\) between \(t\) and \(\bar{b}\). This holds by construction, since an agent \(a \in \mathcal{F}\) has its matched task \(\mu_a\) inserted into the set \(\mathcal{J}\) only after \(a\) is explored. We alternate between edges in the matching and not in the matching as outlined in Definition 4.1. If one or more unmatched agents are explored in Line 8, then the search is successful and it remains for one of the augmenting paths to be selected. Following Proposition 4.2, no augmenting path exists if \(\mathcal{N}\) is empty. In such a case, all agents \(a \in \mathcal{A}\) for which there exists an alternating path between \(a\) and root \(\bar{b}\) have been explored and none of these agents are free vertices. \(\square\)
Agents can execute Lines 8 to 12 without waiting for other agents. The key point is that agents are explored in a way that adheres to Definition 4.1. This indicates that the synchronous communication between agents in Assumption 2.2 is a stronger assumption than necessary. However, we limit the discussion to the synchronous case in this thesis.

### 4.3.5 Comparing Distributed Search Methods

Table 4.1 shows a comparison of AugDFS to AugBFS. Iterations refers to the number of iterations of the while-loops of AugDFS or AugBFS, \( n \) is the cardinality of an MCM of the given bipartite graph, \( D \) is the diameter of the communication graph, and a time step is as provided in Definition 4.2.

<table>
<thead>
<tr>
<th>Function</th>
<th>AugDFS</th>
<th>AugBFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of agents explored per iteration</td>
<td>1</td>
<td>Multiple</td>
</tr>
<tr>
<td>Information communicated per explored agent ( a \in \mathcal{F} )</td>
<td>( \mu_a, W({a, \mu_a}), \text{ and } a )</td>
<td>( \mu_a, \nu_a, \text{ and } a )</td>
</tr>
<tr>
<td>Worst-case number of iterations to complete search</td>
<td>( 2n - 1 )</td>
<td>( n )</td>
</tr>
<tr>
<td>Number of time steps per iteration</td>
<td>( D )</td>
<td>( D )</td>
</tr>
<tr>
<td>Information stored by all agents ( a \in \mathcal{A} )</td>
<td>( \nu_a ) and a First-In/Last-Out stack of tasks</td>
<td>( \nu_a ) and the set of descendant tasks of ( a )</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of AugDFS and AugBFS.

In AugDFS, only one vertex is explored in every iteration. Since only one agent is explored per iteration of the while-loop of AugDFS, the explored agent can be selected greedily, i.e., the edge with smallest weight can be selected; see Remark 4.4. To reach consensus on which agents to explore next, agents collect information from their neighbours, choose a local candidate and send their choice to their neighbours. In contrast, AugBFS explores all vertices with the same level in the tree at every iteration, where the level of a vertex is described in Definition 2.9. Therefore, it returns the shortest augmenting path. However, since multiple agents are explored, agents collect information from their neighbours and then in turn send all this information to their neighbours. Thus, there is a trade-off between exploring multiple vertices per iteration and communicating more information between agents per iteration.

Remark 4.4 discusses the reasoning behind having to communicate \( \mu_a \) and \( W(\{a, \mu_a\}) \) for each explored agent \( a \in \mathcal{F} \) in AugDFS as this is the information communicated between agents.
in a min-consensus problem. Remark 4.5 discusses the reasoning behind having to communicate $\mu_a$ and $\nu_a$ for each explored agent $a \in F$ in AugBFS. Remark 4.1 discusses the reasoning behind having to communicate $a$ in both searches as this information is required for breaking ties in Line 9 of AugDFS and Line 18 of AugBFS.

We consider the worst-case number of iterations for the searches to terminate. The worst-case for AugDFS is if all agents only have length-one paths to the root task vertex. Then, the search would explore $n$ times and backtrack $n - 1$ times. On the other hand the worst-case for AugBFS is if graph $(V_B, \bar{E})$ is a path graph. Each agent has a different level, so the worst-case number of iterations is $n$. Whether or not an augmenting path exists, the worst-case number of iterations to conclude the search is lower for AugBFS than for AugDFS.

The number of time steps per iteration for both AugDFS and AugBFS is $D$ corresponding to the diameter of the communication graph. This ensures information from one particular agent is propagated to every other agent. Remark 4.3 discusses the reasoning behind all agents $a \in A$ having to store $\nu_a$ and a First-In/Last-Out stack of tasks in AugDFS. Remark 4.5 discusses the reasoning behind all agents $a \in A$ having to store $\nu_a$ and the set of descendant tasks of $a$ in AugBFS. Fig. 4.2 demonstrates the difference between the alternating trees constructed by the searches in AugDFS and AugBFS.

### 4.4 Warm-starting Property

We apply concepts in Sections 3.5 to provide a property of PruneBAP that exploits structure of the BAP to allow warm-starting. Once again, we consider the merging of two BAPs in Problem 3.1 from Section 3.5.1. Solving this problem using PruneBAP with an arbitrary MCM $M_0$ at initialisation does not make use of the fact that $M_1$ and $M_2$ are known. We denote this as a cold-start to PruneBAP.

Given $M_1$ and $M_2$, we instead consider the following. It holds that the set $\tilde{M} := M_1 \cup M_2$ is an MCM of the graph $G_3$. Without loss of generality, let $W(e_1) \geq W(e_2)$. Then, it also holds that $e_1$ is the largest edge in $\tilde{M}$. We make use of $\tilde{M}$ to solve Problem 3.1 by choosing it as the initial MCM of pruneBAP. Edges in the set $\{e \in E_3 | W(e) \geq W(e_1), e \notin M_0\}$ are removed from $E_3$ in the first iteration. We denote this as a warm-start to pruneBAP. Fig. 4.3 illustrates a warm-start to PruneBAP.
A Distributable Approach for Solving the Bottleneck Assignment Problem

Figure 4.2: Comparison of alternating trees constructed by AugDFS and AugBFS. The solid lines represent an MCM $\overline{M}$, the edge highlighted in red is the edge to be removed, $\overline{e} = \{\overline{a}, \overline{b}\}$, and $b_1$ is the root vertex.
\[ M_1 \text{ is known:} \]

\[ (e_{12}, e_{21}, e_{22}, e_{11}) \]

\[ M_2 \text{ is known:} \]

\[ (e_{43}, e_{34}) \]

Warm-start, Iteration 1:

\[ (e_{24}, e_{42}, e_{43}, e_{34}, e_{12}, e_{21}, e_{13}, e_{22}, e_{33}, e_{23}, e_{14}, e_{31}, e_{11}, e_{41}, e_{32}, e_{44}) \]

Figure 4.3: Demonstration of applying a warm-start to PruneBAP to solve Problem 3.1. The sets of agents and tasks are \( A_1 = \{a_1, a_2\} \), \( A_2 = \{a_3, a_4\} \), \( B_1 = \{b_1, b_2\} \), and \( B_2 = \{b_3, b_4\} \). Then, \( A_3 \) and \( B_3 \) are the same as \( A \) and \( B \) in Fig. 4.1. In this example, warm-starting pruneBAP allows a bottleneck assignment of \( G_3 \) to be found in 1 iteration.

**Remark 4.6.** Warm-starting is a heuristic and does not guarantee fewer iterations for convergence to a solution to the BAP. For a cold-start, we choose an arbitrary initial MCM \( M_{\text{cold}} \) and by chance this \( M_{\text{cold}} \) could be a solution to the BAP.

In Figure 4.4, we demonstrate an application of warm-starting when solving Problem 3.1 with a numerical example. The locations of agents and tasks in Fig. 4.4 were generated using independent normal distributions with a variance of 100. The distributions for sets \( A_1 \) and \( B_1 \) are centred at the point \((x, y) = (40, 60)\). The distributions for sets \( A_2 \) and \( B_2 \) are centred at the point \((x, y) = (60, 40)\). The number of agents and tasks in this example are \( m_1 = 20 \), \( m_2 = 20 \), \( n_1 = 20 \) and \( n_2 = 20 \). In this particular example, we illustrate a situation where not all of the conditions i., ii., and iii. in Lemma 3.4 hold. That is, the union of the bottleneck assignments from each subproblem is an optimal bottleneck assignment of the combined problem, illustrating the benefit of warm-starting using this prior information in this example problem.

### 4.5 Numerical Analysis

In this section, we present a numerical analysis of PruneBAP and the functions AugDFS and AugBFS. A time step is described in Definition 4.2. Iteration of PruneBAP represents the number of iterations of the while-loop of PruneBAP.

Consider an equal number of agents \( m \) and tasks \( n \) that are all represented by points in \( \mathbb{R}^2 \). Agents are to be assigned to move from their initial positions to assigned target positions such
that a BAP with distance as weights is solved. To this end, we define the weights of the complete bipartite assignment graph $G_B$ to be the Euclidean distance between agents and tasks. Note, since there are an equal number of agents and tasks, the cardinality of any MCM of $G_B$ is also $n$. All coordinates are generated from a uniform distribution between 0 and 100 normalised distance units.

By Assumption 2.2, agents communicate synchronously and share a global clock. For the following examples, the communication between agents is modelled as a complete graph $G_C$, i.e., all agents have a communication link to all other agents and diameter $D(G_C) = D = 1$. For connected communication graphs that are not complete, the number of required time steps scales proportionally with diameter $D$. Thus, the relative comparison of methods is fully illustrated by the case with fully connected communication.

### 4.5.1 Complexity of PruneBAP

We evaluate the number of time steps it takes to run PruneBAP and compare the two implementations of an augmenting path search, AugDFS and AugBFS. Fig. 4.5 shows the average number of iterations of the while-loop of PruneBAP for completion versus the number of tasks, $n$. For every value of $n$, the figure shows the average of 100 realisations of the agent and task positions. As expected, AugBFS requires more iterations of the while-loop of PruneBAP than...
AugDFS. This is because in AugDFS the augmenting path is constructed by a greedy minimisation of weights as described in Remark 4.4. Therefore, the augmenting path that is found typically contains edges with small weights and fewer iterations of the while-loop of PruneBAP are required.

Figure 4.5: Average number of iterations of the while-loop of PruneBAP for completion versus the number of tasks, \( n \).

From Proposition 4.3 with \( m = n \) and \( D = 1 \), the worst-case complexity of PruneBAP is \( O(n^3) \). Fig. 4.6 shows the empirical average number of time steps for completion of PruneBAP versus the number of tasks \( n \). Again, the values are averaged over 100 realisations of the agent and task positions. Although AugDFS results in fewer iterations of the while-loop of PruneBAP than AugBFS, AugBFS results in fewer time steps for completion.

Figure 4.6: Average number of time steps for completion of PruneBAP versus the number of tasks, \( n \).
4.5.2 Size of Messages Exchanged Between Agents

Table 4.1 states that each iteration of the while-loop of AugDFS or AugBFS consists of $D$ time steps, i.e., information from any one agent reaches all agents within $D$ time steps. Both functions require a similar amount of information to be exchanged per explored agent, so the number of explored agents per $D$ time steps is a proxy for the size of messages exchanged between agents.

Note that $D = 1$ is a special case where agents collect information from all agents and do not need to further relay any information. Fig. 4.7 shows the maximum number of explored agents and the mean number of explored agents per $D$ time steps for AugBFS, as well as the exact number of explored agents per $D$ times steps for AugDFS. Once again, the results are averaged over 100 realisations of the agent and task positions for every value of $n$.

PruneBAP converges faster when we apply AugBFS rather than AugDFS, i.e., it requires fewer time steps on average to produce a solution to the BAP. However, the trade-off is that the messages exchanged between agents are larger and not of fixed length.

![Figure 4.7: Average maximum number of explored agents and overall average number of explored agents per $D$ time steps for AugBFS versus number of tasks, $n$. For AugDFS, one agent is explored per $D$ time steps.](image)

4.5.3 Comparison to a Greedy Method

By lack of other distributed algorithms that optimally solve the BAP for comparison, we compare the performance of PruneBAP with a distributed greedy algorithm. A greedy assignment algorithm is one that sequentially chooses the edge with lowest weight given prior selections while ensuring the chosen edges form a matching of $G_B$. The greedy algorithm we consider
as a benchmark is CBAA from [6], which can be applied in the distributed setting defined in Assumptions 2.1 and 2.2. A greedy approach does not necessarily solve the BAP, but is expected to converge faster. We denote the largest weight of all the edges corresponding to the assignment obtained from the CBAA approach as $\Lambda$.

In Fig. 4.8, we consider one example realisation of the agent and task positions with $m = n = 50$. We investigate the number of time steps PruneBAP requires to find an MCM $\mathcal{M}$ with largest weight smaller than $\Lambda$. This is indicated by the time step at which the blue and red marks drop below the black line in Fig. 4.8. CBAA only produces an MCM in the final time step, whereas PruneBAP produces a series of MCMs with a non-increasing largest edge weight.

![Figure 4.8: Weight of largest edge in MCM versus number of time steps.](image)

Fig. 4.9 shows the average number of time steps required for PruneBAP to find an MCM $\mathcal{M}$ that has a largest weight smaller than $\Lambda$. Fig. 4.9 also shows the number of time steps for CBAA to find an assignment. Once again, the averages were taken across 100 realisations for every $n$. Fig. 4.9 suggests that on average, PruneBAP would converge faster by warm-starting, i.e., initialising PruneBAP with the assignment found via CBAA.

### 4.6 Conclusion

PruneBAP iteratively produces an assignment with lower bottleneck weight with the final assignment being a solution to the BAP. The algorithm has several components, i.e., finding a pruned edge set, finding the largest edge amongst agents, and searching for an augmenting path. We derived methods to distribute the execution of each individual component over a network of agents with limited information. In particular, we compared two methods, AugDFS
and AugBFS, for conducting a distributed search for an augmenting path. The two methods provide a trade-off between computational complexity and amount of information that needs to be communicated between agents. We investigated the average numerical complexity of PruneBAP, AugDFS, and AugBFS. As a benchmark, we compared PruneBAP to CBAA, a greedy distributed assignment-finding algorithm. We highlighted two properties of PruneBAP. The first is that PruneBAP produces a series of MCMs with a non-increasing largest edge weight, i.e., at every iteration, there exists a valid assignment of agents to tasks. The second is that PruneBAP is amenable to warm-starting. It is possible to exploit these two properties together by warm-starting with an MCM found via a greedy algorithm and using PruneBAP to iteratively find MCMs with lower largest weight. This allows the option of terminating at any iteration if the MCM is “good enough” with respect to the BAP, i.e., we can choose to expend more computational resources to find better and better MCMs and reach an optimal solution but we can also terminate if the largest weight of an edge is satisfactorily low enough according to the situation.
Chapter 5

A Distributable Approach for Solving the Lexicographic Bottleneck Assignment Problem

5.1 Introduction

In this chapter, we apply the concepts introduced in Chapter 3 to develop a greedy algorithm for solving the LexBAP that is amenable to distributed computation. Furthermore, we provide conditions for when a solution produced by this greedy algorithm is an exact solution to the LexBAP. The problem of solving the BAP in a distributed manner is formally stated as follows.

**Problem 5.1.** Given Assumptions 2.1 and 2.2, obtain a solution to the LexBAP given by (2.2).

This chapter extends the work in the previous chapters the following ways. The foremost extension is that PruneBAP is used to develop an algorithm to produce a greedy solution to the LexBAP that is similarly amenable to a distributed implementation. We explore a particular greedy reformulation of the LexBAP, which we call the Sequential Bottleneck Assignment Problem (SeqBAP) and we focus on a distributed algorithm to solve the SeqBAP.

Note that applying a BAP algorithm “off-the-shelf” to solve the SeqBAP was first proposed in [47, 48]. However, we henceforth refer to such an approach as a naive one as it does not exploit the groundwork structure that was laid in the previous chapters. In this chapter, we exploit both the price of absence of an edge and the warm-starting property of PruneBAP to modify the aforementioned naive greedy approach so that the resulting novel greedy approach has lower worst-case complexity.

We also investigate the full relationship between the LexBAP and its greedy version to determine conditions for which a solution of the LexBAP and SeqBAP is unique and equal. We
show that the solution set of the LexBAP is a subset of the solutions of the SeqBAP, which in turn is a subset of the solution set of the BAP. By establishing that the SeqBAP is a greedy approximation of the LexBAP with certifiable conditions for exactness, the main contribution of this chapter is the derivation an efficient algorithm to solve the LexBAP. The benefits of the proposed algorithm compared to existing literature are two-fold. Firstly, solving the LexBAP according to [17] has a worst-case complexity of $O(n^4)$, where $n$ is number of allocations of tasks to agents that have to be established. By exploiting aforementioned structure in the BAP and LexBAP, we instead present an algorithm for the SeqBAP that has a worst-case complexity of $O(n^3)$. Secondly, this algorithm is intrinsically distributable and can be implemented with computation distributed over agents. We also prove that the novel approach has lower theoretical worst-case complexity than the naive approach and demonstrate that it also has lower empirical complexity in a case study.

5.2 Greedy Approach to the LexBAP

A greedy approach to the LexBAP involves sequentially solving $n$ BAPs, i.e., sequentially choosing the edges corresponding to $T_1(M), T_2(M), ..., T_n(M)$ one at a time, where $T_k$ is the $k$th element of the ordered tuple of weights defined in Section 2.2.3. This does not in general produce a solution to the LexBAP.

Each time an edge is selected, it affects the remaining selections in the sequence. The greedy selection of edges may involve an arbitrary choice between edges with the same weight. For example, consider an MCM $M$ with tuple of weights $T(M) = (5, 3, 3, 3)$ and an MCM $M'$ with tuple $T(M') = (5, 4, 3, 3)$ such that $M \cap M' = \emptyset$. Despite the fact that the MCMs contain different edges, their tuple of costs have the same largest weight of 5. In a greedy approach, the edge in $e' \in M'$ with weight 5 may be selected instead of the edge $e \in M$ with weight 5 since they have the same weight and the choice is an arbitrary one. A greedy approach does not look ahead to take into consideration the weights of the future edges to be selected in the sequence, it only takes into consideration the current selection. By selecting $e'$ first, it is impossible for $M$ to be produced by the greedy approach from this point on. Therefore, this contrived example demonstrates that a greedy approach can potentially produce a suboptimal solution to the LexBAP. The following is an example of a generic greedy algorithm.
Algorithm  A greedy approach for the LexBAP.

Input: Graph $G_B = (V_B, E_B)$.
Output: An MCM $M_B$ of $G_B$ that is a solution to (5.2).

1: $\hat{V} \leftarrow V_B$
2: $\hat{E} \leftarrow E_B$
3: $M_B \leftarrow \emptyset$
4: while $|M_B| < n$ do
5: $\bar{G} \leftarrow (\hat{V}, \hat{E})$ $\triangleright$ Current graph
6: $M^* \leftarrow M \in S(\bar{G})$ $\triangleright$ Find a bottleneck assignment, e.g., via PruneBAP
7: $e^* \leftarrow e \in \eta(M^*)$ $\triangleright$ Find a bottleneck edge
8: $\hat{V} \leftarrow \hat{V} \setminus e^*$ $\triangleright$ Remove bottleneck agent and task
9: $\hat{E} \leftarrow \{\{a, b\} \in \hat{E} | a, b \in \hat{V}\}$ $\triangleright$ Shrink $\hat{E}$
10: $M_B \leftarrow M_B \cup \{e^*\}$
11: end while
12: return $M_B$

We introduce a particular greedy approximation of the LexBAP, which we henceforth refer to as SeqBAP. We are interested in the SeqBAP as it is possible to derive conditions for when the SeqBAP and LexBAP have the same solution set. The SeqBAP is constructed by sequentially choosing the bottleneck edge with maximal price of absence and removing the corresponding bottleneck agent and task from the graph. Thus, it is a greedy solution to LexBAP. Given a set of edges $E' \subseteq E_B$, let the set of edges with largest weight in $E'$ be

$$\eta(E') := \arg\max_{e \in E'} W(e). \quad (5.1)$$

The SeqBAP is formulated as

\[
\text{SeqBAP :} \quad \text{Find } \{\{a_1, b_1\}, \{a_2, b_2\}, ..., \{a_n, b_n\} \} \in X(G_B), \quad (5.2a) \\
\text{s.t. } \forall k \in \{z \in \mathbb{Z}^+ | z \leq n\}, \quad (5.2b) \\
\{a_k, b_k\} \in \arg\max_{e \in \eta(M^k)} \zeta(G^k, e), \quad (5.2c) \\
M^k \in S(G^k), \quad (5.2d) \\
G^k = (A^k \cup B^k, E^k), \quad (5.2e) \\
E^k = \{\{a, b\} \in E_B | a \in A^k, b \in B^k\}, \quad (5.2f) \\
\text{where } \forall k \in \{z \in \mathbb{Z}^+ | 2 \leq z \leq n\}, \quad (5.2g) \\
A^1 = A, B^1 = B, \quad (5.2h) \\
A^{k} = A^{k-1} \setminus \{a_{k-1}\}, \quad (5.2i) \\
B^{k} = B^{k-1} \setminus \{b_{k-1}\}, \quad (5.2j) \\
\]
where \( n \) is the cardinality of any MCM of \( G_B \), \( X(\cdot) \) is the set of MCMs defined in Section 2.1.3, \( \zeta(\cdot) \) is the price of absence from Definition 3.4, and \( S(\cdot) \) is the set of bottleneck assignments from Definition 2.14.

The following lemma shows the sequential selection of edges in the SeqBAP results in edges being selected in order of decreasing weight, the same way they are arranged in the ordered tuple of weights \( T(\cdot) \) in Section 2.2.3. Hence, it shows that the SeqBAP is indeed a greedy approximation of the LexBAP.

**Lemma 5.1.** The edges in (5.2a) have weights such that \( W(\{a_k, b_k\}) \geq W(\{a_{k+1}, b_{k+1}\}) \), for all \( k \in \{1, 2, ..., n-1\} \).

**Proof.** For all \( k \in \{1, 2, ..., n-1\} \), \( W(\{a_k, b_k\}) \geq W(e_k) \) for any \( e_k \in M^k \) because from (5.2a), \( \{a_k, b_k\} \) is selected from the set \( \eta(M^k) \). The matching \( M^k \setminus \{a_k, b_k\} \) is an MCM of \( G^{k+1} \) but not necessarily the solution to the BAP for the graph \( G^{k+1} \). On the other hand, by (5.2c), \( M^{k+1} \) is a solution to the BAP for the graph \( G^{k+1} \). Thus, \( W(e_{k+1}) \leq W(e) \), where \( e_{k+1} \in \eta(M^{k+1}) \) and \( e \in \eta(M^k \setminus \{a_k, b_k\}) \). Therefore, \( W(\{a_k, b_k\}) \geq W(e) \geq W(e_{k+1}) = W(\{a_{k+1}, b_{k+1}\}) \). \( \square \)

By Proposition 3.4, we observe that selecting edges for an MCM that solves the SeqBAP in (5.2a) only requires checking strict positivity of the price of absence, not the explicit value. In order to evaluate the price of absence of an edge explicitly, an additional BAP would have to be solved. Theorem 3.1 provides a method to determine if an edge has positive price of absence without evaluating this extra BAP by instead searching for an augmenting path. Finding one augmenting path is less complex than solving one BAP, which as seen in Chapter 4, itself requires searching for augmenting paths multiple times. This is exploited in the algorithm in Section 5.4 to circumvent the need to find the edge with maximum price of absence as required in (5.2b). In other words, computing the explicit price of absence is expensive and we focus instead on applying Theorem 3.1 to efficiently identify edges with positive price of absence and exploit Proposition 3.4 to show that we do not actually need the explicit price for the SeqBAP.

On the other hand, to guarantee that a solution to the SeqBAP is an exact solution to the LexBAP, we consider the relationship between their solution sets. In the following section, we analyze conditions for a solution to SeqBAP to be an exact solution to the LexBAP. We combine these results to derive a SeqBAP algorithm in Section 5.4, which serves as a greedy approach to finding a solution to the LexBAP with exactness guarantees.
5.3 Conditions for Correct Solutions from Greedy Approach

To find conditions under which an algorithm that solves the SeqBAP can be used to solve the LexBAP with exactness guarantees, we explore the relationship between the solutions to the BAP, LexBAP, and SeqBAP.

**Proposition 5.1.** Given an MCM $\mathcal{M}$ of $G_B$, if $\mathcal{M}$ is a solution to the SeqBAP described in (5.2), then $\mathcal{M}$ is a solution to the BAP described in (2.1).

**Proof.** Let $\mathcal{M}$ be a solution to (5.2). From (5.2b) and (5.2c), $\{a_1, b_1\} \in \eta(\mathcal{M}^1)$, where $\mathcal{M}^1$ is a bottleneck assignment of $G_B$. By Lemma 5.1, $W(\{a_1, b_1\}) \geq W(e)$ for all edges $e \in \mathcal{M}$. Thus, $\mathcal{M}$ is also a solution to (2.1) since $\{a_1, b_1\} \in \eta(\mathcal{M}^1)$ and $\{a_1, b_1\} \in L(\mathcal{M})$. \hfill $\Box$

Combining the statements from Propositions 3.4 and 5.1, we obtain the following corollary.

While Theorem 3.1 provides a method to identify an edge with positive price of absence, Corollary 5.1 motivates the need to identify edges with positive price of absence by showing their relevance in connection to the SeqBAP.

**Corollary 5.1.** If an edge $e_p \in \mathcal{E}_B$ has positive price of absence with respect to $G_B$, then every MCM $\mathcal{M}$ of $G_B$ that is a solution to the SeqBAP given in (5.2) contains $e_p$, i.e., $e_p \in \mathcal{M}$.

**Remark 5.1.** Since Corollary 5.1 also applies for each consecutive bipartite graph $G_k$ in (5.2d), the following holds. Instead of selecting and removing one edge per iteration of $k$ in (5.2b), (5.2g), (5.2h), all edges with positive price of absence in that iteration can be selected and removed as a batch. Intuitively, edges are “locked” into the solution whenever they are found to have positive price of absence.

Given Proposition 5.1, we can go one step further; the following proposition states that all solutions to the LexBAP are also solutions to the SeqBAP.

**Proposition 5.2.** Given an MCM $\mathcal{M}$ of $G_B$, if $\mathcal{M}$ is a solution to the LexBAP described in (2.2), then $\mathcal{M}$ is also a solution to the SeqBAP described in (5.2).

**Proof.** Assume $\mathcal{M} = \{e_1, e_2, ..., e_n\}$ is a solution to (2.2). Without loss of generality assume that $W(e_1) \geq W(e_2) \geq ... \geq W(e_n)$. Let $\mathcal{M}^1 = \mathcal{M}$ and $\mathcal{M}^{k+1} = \mathcal{M}^k \setminus \{e_k\}$ for all $k \in \{1, 2, ..., n-1\}$. Then, it holds that for all $k \in \{1, 2, ..., n\}$, $e_k \in \eta(\mathcal{M}^k)$. Since $\mathcal{M}$ is a solution to (2.2),
for all $k \in \{1, 2, \ldots, n\}$, $M^k \in \arg\min_{M \in \mathcal{X}(G^k)} \max_{e \in M} W(e)$, with $G^k$ defined as in (5.2d). Then, it remains to show that $e_k \in \arg\max_{e \in \eta(M^k)} \zeta(G^k, e)$. If $W(e_k) > W(e_{k+1})$, then $\eta(M^k)$ is a singleton, so $e_k$ is trivially the edge with largest price of absence in $G^k$. If $\eta(M^k)$ is not a singleton, then it holds that $\eta(M^k) = \{e_k, e_{k+1}, \ldots, e_{k+q}\}$, where either $k + q = n$ or $W(e_{k+q}) > W(e_{k+q+1})$. For the LexBAP solution, the choice of which $e \in \eta(M^k)$ is denoted as $e_k$ is arbitrary; edges $e_k, e_{k+1}, \ldots, e_{k+q}$ can be rearranged in any order because their weights are equal. Since the choice is arbitrary, we choose $e_k \in \arg\max_{e \in \eta(M^k)} \zeta(G^k, e)$. Thus for all $k \in \{1, 2, \ldots, n\}$, $e_k \in \arg\max_{e \in \eta(M^k)} \zeta(G^k, e)$ by construction as required in (5.2a).

The following corollary is a special case of Proposition 5.2, where the solution set of the SeqBAP is a singleton.

**Corollary 5.2.** Given an MCM $M$ of $G_B$, if $M$ is the unique solution to the SeqBAP described in (5.2), then $M$ is also the unique solution to the LexBAP described in (2.2).

The converse of Corollary 5.2 is not true. We provide the following counterexample with two agents $a_1, a_2 \in \mathcal{A}$ and two tasks $b_1, b_2 \in \mathcal{B}$, and edges with weights $W(\{a_1, b_1\}) = 2$, $W(\{a_1, b_2\}) = 2$, $W(\{a_2, b_1\}) = 1$, $W(\{a_2, b_2\}) = 2$. In this case, the LexBAP has a unique solution, but the SeqBAP does not.

Fig. 5.1a illustrates Propositions 5.1 and 5.2 while Fig. 5.1b illustrates Corollary 5.2. The following proposition provides conditions for existence of a unique solution to the SeqBAP.

**Proposition 5.3.** Consider an MCM $M$ of $G_B$. Let $M$ be the solution to the SeqBAP given in (5.2). All sequentially selected edges in (5.2b) have positive price of absence in their respective graphs $G^k$ defined in (5.2d) if and only if $M$ is a unique solution to (5.2).

**Proof.** Assume all sequentially selected edges in (5.2b) have positive price of absence in their respective graphs $G^k$. By Corollary 5.1, every solution to (5.2) must contain this set of edges. Thus, the solution is unique.

Assume there exists an edge $e_q$ selected in (5.2b) at iteration $q$ that does not have positive price of absence in $G^q$. Then there exists another bottleneck edge of $G^q$ that can be selected in lieu of $e_q$ at iteration $q$. Thus, the solution to (5.2) is not unique.

Proposition 5.3 is stronger than the result in [47], which only considers the sufficiency but the not necessity of all edges having positive price for uniqueness of the SeqBAP solution. The
relationship between the solution sets of the BAP, SeqBAP and LexBAP allows us to re-derive the following result from [47]. In particular, by combining Corollary 5.2 and Proposition 5.3, we have a sufficient condition for a solution to the LexBAP being unique.

**Corollary 5.3.** Consider an MCM $M$ of $G_B$. Let $M$ be the solution to the SeqBAP given in (5.2). If all sequentially selected edges in (5.2b) have positive price of absence in their respective graphs $G^k$ defined in (5.2d), then $M$ is a unique solution to the LexBAP described in (2.2).

We in turn note that a sufficient condition for all sequentially selected edges in a solution to the SeqBAP having positive price of absence is for all weights in the bipartite graph to be distinct. Therefore, if the weights of all edges in $G_B$ are distinct, then the SeqBAP has a unique solution and this solution is also the unique solution of the LexBAP. In realistic applications, the weights can often be considered to belong to a non-empty interval of real numbers. This occurs for instance in applications where the weights consist of distances between agents and tasks. In such applications, the situation where the weights are non-distinct has zero measure.
5.4 An Algorithm for Solving the LexBAP

We present a greedy distributable algorithm to solve the LexBAP that exploits the structure analysed in Section 3.4. This algorithm provides a flag when the solution is exact based on the results in Section 5.3.

PruneSeq can be initialised with any arbitrary MCM $\mathcal{M}_0$. It uses the functions $\text{MaxEdge}(\cdot)$ and $\text{AugPath}(\cdot)$ introduced Section 4.2. In Section 4.3.3, it is shown that these two functions can be implemented with the distributed setting given in Assumptions 2.1 and 2.2. Once again, we make the following assumption on the number of agents and tasks.

**Assumption 5.1.** Assume that the bipartite graph $\mathcal{G}_B = (A \cup B, \mathcal{E}_B)$ has cardinalities of agent and task sets satisfying $|A| \geq |B| = n$, where $n$ is the cardinality of any MCM of $\mathcal{G}_B$, i.e., the number of agents is greater than or equal to the number of tasks.

The algorithm runs by systematically removing elements from a graph that is initialised with $\mathcal{G}_B$. The following steps are repeated in each iteration of the while-loop beginning in Line 6. First, one edge from the current graph is tested to see if it is a critical edge. Testing to see if an edge is a critical edge involves an augmenting path search. If a critical edge is found, all edges with positive price of absence in the current MCM are identified. This can be implemented by again searching for augmenting paths, which can be carried out with $\text{AugPath}(\cdot)$. All edges with positive price of absence and all edges adjacent to these edges are removed from the current graph. If none of the bottleneck edges have positive price of absence, i.e., $\mathcal{E}_{\zeta^+} \cap \eta(\bar{\mathcal{M}}) = \emptyset$, where $\mathcal{E}_{\zeta^+}$ is given in Corollary 3.2 and $\eta(\cdot)$ is defined in (5.1), then any arbitrary bottleneck edge satisfies (5.2b).

**Remark 5.2.** If none of the bottleneck edges have positive price of absence, i.e., $\mathcal{E}_{\zeta^+} \cap \eta(\bar{\mathcal{M}}) = \emptyset$, where $\mathcal{E}_{\zeta^+}$ is given in Corollary 3.2 and $\eta(\cdot)$ is defined in (5.1), then any arbitrary bottleneck edge satisfies (5.2b).

A key observation is that the graph always reduces in each iteration by removal of edges that are not found to be critical edges, by removal of edges with positive price of absence together with edges adjacent to them, or by removal of an edge that is found to be a critical edge together
Algorithm PruneSeq
Input: Graph $G_B = (V_B, E_B)$ and an MCM $M_0$.
Output: An MCM $M$ of $G_B$ that is a solution to the SeqBAP and a flag $U$ for it being an exact solution to the LexBAP.

1: $\bar{V} \leftarrow V_B$
2: $\bar{E} \leftarrow E_B$
3: $\bar{M} \leftarrow M_0$
4: $M \leftarrow \emptyset$
5: $U \leftarrow \text{True}$
6: while $|\bar{M}| > 0$ do
   7: $\bar{G} \leftarrow (\bar{V}, \bar{E})$ $\triangleright$ Current graph
   8: $(\bar{e}, W(\bar{e})) \leftarrow \text{MaxEdge}(\bar{M})$ $\triangleright$ Find edge with largest weight, i.e., $\bar{e} \in \eta(\bar{M})$
   9: $\bar{E} \leftarrow \phi(\bar{G}, \bar{M})$ $\triangleright$ Shrink $\bar{E}$ by pruning edges
   10: $M_\nu \leftarrow \text{AugPath}(\bar{e}, \bar{M}\{\bar{e}\}, (\bar{V}, \bar{E}\{\bar{e}\}))$ $\triangleright$ Check conditions for critical edge
   11: if $M_\nu \neq \bar{M}$ then
      12: $\bar{M} \leftarrow M_\nu$ $\triangleright$ New MCM within $\bar{E}$ identified
   13: else
      14: $E_\zeta + \leftarrow \emptyset$ $\triangleright$ Set of edges with positive price of absence
      15: for $e' \in \bar{M}$ do
         16: $E' \leftarrow \psi(\bar{G}, \bar{M})$
         17: $M_\nu \leftarrow \text{AugPath}(e', \bar{M}\{e'\}, E'\{e'\})$ $\triangleright$ Check conditions for $\zeta(\bar{G}, e') > 0$
         18: if $M_\nu = \bar{M}$ then
            19: $E_\zeta + \leftarrow E_\zeta + \cup \{e'\}$ $\triangleright$ $e'$ has positive price of absence $\triangleright$ See Corollary 3.2
         20: end if
      21: end for
   22: if $E_\zeta + \cap \eta(\bar{M}) = \emptyset$ then
      23: $U \leftarrow \text{False}$ $\triangleright$ See Remark 5.2
      24: $E_\zeta + \leftarrow E_\zeta + \cup \{e\}$
   25: end if
   26: $\mathcal{V}' \leftarrow \{v \in V_B | v, e' \in E_\zeta +\}$ $\triangleright$ Set of vertices incident to edges in $E_\zeta +$
   27: $\bar{V} \leftarrow \bar{V} \setminus \mathcal{V}'$ $\triangleright$ Remove vertices incident to edges in $E_\zeta +$
   28: $\bar{E} \leftarrow \{a, b \in \bar{E} | a, b \in \bar{V}\}$ $\triangleright$ Shrink $\bar{E}$ by removing edges incident to vertices in $\mathcal{V}'$
   29: $\bar{M} \leftarrow \bar{M} \setminus E_\zeta +$
   30: $M \leftarrow \bar{M} \cup E_\zeta +$
   31: end if
32: end while
33: return $M, U$

with edges adjacent to it. The total number of iterations of the while-loop beginning in Line 6 is upper bounded by $|E_B|$, and depends on how quickly the pool of candidate critical edges $\bar{E}$ shrinks and how many edges with positive prices of absence $|E_\zeta +|$ are found in each iteration.

**Theorem 5.1.** PruneSeq produces a solution to the SeqBAP and can be implemented in the distributed setting given by Assumptions 2.1 and 2.2.

**Proof.** We first prove that PruneSeq is amenable to implementation in the distributed setting.
To this end, we observe that PruneSeq only requires three procedures, i.e., the procedures of edge removal, finding the edge with largest weight and searching for an augmenting path. All three procedures have been shown to be distributable in Section 4.3 and can therefore be implemented in the distributed setting.

Next, we apply the results from the previous chapters to prove convergence to a SeqBAP solution. By Proposition 4.1, a bottleneck assignment is found in Lines 7 to 13. By Corollary 3.2, we can identify multiple edges in a SeqBAP solution by checking their positivity of price of absence. By Theorem 3.1, this involves augmenting path searches as indicated by Line 17. The graph is pruned in Lines 22 to 29 according to how many edges have positive price of absence, i.e., the number of edges in (5.2a) that have been selected so far. Line 24 ensures that at least one edge is selected in each loop of PruneSeq. The process is repeated until all \( n \) edges in (5.2a) have been selected.

Apart from an MCM, the algorithm returns a flag \( \mathcal{U} \) that is true when this MCM is a unique solution to the LexBAP in accordance with Corollary 5.3. If the guard in Line 22 is false for all iterations, then the solution is unique as described in Corollary 5.2. If the flag is returned as false, then the SeqBAP has multiple solutions and the produced MCM may not be a solution to the LexBAP.

**Proposition 5.4.** Assume the number of agents and the number tasks in \( G_B \) are both equal to \( n \). The worst-case complexity of PruneSeq is \( O(n^3D) \), where \( D \) is the diameter of the communication graph.

**Proof.** At most \( n^2 \) edges are tested as candidate critical edges in Line 10 of PruneSeq. Finding and testing an edge involves a distributed max-consensus and an augmenting path search. In the distributed setting, these procedures have orders \( O(D) \) and \( O(nD) \) respectively. The function \( \text{AugPath}(\cdot) \) has complexity \( O(nD) \) because it exploits the fact that a matching of size \( n - 1 \), and the most recently removed edge \( \bar{e} \) are both known inputs. In the worst-case, every edge in \( \mathcal{E}_B \) is tested in this way. Therefore, the worst-case complexity of applying these two procedures to every edge is \( O(n^3D) \). No edge is tested for being a critical edge more than once, and at most \( n \) critical edges must be found.

Each time a candidate proves to be a critical edge, the test in Line 17 is carried out to identify edges with positive price of absence. Testing one edge for positive price according to
Theorem 3.1 involves an augmenting path search, which as mentioned, has complexity $O(nD)$. By the contrapositive of Proposition 3.4, only edges in the current MCM $\mathcal{M}$ are candidates that need to be tested and $\mathcal{M}$ has at most $n$ edges. The complexity of testing all edges in an MCM that has a maximum cardinality of $n$ is $O(n^2D)$.

Testing all edges for being critical edges has complexity $O(n^3D)$. Testing edges for positive price of absence has complexity $O(n^2D)$ per MCM, but it is done at most $n$ times corresponding to the maximum number of critical edges, so in the worst-case it is also $O(n^3D)$. Thus, the complexity of PruneSeq is $O(n^3D)$.

As a comparison, the centralised algorithm to solve the LexBAP presented in [17] has complexity $O(n^4)$ and involves $n$ iterations of solving both a BAP with a complexity of $O(n^{2.5})$ [17] and an SAP [32, 33] with a complexity of $O(n^3)$.

With $D = 1$, PruneSeq has a worst-case complexity of $O(n^3)$; $D = 1$ is a special case corresponding to a centralised algorithm as agents communicate with all other agents. The more complex SAPs are bypassed by applying the augmenting path searches in this greedy approach, yet under the conditions described in Section 5.3 the solution found using either algorithm is identical.

A naive greedy approach for solving the LexBAP that finds $n$ bottleneck edges from scratch, where each subsequent bottleneck edge is found without exploiting knowledge of previous bottleneck assignments, has complexity $O(n^{3.5})$. This type of naive greedy approach using an “off-the-shelf” BAP algorithm in sequence was first proposed in [47, 48] and is shown in Section 5.2. The algorithm proposed in [47] is in fact a naive SeqBAP algorithm and it additionally returns the explicit prices of absence of edges. This means it requires solving $n$ iterations of two BAPs, but its complexity is still $O(n^{3.5})$.

PruneSeq is one example of how the warm-starting property of PruneBAP is exploited. Rather than solving $n$ bottleneck edges from scratch, each subsequent bottleneck edge is found from within the set $\bar{\mathcal{E}}$, which has strictly decreasing cardinality with every iteration of the while-loop of PruneSeq. To use the warm-starting property, we require an MCM to be constructed from the MCM of the previous problem, which is seen in Line 29. Figure 5.2 illustrates how the warm-starting property of PruneBAP is applied in PruneSeq. The set of edges $\bar{\mathcal{E}}$ always
reduces in cardinality with each iteration and the MCM at each iteration is constructed from the previous MCM.

Iteration 1:
\begin{align*}
& e_{44} \quad e_{41} \quad e_{42} \quad e_{43} \quad e_{33} \quad e_{32} \quad e_{34} \quad e_{22} \quad e_{21} \quad e_{23} \quad e_{24} \quad e_{11} \quad e_{12} \quad e_{13} \quad e_{14} \\
\end{align*}

Iteration 2:
\begin{align*}
& e_{44} \quad e_{42} \quad e_{43} \quad e_{33} \quad e_{32} \quad e_{34} \quad e_{22} \quad e_{13} \quad e_{14} \\
\end{align*}

Iteration 3:
\begin{align*}
& e_{44} \quad e_{43} \quad e_{33} \quad e_{32} \quad e_{34} \quad e_{22} \quad e_{14} \\
\end{align*}

Iteration 4:
\begin{align*}
& e_{44} \\
\end{align*}

Figure 5.2: Demonstration of PruneSeq with four agents \( A = \{a_1, a_2, a_3, a_4\} \) and four tasks \( B = \{b_1, b_2, b_3, b_4\} \). The set of edges \( E_B \) are arranged in order of ascending weight, where \( e_{pq} \) is the edge between agent \( a_p \) and task \( b_q \). The circled edges represent edges in \( \bar{M} \). At each iteration, a warm-start of PruneBAP is applied since we already know the circled edges from the previous iteration.

Furthermore, we have discussed how PruneSeq also exploits the structure of the BAP and LexBAP discussed in Chapter 3. Although PruneSeq relies on identifying edges with positive price of absence, the explicit value of the price of absence is never computed. We exploited this fact to further reduce the complexity of PruneSeq in Proposition 5.4. In applications that utilise the value of the price of absence, e.g., to quantify the robustness of an assignment in [47], additional computation is required.

### 5.5 Case Study

Consider \( n \) agents represented by points in \( \mathbb{R}^2 \). Each agent must move from its initial position to one of \( n \) goal positions. The assignment of agents to goal positions is done by solving the LexBAP, where the weights are given by the agent-goal distances. All coordinates are generated from a uniform distribution between values of 0 and 100 normalised distance units. Since the weights are almost surely distinct, the solution to the SeqBAP is the solution to the LexBAP and it is unique.
An example application for this case study would be a ride-sharing service, where the agents are cars and the goal positions are pick-up locations. Another example is a drone delivery service or autonomous forklifts in a warehouse, where the agents are drones or forklifts and the goal positions are item pick-up locations. Assuming tasks are executed in parallel, all these applications correspond to a BAP as the longest pick-up time needs to be minimised. In contrast, minimising the sum of costs may result in some agents completing their tasks very quickly at the expense of other agents completing theirs very slowly. Furthermore from Fig. 5.1, solving the BAP allows the possibility of further minimising the sequential bottleneck edges, i.e., solving the LexBAP. The benefit of this is that the LexBAP itself has further desirable properties, e.g., an inherent property that provides collision avoidance guarantees discussed in Chapter 6.

Fig. 5.3 shows the performance of a centralised version of PruneSeq benchmarked against an algorithm for solving general LexBAPs and a naive greedy approach for solving the LexBAP. The SAP component of the exact LexBAP method is solved using the Hungarian Algorithm from [32] while the BAP component in both the exact LexBAP and the naive greedy approach is solved using an off-the-shelf threshold method from [17]. That is, finding a bottleneck assignment in Line 6 of the naive greedy algorithm shown in Section 5.2 is carried out using an off-the-shelf threshold method. The average time for each value of \(n\) is taken over 100 realisations of agent and task positions. These results are obtained with an Intel i5-6600 CPU at 3.30GHz. For all algorithms tested, Fig. 5.3 shows the average time increases as the number of agents and tasks in the problem increases, i.e., as \(n\) increases.

![Figure 5.3: Comparison of the runtime of an exact LexBAP algorithm, a naive greedy approach, and PruneSeq.](image)

From Proposition 5.4, the theoretical worst-case complexity of PruneSeq is \(O(n^3D)\), which
is lower than the theoretical worst-case complexities of the exact LexBAP and naive greedy approach that are order $O(n^4D)$ and $O(n^{3.5}D)$ respectively. On the other hand, Fig. 5.3 shows that the empirical complexity of PruneSeq is also lower than the empirical complexities the exact LexBAP and naive greedy approach for this case study.

The following demonstrates that PruneSeq can be implemented under the distributed setting given by Assumptions 2.1 and 2.2. Fig. 5.4 shows one realisation of agent and task positions with $n = 10$. First, PruneSeq is applied for the case where any agent can communicate with all other agents directly in one time step of the global clock, where a time step is given in Definition 4.2, i.e., the communication graph has diameter $D = 1$. Additionally, we consider the case where agents only communicate with other agents that are located within a radius of 30 units as illustrated by the shaded areas in Fig. 5.4. This results in a communication graph with diameter $D = 5$. For both cases, PruneSeq returns the exact the solution of the LexBAP. For the case with $D = 1$, it takes 111 time steps for PruneSeq to obtain the solution. For the case with $D = 5$, it takes 555 time steps.

Figure 5.4: Demonstration of PruneSeq in a distributed setting. The shaded circles show the communication range of each agent. Agents are only able to communicate with other agents within their circles.
5.6 Conclusion

We presented an approach to find an MCM of a bipartite graph that is the solution to the LexBAP by employing a method that solves a series of BAPs, where the edges in the bipartite graph are removed in each iteration. For each of these BAPs, we showed that if an edge has a positive price of absence, then that edge is guaranteed to be an element of the LexBAP solution, and there may be multiple such edges each time a BAP is solved. We called this greedy reformulation of the LexBAP the SeqBAP. In Chapter 3, we considered the similarities in structure of a critical edge to an edge with positive price of absence and used this to derive a method to identify edges with positive price of absence that involves a search for augmenting paths. In this chapter, we proceeded to exploit this to enable the SeqBAP to be solved efficiently. We derived the relationship between the BAP, the SeqBAP, and the LexBAP by comparing their solution sets. In particular, we showed that the solutions to the LexBAP are a subset of the solutions of the SeqBAP. We provided conditions for when the SeqBAP has a unique solution, which implies that this solution also uniquely solves the LexBAP. Furthermore, we showed that all edges of the graph having distinct weight values is a sufficient condition for this uniqueness. We combined these results into a proposed algorithm that provides a greedy solution to the LexBAP and a certificate for when this solution is exact. The algorithm has a complexity of $O(n^3)$, which is lower than methods for solving the LexBAP that have a complexity of $O(n^4)$. Moreover, the proposed algorithm can be implemented with computation that is distributed across agents.
Chapter 6

Collision Avoidance for Carrying Out Tasks with Mobile Robots

6.1 Introduction

In this chapter, we consider the execution of tasks after the assignment problem has been computed. In particular, we consider the situation where agents are mobile robots and weights used in the task assignment are based on the distances between the agents and the tasks. We focus on one aspect of the task execution phase, i.e., providing collision avoidance guarantees as the mobile robots move towards their assigned tasks. Two methods for guaranteeing collision avoidance are investigated in this chapter.

The first method concerns assigning agents in a particular way during the task assignment phase. Assigning agents according to certain types of assignment problems like the SAP and the LexBAP provide inherent collision avoidance certificates. As we are motivated by the time-critical decoy seduction problem in Section 1.1, we focus on the LexBAP and show that solving this assignment problem provides time-varying position constraints for agents as they move to carry out their tasks. Rather than determining trajectories for all agents, the method proposed here provides time-varying sets of positions that guarantee collision avoidance but leave some degree of freedom for low-level path planning and motion control. This degree of freedom depends on a so-called robustness margin, which is in fact a special application of the price of absence discussed in previous chapters. The constraints for an individual agent do not explicitly depend on the positions of the other agents. We prove that it is sufficient for every agent to satisfy its local constraints in order to guarantee that no agent that is assigned to one of the tasks will collide with any other agent.
The second method concerns the use of CBFs to derive linear time-varying constraints on the acceleration of agents, which provide collision avoidance guarantees for agents modelled as double integrators as they move towards their task locations. The constraints depend on the positions and velocities of the other agents but each agent computes its own constraints locally using only this information. One benefit of this second method is that it is readily applicable to collision avoidance of obstacles modelled as convex polytopes.

Once again, we consider a bipartite graph $G_B = (V_B, E_B)$, with vertex set $V_B = A \cup B$ as introduced in Section 2.1.3.

### 6.2 Collision Avoidance Guarantee via SeqBAP

At a given time $t$, the centroid location of an agent $a \in A$ is given by the vector $p_a(t) \in \mathbb{R}^q$ and the location of a task $b \in B$ is given by $p_b \in \mathbb{R}^q$, where $q \in \mathbb{N}$. In this section, we use an arbitrary distance function that satisfies the triangle inequality

$$d(p, p') \leq d(p, p'') + d(p'', p')$$

for all vectors $p, p', p'' \in \mathbb{R}^q$. In order to guarantee collision avoidance according to this distance function, any two agents $a, a' \in A, a \neq a'$ must satisfy

$$d(p_a(t), p_{a'}(t)) > s_{a,a'},$$

where $s_{a,a'} = s_{a',a} \geq 0$ is the safety distance between $a$ and $a'$.

We note that the collision avoidance condition in (6.2) couples the motion control problem of agents $a$ and $a'$. Satisfying this condition introduces a non-convex constraint with respect to the positions of the agents, $p_a(t)$ and $p_{a'}(t)$, and is therefore challenging. The objective considered in this section is to derive local position constraints for each individual agent, that sufficiently guarantee avoidance of collisions of agents that are assigned to tasks such that the maximum agent-to-destination distance, i.e., the bottleneck, is minimised.

We introduce the notion of a robustness margin that characterises the largest price of absence amongst all bottleneck edges.
**Definition 6.1** (Robustness margin). The robustness margin of $G_B$ is $R(G_B) := \max_{e \in Z(G_B)} \zeta(e)$, where $Z(\cdot)$ is the set of all bottleneck edges given in Definition 2.15 and $\zeta(\cdot)$ is the price of absence of an edge given in Definition 3.4.

When computing the SeqBAP given in (5.2), we sequentially choose bottleneck edges that meet the robustness margin value of each sequential subgraph $G_k$. Let $M^* \in S(G_B)$ be a solution to the SeqBAP. Without loss of generality, we denote the edges in $M^*$ as $M^* = \{(a^*_1, b^*_1), (a^*_2, b^*_2), \ldots, (a^*_n, b^*_n)\}$, where $n = |M^*|$. We define the $k$th order bottleneck edge as

$$\{a^*_k, b^*_k\} \in M^*,$$

i.e., the $k$th edge selected in (5.2), and define the $k$th order robustness margin as

$$R_k = R(G^k), \quad (6.3)$$

i.e., the robustness margin of the $k$th subgraph in (5.2), for $k \in \{1, \ldots, n\}$.

For the remainder of Section 6.2, we assume the following things about the bipartite graph $G_B$. Without loss of generality, we consider the case where there are more agents than tasks. We also only consider the non-trivial case where there is at least one task and the case where the number of tasks is equal to the cardinality of any MCM of $G_B$.

**Assumption 6.1.** Assume that the cardinalities of agent and task sets satisfy $|A| = m \geq |B| = n \geq 1$, where $n$ is the cardinality of any MCM of $G_B$.

In terms of the assignment problem, we assume the following things about $G_B$.

**Assumption 6.2.** Assume the assignment weights are the distances between initial agent positions and destinations, i.e., $W(\{a, b\}) = d(p_a(0), p_b)$ for all agents $a \in A$ and tasks $b \in B$. Assume agents are allocated to tasks according to the SeqBAP defined in (5.2) and that the $k$th order robustness margins are all strictly positive, i.e., $R_k > 0$ for all $k \in \{1, \ldots, n\}$.

Note that by Corollary 5.3, Assumption 6.2 implies that there is a unique solution to the SeqBAP that is also the unique solution to the LexBAP. We investigate the collision avoidance property of such a problem in this section.
Proposition 6.1. Given Assumptions 6.1 and 6.2, the weights of the kth order bottleneck edge satisfy the inequality

\[ W(\{a_k,b_k^*\}) + R_k \leq \max\{W(\{a_k^*,b_k^*\}), W(\{a_l^*,b_k^*\})\}, \]

for all \( k \in \{1, ..., n - 1\} \) and \( l \in \{k + 1, ..., n\} \).

Proof. By (6.3) and Definition 6.1, \( W(\{a_k,b_k^*\}) + R_k = Y((\mathcal{V}^k, \mathcal{E}^k \setminus \{a_k^*,b_k^*\})) \), where \( Y(\cdot) \) is the bottleneck weight given in Definition 2.13. Let \( M^* \) be a bottleneck assignment of \( G^k \) such that \( \{a_k,b_k^*\}, \{a_l^*,b_k^*\} \in M^* \). Consider the augmenting path \( P = \{\{a_k^*,b_k^*\}, \{a_l^*,b_k^*\}\} \) relative to \( M^* \setminus \{a_k^*,b_k^*\} \). By Theorem 3.1, since \( \{a_k^*,b_k^*\} \) has positive price of absence in \( G^k \), there does not exist an augmenting path relative to \( M^* \setminus \{a_k^*,b_k^*\} \) within the matching-sublevel set \( \psi(G^k) \setminus \{a_k^*,b_k^*\} \). In other words, the augmenting path \( P \) that gives the symmetric difference \( M' = M^* \setminus \{a_k^*,b_k^*\} \oplus P \) contains at least one edge with weights strictly larger than \( W(\{a_k^*,b_k^*\}) \). To be explicit, \( M' \) is a new matching

\[ M' = (M^* \cup \{\{a_k^*,b_l^*\}, \{a_l^*,b_k^*\}\}) \setminus \{a_k^*,b_k^*\}, \{a_l^*,b_l^*\} \]

in which \( a_k^* \) and \( a_l^* \) have swapped matched tasks. After swapping matched tasks, one of the new edges has the largest weight of all edges in \( M' \), i.e.,

\[ \max\{W(\{a_k^*,b_l^*\}), W(\{a_l^*,b_k^*\})\} = \max_{e \in M'} W(e) \]

as a consequence of \( P \) containing at least one edge with weights strictly larger than \( W(\{a_k^*,b_k^*\}) \).

We now observe that this matching \( M' \) is an MCM of the subgraph \( (\mathcal{V}^k, \mathcal{E}^k \setminus \{a_k^*,b_k^*\}) \). By Definition 2.13,

\[ Y((\mathcal{V}^k, \mathcal{E}^k \setminus \{a_k^*,b_k^*\})) \leq \max_{e \in M'} W(e) = \max\{W(\{a_k^*,b_l^*\}), W(\{a_l^*,b_k^*\})\}. \]

\[ \square \]

Proposition 6.2. Given Assumptions 6.1 and 6.2, the weights of the kth order bottleneck edge satisfy the inequality

\[ W(\{a_k^*,b_k^*\}) + R_k \leq W(\{a_l^*,b_k^*\}), \]
for all $k \in \{1, ..., n\}$ and $a' \in A \setminus \{a_1^*, ..., a_n^*\}$.

Proof. By (6.3) and Definition 6.1, $W(\{a_k^*, b_k^*\}) + R_k = Y((V^k, E^k \setminus \{\{a_k^*, b_k^*\}\}))$, where $Y(\cdot)$ is the bottleneck weight given in Definition 2.13. Let $\mathcal{M}^*$ be a bottleneck assignment of $\mathcal{G}^k$ such that $\{a_k^*, b_k^*\} \in \mathcal{M}^*$ and $a'$ is free. Consider the augmenting path $\mathcal{P} = \{\{a', b_k^*\}\}$ relative to $\mathcal{M}^* \setminus \{\{a_k^*, b_k^*\}\}$. By Theorem 3.1, since $\{a_k^*, b_k^*\}$ has positive price of absence in $\mathcal{G}^k$, there does not exist an augmenting path relative to $\mathcal{M}^* \setminus \{\{a_k^*, b_k^*\}\}$ within the matching-sublevel set $\psi(\mathcal{G}^k) \setminus \{\{a_k^*, b_k^*\}\}$. In other words, the augmenting path $\mathcal{P}$ that gives the symmetric difference $\mathcal{M}' = \mathcal{M}^* \setminus \{\{a_k^*, b_k^*\}\} \oplus \mathcal{P}$ contains at least one edge with weights strictly larger than $W(\{a_k^*, b_k^*\})$. To be explicit, $\mathcal{M}'$ is a new matching

$$\mathcal{M}' = (\mathcal{M}^* \cup \{\{a', b_k^*\}\}) \setminus \{\{a_k^*, b_k^*\}\}$$

in which $a'$ has matched to $b_k^*$ and $a_k^*$ has become free. The new edge has the largest weight of all edges in $\mathcal{M}'$, i.e., $W(\{a', b_k^*\}) = \max_{e \in \mathcal{M}'} W(e)$, as a consequence of $\mathcal{P}$ containing at least one edge with weights strictly larger than $W(\{a_k^*, b_k^*\})$. We now observe that this matching $\mathcal{M}'$ is an MCM of the subgraph $(V^k, E^k \setminus \{\{a_k^*, b_k^*\}\})$. By Definition 2.13,

$$Y((V^k, E^k \setminus \{\{a_k^*, b_k^*\}\})) \leq \max_{e \in \mathcal{M}'} W(e) = W(\{a', b_k^*\}).$$

We first investigate sufficient conditions for collision avoidance based on the distances of agents from their initial position to their destination assigned according to the SeqBAP. Then, we introduce time-dependent position constraints for the individual agents that provide collision avoidance guarantees.

### 6.2.1 Sufficient Conditions for Collision Avoidance

Using the safety distances from (6.2), we provide a first condition which guarantees that an assigned agent does not collide with any other agent at a particular time.
Lemma 6.1. Given Assumptions 6.1 and 6.2, the $k$-th order bottleneck agent $a^*_k$ does not collide with any other agent $a' \in A \setminus \{a^*_k\}$ at time $t$, for $k \in \{1, ..., n\}$ if

$$d(p_{a'}(0), p_{a'}(t)) + d(p_{a^*_k}(t), p_{b^*_k}) < W(\{a', b^*_k\}) - s_{a', a^*_k}. \tag{6.4}$$

Proof. Considering the triangle inequality in (6.1), the distance between the initial position of agent $a'$ and the target destination assigned to agent $a^*_k$ is bounded,

$$d(p_{a'}(0), p_{a'}(t)) \leq d(p_{a'}(0), p_{a^*_k}(t)) + d(p_{a^*_k}(t), p_{a^*_k}).$$

By applying (6.4) with $W(\{a', b^*_k\}) = d(p_{a'}(0), p_{b^*_k})$, we obtain $d(p_{a'}(t), p_{a^*_k}(t)) > s_{a', a^*_k}$ as required by (6.2).

For guaranteed collision avoidance among assigned agents we combine the concept of robustness margin with the condition given in Lemma 6.1.

Proposition 6.3. Given Assumptions 6.1 and 6.2, the $k$-th order bottleneck agent $a^*_k$ does not collide with a higher order bottleneck agent $a^*_l$ at time $t$, for $k \in \{1, ..., n-1\}$ and $l \in \{k+1, ..., n\}$ if both

$$d(p_{a^*_l}(0), p_{a^*_l}(t)) + d(p_{a^*_k}(t), p_{b^*_l}) < W(\{a^*_l, b^*_k\}) + R_k - s_{a^*_l, a^*_k}, \tag{6.5a}$$

$$d(p_{a^*_l}(0), p_{a^*_k}(t)) + d(p_{a^*_k}(t), p_{b^*_l}) < W(\{a^*_l, b^*_k\}) + R_k - s_{a^*_l, a^*_k}. \tag{6.5b}$$

Proof. For an arbitrary order $k \in \{1, ..., n-1\}$, let $l \in \{k+1, ..., n\}$ be an arbitrary higher order. If $W(\{a^*_l, b^*_k\}) > d(p_{a^*_l}(0), p_{a^*_l}(t)) + d(p_{a^*_k}(t), p_{b^*_l}) + s_{a^*_l, a^*_k}$, agents $a^*_k$ and $a^*_l$ do not collide at time $t$ as shown in Lemma 6.1, with agent $a^*_k$ playing the role of the ‘other agent’.

It remains to consider the case where $W(\{a^*_l, b^*_k\}) \leq d(p_{a^*_l}(0), p_{a^*_l}(t)) + d(p_{a^*_k}(t), p_{b^*_l}) + s_{a^*_l, a^*_k}$. Assume for the sake of contradiction that the condition in (6.4) also does not hold from perspective of agent $a^*_l$, where agent $a^*_k$ is the ‘other agent’, i.e., $W(\{a^*_k, b^*_l\}) \leq d(p_{a^*_l}(0), p_{a^*_k}(t)) + d(p_{a^*_l}(t), p_{b^*_l}) + s_{a^*_l, a^*_k}$. Then, from (6.5a) we have $W(\{a^*_l, b^*_k\}) < W(\{a^*_l, b^*_k\}) + R_k$ and from (6.5b) we have $W(\{a^*_k, b^*_l\}) < W(\{a^*_k, b^*_l\}) + R_k$. This however contradicts Proposition 6.1. It follows that $W(\{a^*_k, b^*_l\}) > d(p_{a^*_l}(0), p_{a^*_k}(t)) + d(p_{a^*_l}(t), p_{b^*_l}) + s_{a^*_k, a^*_l}$. Thus, according to Lemma 6.1, agents $a^*_k$ and $a^*_l$ do not collide at time $t$. \qed
6.2.2 Local Constraints for Guaranteed Collision Avoidance

We now derive individual position constraints for every agent such that if all agents satisfy their associated constraints, collisions involving assigned agents are avoided. The constraints rely on the robustness of the sequential bottleneck assignment as formalised in the following assumption.

**Assumption 6.3.** There exists an upper bound on the safety distance, \( s \geq s_{a,a'} \), between all agents, \( a, a' \in A \), \( a \neq a' \), that is smaller than the \( k \)-th order bottleneck robustness margin for all \( k \in \{1, ..., n\} \), i.e., \( s < \hat{R} := \min_{k \in \{1, ..., n\}} R_k \).

The agent position constraints are composed of up to two components. The first component is a bound on the distance of an agent from its initial position. This bound is characterised by the same time-varying parameter \( \delta(t) \), for all agents but has a saturation value \( \hat{\delta}_a \) that is agent dependent.

**Assumption 6.4.** The distance of an agent \( a \in A \), to its initial position is bounded by

\[
d(p_a(0), p_a(t)) < r_a^o(t) := \min\{\delta(t), \hat{\delta}_a\},
\]

for \( t \in [0, T] \), where

\[
\hat{\delta}_a = \begin{cases} 
\Pi_a & \text{if } a \text{ is assigned}, \\
\Pi_{a^*_n} & \text{otherwise},
\end{cases}
\]

such that for assigned agent \( a^*_k \), \( k \in \{1, ..., n\} \) corresponding to the \( k \)-th order bottleneck edge

\[
\Pi_{a^*_k} := \min_{l \in \{1, ..., k\}} W(a^*_k, b^*_k) + R_l - \frac{1}{2}(\hat{R} + s),
\]

and \( \delta(t) \geq \frac{1}{2}(\hat{R} - s) \).

The second component of the position constraints applies only to assigned agents. It consists of a bound on the distance of an agent to its target destination that decreases when the bound on the distance from the initial position increases.
Assumption 6.5. The distance of an assigned agent $a^*_k$ to its destination is bounded by

$$d(p_{a^*_k}(t), p_{b^*_k}) < r_{a^*_k}^d(t) := \Pi_{a^*_k} - \min\{\delta(t), \hat{\delta}_{a^*_k}\} + \frac{1}{2}(\hat{R} - s),$$

for $t \in [0, T]$ and all assigned agents $a^*_k$, $k \in \{1, ..., n\}$ corresponding to the $k$th order bottleneck edges.

Satisfaction of these local bounds provides a sufficient condition for collision avoidance as shown in the following.

Theorem 6.1. Given Assumptions 6.1 and 6.2 with robustness margins satisfying Assumption 6.3, no assigned agent $a^*_k$, $k \in \{1, ..., n\}$ collides with any other agent $a' \in A$ at any time $t \in [0, T]$ if all agents satisfy the position bounds of Assumption 6.4 and all assigned agents additionally satisfy the position bounds of Assumption 6.5.

Proof. By construction, the saturation values in Assumptions 6.4 and 6.5 satisfy $\Pi_{a^*_k} \geq \Pi_{a^*_l}$, for all $k \in \{1, ..., n\}, l \in \{k, ..., n\}$. Therefore by Assumption 6.4, $d(p_{a^*_l}(0), p_{a^*_l}(t)) < \min\{\delta(t), \hat{\delta}_{a^*_l}\} \leq \min\{\delta(t), \hat{\delta}_{a^*_l}\}$, which gives $d(p_{a^*_l}(0), p_{a^*_l}(t)) + d(p_{a^*_l}(0), p_{b^*_l}) < \min_{j \in \{1, ..., k\}} W(\{a^*_j, b^*_j\}) + R_j - s \leq W(\{a^*_k, b^*_k\}) + R_k - s$. We also have by Assumption 6.5, $d(p_{a^*_l}(t), p_{b^*_l}) < \Pi_{a^*_l} - \min\{\delta(t), \hat{\delta}_{a^*_l}\} + \frac{1}{2}(\hat{R} - s) \leq \Pi_{a^*_k} - \min\{\delta(t), \hat{\delta}_{a^*_k}\} + \frac{1}{2}(\hat{R} - s)$, which gives $d(p_{a^*_l}(0), p_{a^*_l}(t)) + d(p_{a^*_l}(t), p_{b^*_l}) < \Pi_{a^*_k} - \min\{\delta(t), \hat{\delta}_{a^*_k}\} + \min\{\delta(t), \hat{\delta}_{a^*_k}\} + \frac{1}{2}(\hat{R} - s) \leq \min_{j \in \{1, ..., k\}} W(\{a^*_j, b^*_j\}) + R_j - s \leq W(\{a^*_k, b^*_k\}) + R_k - s$. By Assumption 6.3 the conditions in Proposition 6.3 are satisfied for all $k \in \{1, ..., n - 1\}$ and $t \in [0, T]$. Thus, none of the assigned agents collide with each other.

We also have for the unassigned agents $d(p_{a'}(0), p_{a'}(t)) < \min\{\delta(t), \hat{\delta}_{a'}\} = \min\{\delta(t), \hat{\delta}_{a'}\}$, for all $k \in \{1, ..., n\}, a' \in A \setminus \{a^*_1, ..., a^*_n\}$. Because of Proposition 6.2, Lemma 6.1 is satisfied for all $k \in \{1, ..., n\}, i' \in A \setminus \{i^*_1, ..., i^*_n\}$, and $t \in [0, T]$. Thus, assigned agents also do not collide with unassigned agents. \qed

We note that there exist positions $p_{a^*_k}(t)$, for every assigned agent $a^*_k$, $k \in \{1, ..., n\}$ at every time $t \in [0, T]$, that satisfy Assumptions 6.4 and 6.5 if Assumption 6.3 holds. That is, Assumption 6.4 bounds the position $p_{a^*_k}(t)$ to lie within a ball centred at the initial position $p_{a^*_k}(0)$ with radius $r_{a^*_k}^d(t)$ in $[\frac{1}{2}(\hat{R} - s), \Pi_{a^*_k}]$ and Assumption 6.5 bounds the position $p_{a^*_k}(t)$ to lie within a ball centred at the target destination $p_{b^*_k}$ with radius $r_{a^*_k}^d(t)$ in $[\frac{1}{2}(\hat{R} - s), \Pi_{a^*_k}]$. The two balls intersect for all $t \in [0, T]$ if $\hat{R} > s$ because the sum of the radii is larger than the distance
between the centres

\[
\begin{align*}
    r_{a_k}^d(t) + r_{a_k}^d(t) &= \min\{\delta(t), \delta_{a_k}\} + \Pi_{a_k} - \min\{\delta(t), \delta_{a_k}\} + \frac{1}{2}(\hat{R} - s) \\
    &= \Pi_{a_k} + \frac{1}{2}(\hat{R} - s) \\
    &= \min_{\ell \in \{1, \ldots, k\}} W(\{a_k^\ell, b_k^\ell\}) + R_l - s \\
    &\geq W(\{a_k^\ell, b_k^\ell\}) + \hat{R} - s \\
    &> W(\{a_k^\ell, b_k^\ell\}) \\
    &= d(p_{a_k}(0), p_{a_k}).
\end{align*}
\]

It follows that the constrained sets of safe positions constructed from the bounds in Assumptions 6.4 and 6.5 are given by

\[
S_{a_k}(t) = \{p \in \mathbb{R}^q | d(p_{a_k}(0), p) < r_{a_k}^d(t), d(p, p_{b_k}) < r_{a_k}^d(t)\},
\]  

(6.6a)

for all assigned agents \(a_1^k, k \in \{1, \ldots, n\}\) and

\[
S_{a'}(t) = \{p \in \mathbb{R}^q | d(p_{a'}(0), p) < r_{a'}^d(t)\},
\]

(6.6b)

for all unassigned agents \(a' \in A \setminus \{a_1^*, \ldots, a_n^*\}\) and are both non-empty if Assumption 6.3 holds.

For any agent, \(a \in A\), the safe set \(S_a(t)\) depends on the timing parameter \(\delta(t)\), the \(k\)-th order bottleneck weights \(W(\{a_k^\ell, b_k^\ell\})\) and robustness margins \(R_k\) obtained in the SeqBAP, for \(k \in \{1, \ldots, n\}\), the initial position, \(p_a(0)\), and the location of its assigned destination, \(p_{b_a}\), if assigned. We see that the larger the robustness margins are, the less the agent positions need to be constrained in (6.6).

A SeqBAP solution and the robustness margins are determined only based on information of the initial agent positions relative to the target destinations. Knowledge of the absolute positions of the other agents is not required to compute the individual constraints.

The local agent position constraints proposed in (6.6) can be incorporated in many different motion control or trajectory planning applications. For instance, because the position bounds provide collision avoidance guarantees without fully specifying position trajectories, they can be included in predictive optimisation approaches with objective functions that do not consider...
the coordination among the agents. The resulting optimisation problems can incorporate additional constraints such as avoidance of other objects. The sets in (6.6) are convex as they are constructed from distance functions. This allows to bypass an extra procedure for convex approximation of a safe region which is typically required in model predictive motion control, see [107] for instance. For specific choices of the applied distance functions, e.g., the 1-norm or the infinity-norm distances, the constraints are linear in the position variables and can be efficiently encoded. The time-varying position bounds can also be used to verify that motion control strategies derived from simplified assumptions do not result in collisions in practice or in higher fidelity simulations.

### 6.3 Collision Avoidance Guarantee via CBF

Consider a set of mobile agents $\mathcal{A}$. Let the dynamics of agent $a \in \mathcal{A}$ be

$$
\begin{bmatrix}
\dot{p}_a \\
\dot{v}_a 
\end{bmatrix} =
\begin{bmatrix}
0 & I \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
p_a \\
v_a
\end{bmatrix} +
\begin{bmatrix}
0 \\
I
\end{bmatrix}
u_a,
$$

where $p_a \in \mathbb{R}^3$ is the position, $v_a \in \mathbb{R}^3$ is the velocity, and $u_a \in \mathbb{R}^3$ is the acceleration of agent $a$. Unless specified otherwise, we have dropped the notation for time in all the vectors, and assume an arbitrary given time $t$ throughout this section. Unless specified otherwise, the norm of a vector $\| \cdot \|$ refers to the Euclidean norm in this section.

Let the relative position of agents $a, a' \in \mathcal{A}$ be $\Delta p_{aa'} = p_a - p_{a'}$ and let their relative velocity be $\Delta v_{aa'} = v_a - v_{a'}$. In order to guarantee collision avoidance, the distance between any two agents $a, a' \in \mathcal{A}$, $a \neq a'$ must always satisfy

$$
\|\Delta p_{aa'}\| \geq s, \tag{6.7}
$$

where $s$ is the safety distance for all agents. Unlike the previous section, here we use the same safety distance for all agents.

Collision avoidance via a potential field method does not guarantee (6.7) always holds. This is because collisions would depend on the tuning of the repulsive force. However, there are three characteristics of the potential field method that are advantageous. Firstly, the method does not involve generating explicit conflict-free trajectories for the agents. Secondly, the nominal
control of the agents is only dominated by the collision avoidance method when agents are in danger of colliding. Thirdly, agents only require limited information from each other, i.e., an agent only requires knowledge of the positions and velocity of other agents but not their accelerations. These characteristics motivate the CBF approaches in this section. The following are assumptions on the limitations of agents that required in these approaches.

**Assumption 6.6.** Assume every agent \( a \in A \) has a velocity limit \( \|v_a\| \leq \Lambda_V \) and an acceleration limit \( \|u_a\| \leq \Lambda_A \), where \( \Lambda_V \) and \( \Lambda_A \) are the maximum velocity and maximum acceleration of all agents respectively.

The following assumption is for distributed computation amongst agents. As in the previous chapters, this means agents have limited knowledge of the information from other agents.

**Assumption 6.7.** Assume agent \( a \in A \) has knowledge of \( p_a \) and \( v_a \), as well as \( p_{a'} \) and \( v_{a'} \) from all other agents \( a' \neq a \).

### 6.3.1 Existing CBF Methods

We first discuss two CBF methods based on \([63, 64]\) and point out issues that arise when implementing them. Then, we propose a novel CBF method that addresses those issues.

#### 6.3.1.1 CBF Method with Feasibility Issues

We first summarise the control barrier approach in \([63]\) and discuss the issue of feasibility. The authors in \([63]\) set a pairwise constraint on the positions of agents \( a, a' \in A \), i.e.,

\[
\|\Delta p_{aa'}\| - \frac{(\bar{v}_{aa'})^2}{2(2\Lambda_A)} \geq s, \tag{6.8}
\]

where \( \bar{v}_{aa'} = \frac{\Delta p_{aa'}}{\|\Delta p_{aa'}\|} \Delta v_{aa'} \) is the scalar projection of \( \Delta v_{aa'} \) onto \( \Delta p_{aa'} \). This constraint is for any two agents to remain at a distance apart such that both agents applying maximum acceleration could bring the component of their relative velocity in direction of collision to zero before the safety distance is violated. We are only concerned about the case where the relative velocity brings the two agents closer together, so we rearrange (6.8) to get

\[
h_{aa'} = \bar{v}_{aa'} + \sqrt{4\Lambda_A(\|\Delta p_{aa'}\| - s)} \geq 0, \tag{6.9}
\]
where \( h_{aa'} \) is a function of relative position and relative velocity, i.e., \( h_{aa'} = f(p_{aa'}, p_{aa'}, v_a, v_{a'}) \).

This function characterises a set of tuples \( T_{aa'} = \{(p_{a}, p_{a'}, v_a, v_{a'}) | h_{aa'} \geq 0 \} \) of safe positions and velocities for the two agents. Any pair of agents \( a, a' \in A \) that have positions and velocities satisfying such tuples are guaranteed to satisfy the safety constraint in (6.7) since (6.8) \( \implies \) (6.7).

It remains to construct a barrier function that guarantees forward invariance of \( T_{aa'} \), i.e., if agents start with positions and velocities within the set, they remain at positions and velocities within the set for all time. From [51], the condition \( \dot{B}_{aa'} \leq \frac{\gamma}{h_{aa'}} \) guarantees forward invariance of the set \( T_{aa'} \), where \( B_{aa'} = \frac{1}{h_{aa'}} \), and \( \gamma > 0 \). Based on this, the authors in [63] derive a constraint on relative acceleration \( \Delta a_{aa'} \) of agents \( a, a' \in A \)

\[
-\Delta p_{aa'}^\top \Delta a_{aa'} \leq \gamma h_{aa'}^3 \| \Delta p_{aa'} \| - \frac{(\Delta v_{aa'}^\top \Delta p_{aa'})^2}{\| \Delta p_{aa'} \|^2} + \| \Delta v_{aa'} \|^2
\]

\[+ \frac{2\Lambda_A \Delta v_{aa'}^\top \Delta p_{aa'}}{\sqrt{4\Lambda_A(\| \Delta p_{aa'} \|- s)}}.
\]

(6.10)

In order to compute this constraint in the distributed setting given by Assumption 6.7, we can use a conservative approach and force an individual agent \( a \in A \) to be responsible for maintaining this safety distance alone, i.e., modifying the constraint to be

\[
-\Delta p_{aa}^\top u_i \leq \gamma h_{aa}^3 \| \Delta p_{aa} \| - \frac{(\Delta v_{aa}^\top \Delta p_{aa'})^2}{\| \Delta p_{aa'} \|^2} + \| \Delta v_{aa'} \|^2
\]

\[+ \frac{2\Lambda_A \Delta v_{aa}^\top \Delta p_{aa'}}{\sqrt{4\Lambda_A(\| \Delta p_{aa'} \|- s)}}.
\]

Each agent is constrained by \( |A| - 1 \) such pairwise constraints with the other agents. The set of allowable acceleration for an agent is an intersection of halfspaces and may be empty. Thus, this method does not guarantee feasibility, i.e., the agents may reach situations such that they no longer have actions that guarantee collision avoidance. Figure 6.1 shows an example of an empty intersection of halfspaces. Agent \( a \in A \) has no allowable acceleration, not even zero acceleration is an allowed action. We summarise the approach in [64] to address this issue of feasibility.
Figure 6.1: Illustration of empty set of allowable acceleration in $\mathbb{R}^2$. The blue shaded region represents the halfspace of possible accelerations agent $a \in \mathcal{A}$ can take to maintain the safety distance with agent $a'' \in \mathcal{A}$. The orange shaded region represents the halfspace of possible accelerations agent $a$ can take to maintain the safety distance with agent $a' \in \mathcal{A}$.

6.3.1.2 CBF Method with Decoupling Issues

The main idea here is to ensure that maximum braking by both agents always guarantees the safety distance is not violated, thus enabling agents to take this action should the intersection of halfspaces from the constraints on acceleration be empty. In a similar fashion to above, the authors in [64] derive a condition that guarantees safety between agents $a, a' \in \mathcal{A}$, i.e.,

$$h_{aa'}' = \left\| \Delta p_{aa'} + \frac{v_a}{4\Lambda} + \frac{v_{a'}}{4\Lambda} \right\|^2$$

$$- \left( s + \frac{\|v_a\|^2}{4\Lambda} + \frac{\|v_{a'}\|^2}{4\Lambda} \right)^2 \geq 0,$$

where once again $h_{aa'}' = f(p_a, p_{a'}, v_a, v_{a'})$.

This constraint is for any two agents to remain at a distance apart such that if they were to apply maximum braking both agents would come to a stop with a distance larger than the safety distance separating them. By constructing a control barrier function from (6.11), with $B_{aa'} = \frac{1}{h_{aa'}'}$, the authors in [64] derive a set of constraints on the acceleration of agents $a, a' \in \mathcal{A}$.
so that they do not violate the safety distance between each other

\[ \Omega_{aa',1} u_a + \Omega_{aa',2} u_{a'} \leq 2\Delta p_{aa'} \Delta v_{aa'} + \frac{\|v_a\|}{2\Lambda} \Delta v_{aa'} v_a - \frac{\|v_{a'}\|}{2\Lambda} \Delta v_{aa'} v_{a'} + \gamma (h'_{aa'})^3, \]

(6.12)

where

\[ \Omega_{aa',1} = \frac{\|v_a\|\|v_{a'}\|}{8\Lambda^2} v_a' + \frac{v_a v_{a'}}{8\Lambda^2\|v_a\|} v_a + \frac{\Delta p_{aa'} v_a}{2\Lambda\|v_a\|} v_a - \frac{\|v_{a'}\|\Delta p_{aa'}}{2\Lambda} + \frac{s v_a}{\Lambda} + \frac{\|v_{a'}\|^2 v_a}{4\Lambda^2}, \]

\[ \Omega_{aa',2} = \frac{\|v_a\|\|v_{a'}\|}{8\Lambda^2} v_a' + \frac{v_a v_{a'}}{8\Lambda^2\|v_{a'}\|} v_{a'} + \frac{\Delta p_{aa'} v_{a'}}{2\Lambda\|v_{a'}\|} v_{a'} + \frac{\|v_a\|\Delta p_{aa'}}{2\Lambda} + \frac{s v_{a'}}{\Lambda} + \frac{\|v_a\|^2 v_{a'}}{4\Lambda^2}. \]

In order to compute this in the distributed setting given by Assumption 6.7, the authors in [64] set agent \( a \in A \) to be responsible for the portion of the contraint in (6.12) that contains \( \Omega_{aa',1} \), i.e.,

\[ \Omega_{aa',1} u_a \leq \Delta p_{aa'} \Delta v_{aa'} + \frac{\|v_a\|}{2\Lambda} \Delta v_{aa'} v_a + \frac{1}{2} \gamma (h'_{aa'})^3. \]

(6.14)

Once again, each agent \( a \in A \) is constrained by \( |A| - 1 \) pairwise constraints of the form in (6.14), one for every the other agent. The constraint in (6.11) ensures agents can always apply maximum braking, slow to a stop and not violate the safety distance (6.7). Therefore, even if the \( |A| - 1 \) pairwise constraints result in the set of allowable accelerations being empty, there is always one action agents can take to ensure the safety constraint in (6.7) remains satisfied, which is to apply maximum braking.

Let \( S_a = \{ u_a \in \mathbb{R}^3 | u_a \text{ satisfies (6.14), } a' \in A \} \) be the set of allowable accelerations for agent \( a \in A \) based on positions and velocities of all agents in \( A \). The authors in [64] propose the hybrid scheme

\[ u'_a \in \begin{cases} S_a & \text{if } S_a \neq \emptyset, \\ -\Lambda \frac{v_a}{\|v_a\|} & \text{if } S_a = \emptyset \text{ and } \|v_a\| \neq 0 \\ 0 & \text{if } S_a = \emptyset \text{ and } \|v_a\| = 0. \end{cases} \]

(6.15)
We now examine a flaw in this approach that leads to agents losing the safety guarantee, by arguing that satisfaction of (6.14) does not imply satisfaction of (6.12) when the hybrid scheme in (6.15) is applied.

Remark 6.1. Consider three agents \( a, a', a'' \in A \). Assume \( S_a = \emptyset \) and \( S_{a'} = \emptyset \) but \( S_a \neq \emptyset \), i.e., agents \( a' \) and \( a'' \) are in braking mode. From the perspective of agent \( a \), it does not have access to this information as the braking of agent \( a' \) and \( a'' \) is not communicated to agent \( a \). When going from a centralised to distributed approach, i.e., going from the constraint in (6.12) to the constraint in (6.14), we also however made the assumption from the perspective of agent \( a \) that the other two agents are working to satisfy the portion of (6.12) related to \( \Omega_{aa',2} \) and \( \Omega_{aa'',2} \). Due to the application of the hybrid scheme and the agents entering braking mode unannounced, the behaviour of agents \( a' \) and \( a'' \) may not be as expected by agent \( a \). From the perspective of agent \( a \), this leads the loss of the safety guarantee as the constraint in (6.12) is in actual fact violated despite the fact that the constraint in (6.14) is not violated.

Based on the above argument, the hybrid scheme in (6.15) using the constraint in (6.12) only guarantees safety when the problem is limited to two agents, or if agents are able to communicate that they are entering braking mode instantaneously to other agents.

Figure 6.2 shows an illustrative example of an instance where the safety guarantee is violated by three agents. Consider three agents \( a, a', a'' \in A \) with colinear positions and assume that \( h'_{aa'} = 0 \) and that agents \( a \) and \( a' \) are moving at maximum velocity in the direction of agent \( a'' \), i.e., \( v_a = v_{a'} = -\Lambda V \frac{\Delta p_{aa''}}{\|\Delta p_{aa''}\|} \). Since \( h'_{aa'} = 0 \) and \( v_a = v_{a'} \), \( \|\Delta p_{aa'}\| = s + \frac{\Lambda^2}{2\Lambda_A} \). Assume agent \( a \) does not accelerate, i.e., its input acceleration is \( 0 \in S_a \). If \( S_{a'} = \emptyset \) and \( S_{a''} = \emptyset \), then agent \( a' \) must begin to brake. Once the distance between \( a \) and \( a' \) reduces below \( s + \frac{\Lambda^2}{2\Lambda_A} \), there is no possible action for agent \( a \) to take to avoid violating the safety distance with agent \( a' \).

\[
\begin{align*}
\|\Delta p_{aa'}\| &= s + \frac{\Lambda^2}{2\Lambda_A} \\
\|v_a\| &= \Lambda V \\
u^*_a &= 0 \\
u^*_{a'} &= -\Lambda A \frac{v_{a'}}{\|v_{a'}\|} \\
u^*_{a''} &= -\Lambda A \frac{v_{a''}}{\|v_{a''}\|}
\end{align*}
\]

Figure 6.2: Demonstration of the hybrid controller in (6.15) with the condition in (6.12) resulting in the violation of the safety distance when there are three agents. The braking of agent \( a' \) is inadvertently an acceleration in the direction of agent \( a \).
In order to use the hybrid scheme in (6.15), we need to derive an acceleration constraint for agent \( a \in A \) that does not depend on the acceleration of other agents and only on the position and velocities, i.e., as done in (6.10). However, the acceleration constraint must also allow maximum braking as a valid action to address the feasibility issue of (6.10). We introduce a novel method that takes these two issues into consideration.

### 6.3.2 CBF Method to Address Feasibility and Decoupling Issues

We propose a novel approach that addresses the issues of feasibility and decoupling from the previous section. Once again, consider two agents \( a, a' \in A \). We set a new constraint on the relative positions of the two agents

\[
\| \Delta p_{aa'} \| \geq s + \frac{\| v_a \|^2}{2\Lambda_A} + \frac{\| v_{a'} \|^2}{2\Lambda_A}.
\]

(6.16)

Intuitively, this constrains the distance between any two agents to be more than the sum of the distance it would take for both agents to apply maximum braking to become stationary from their current velocities and still have a large enough distance between them once they are both stationary. Thus, their velocities can be reduced independent of each other, which is a prerequisite for using the hybrid controller in (6.15) that was overlooked in [64].

The following proposition shows how this new constraint addresses the situation in Figure 6.2. Agents travelling close together in the same direction do not have to start braking simultaneously to maintain the safety distance if they obey the constraint in (6.17). In fact, they can brake one after another and in any order.

**Proposition 6.4.** Given two agents \( a, a' \in A \), assume that at a given time \( t_0 \), the constraint in (6.16) holds and that \( v_a(t_0) = v'_{a'}(t_0) = \Lambda_V \). Without loss of generality, assume agent \( a \) starts braking with maximum deceleration at \( t_0 \) and becomes stationary at \( t_1 \) and agent \( a' \) starts braking with maximum deceleration at \( t_1 \) and becomes stationary at \( t_2 \), where \( t_0 \leq t_1 \leq t_2 \). The safety constraint between the two agents given by (6.7) is never violated at any time \( t \in [t_0, t_2] \).

**Proof.** At time \( t_0 \), \( \| \Delta p_{aa'}(t_0) \| \geq s + \frac{\| v_a(t_0) \|^2}{2\Lambda_A} + \frac{\| v'_{a'}(t_0) \|^2}{2\Lambda_A} \) so the safety distance is not violated. Let \( \| \Delta p_{aa'}(t_0) \| = s + \frac{\| v_a(t_0) \|^2}{2\Lambda_A} + \frac{\| v'_{a'}(t_0) \|^2}{2\Lambda_A} + \varepsilon \), where \( \varepsilon \geq 0 \). In the worst case, the braking of agent \( a \) results in a decrease in the magnitude of relative position, i.e., at time \( t_1 \), the distance between \( a \) and \( a' \) reduces to \( \| \Delta p_{aa'}(t_1) \| = s + \frac{\| v'_{a'}(t_1) \|^2}{2\Lambda_A} + \varepsilon \), where \( \frac{\| v'_{a'}(t_1) \|^2}{2\Lambda_A} \) is the maximum distance...
travelled by agent $a$ before it becomes stationary. At time $t_1$, $\|\Delta p_{aa'}(t_1)\| \geq s + \frac{\|v_{a'}(t_1)\|^2}{2\Lambda_A}$ so the safety distance is not violated in the time interval $[t_0, t_1]$. In the worst case, the braking of agent $a'$ results in a decrease in the magnitude of relative position, i.e., at time $t_2$, the distance between $a$ and $a'$ reduces to $\|\Delta p_{aa'}(t_2)\| = s + \varepsilon$, where $\frac{\|v_a'(t_2)\|^2}{2\Lambda_A}$ is the maximum distance travelled by agent $a'$ before it becomes stationary. At time $t_2$, $\|\Delta p_{aa'}(t_2)\| \geq s$ so the safety distance is not violated in the time interval $[t_1, t_2]$ either.

We introduce a so-called apparent relative velocity, i.e., let the apparent relative velocity between agents $a$ and $a'$ be $\Delta v_{aa'} = -\frac{\Delta p_{aa'}}{\|\Delta p_{aa'}\|}(\|v_a\| + \|v_{a'}\|)$. By construction, the scalar projection of the apparent relative velocity onto $\Delta p_{aa'}$ is $\bar{v}_{aa'} = -(\|v_a\| + \|v_{a'}\|)$. We apply the apparent relative velocity to insert an intermediary term between the inequality in (6.16)

$$\|\Delta p_{aa'}\| \geq s + \frac{(\bar{v}_{aa'})^2}{2\Lambda_A} \geq s + \frac{\|v_a\|^2}{2\Lambda_A} + \frac{\|v_{a'}\|^2}{2\Lambda_A}. \quad (6.17)$$

The key observation is that this constraint now has a similar form to that in (6.8), with the apparent relative velocity replacing the relative velocity. Note that the denominator is $2\Lambda_A$ to reflect the braking of an individual agent. Figure 6.3 illustrates the inequalities in (6.17). This is of course due to the fact that the squared apparent relative velocity is an upper bound on the sum of squared individual velocities of the two agents, i.e., $(\|v_a\| + \|v_{a'}\|)^2 \geq \|v_a\|^2 + \|v_{a'}\|^2$.

![Figure 6.3: Illustration of apparent relative velocity being an upper bound on the sum of individual velocities of two agents. Using apparent relative velocity forces agents to maintain larger separation than in (6.16).](image)

The reason we insert the apparent relative velocity into the problem is that given the similar form of (6.17) to (6.8), it remains to replace the actual relative velocity with the apparent velocity in (6.9)

$$h_{aa'}'' = \bar{v}_{aa'} + \sqrt{2\Lambda_A(\|\Delta p_{aa'}\| - s)} \geq 0. \quad (6.18)$$
By performing the same procedure of differentiation on (6.18) as was done on (6.9) to get (6.10), we now get

\[
-\Delta p_{aa}^\top \Delta a_{aa} \leq \gamma (h_{aa}')^3 \| \Delta p_{aa} \| - \frac{((\Delta v'_{aa})^\top \Delta p_{aa})^2}{\| \Delta p_{aa} \|^2} + \| \Delta v'_{aa} \|^2 + \frac{\Lambda_A (\Delta v'_{aa})^\top \Delta p_{aa}}{\sqrt{2\Lambda_A (\| \Delta p_{aa} \| - s)}}.
\]

where we once again replace the true relative velocity with apparent relative velocity, i.e, the magnitude and sign of relative velocity between agents are always such that it appears as if they were moving straight at each other. Note, there is also a difference in the scalers, i.e., the term $2\Lambda_A$ replaces $4\Lambda_A$. In this case, $\Delta a_{aa'}$ represents the allowable change in relative velocity from $\Delta v'_{aa'}$ in order to have agent positions and velocities that retain satisfaction of (6.18). Similarly, for the distributed case we have

\[
-\Delta p_{aa}^\top u_a \leq \gamma (h_{aa}')^3 \| \Delta p_{aa} \| - \frac{((\Delta v'_{aa})^\top \Delta p_{aa})^2}{\| \Delta p_{aa} \|^2} + \| \Delta v'_{aa} \|^2 + \frac{\Lambda_A (\Delta v'_{aa})^\top \Delta p_{aa}}{\sqrt{2\Lambda_A (\| \Delta p_{aa} \| - s)}}.
\]  

(6.19)

The hybrid controller is applied in the same way with the new constraints.

### 6.3.3 Agent and Obstacle Collision Avoidance

This approach of creating an apparent relative velocity extends to collision avoidance of agents and static obstacles. We consider an agent $a \in A$ attempting to avoid an obstacle $C$, which is represented as a convex polytope. Let $c \in C$ be the closest point in $C$ to agent $a$. Note that finding the closest point on a polytope from an arbitrary point can be done by solving a quadratic program.

Let the apparent velocity of $c$ be $v_c = 0$. The apparent relative velocity between agent $a$ and the closest point $c$ is $\Delta v'_ac = -\frac{\Delta p_{ac}}{\| \Delta p_{ac} \|}(\| v_a \| + 0)$ and its scalar projection is $\bar{v}'_{ac} = -\| v_a \|$. Once again, we devise a constraint

\[
\| \Delta p_{ac} \| \geq s + \frac{(\bar{v}'_{ac})^2}{2\Lambda_A} = s + \frac{\| v_a \|^2}{2\Lambda_A}.
\]

Intuitively, agent $a$ is constantly trying to avoid its reflection in the surface of the obstacle.
We apply the constraint in (6.19) using this apparent relative velocity to derive constraints on the acceleration of an agent for it to avoid static obstacles represented as convex polytopes.

6.4 Conclusion

We derived local time-varying position constraints for agents that are assigned to move to different target destinations. The conditions for which satisfaction of the constraints guarantees that collisions are avoided are the same as the conditions for uniqueness of the LexBAP and SeqBAP solutions found in Chapter 5, i.e., all $k$th order robustness margins must be strictly positive. Since all LexBAP solutions are also BAP solutions, the introduced method allocates targets to agents such that the largest distance between an agent and its destination is minimised, thus fulfilling the assignment objective of the asset protection scenario in Section 1.1 as well as the need for collision avoidance. There may be agents that are not assigned to any task but all agents are constrained in order to maintain a minimal safety distance from any assigned agent. The parameters of the local constraints are obtained from the SeqBAP and depend only on the distances between agent initial positions and destinations. The local constraints define non-empty convex sets of safe positions for every agent if the optimising assignment is sufficiently robust to changes in the distances relative to the size of the agents. We defined a method to quantify the robustness that is related to the price of absence and observed that the more robust the assignment is, the larger the regions of safe positions are. Thus, there is a connection between this work and the sensitivity analyses of assignment problems in [34, 95]. The constraints can be constructed in polynomial time and with distributed computation where agents only access limited information on other agents, as discussed in Chapter 5.

We also, as an alternative, derived local time-varying acceleration constraints for agents modelled as double integrators. This method is based on the work in [63, 64] and addresses some of the challenges facing those methods. Notably, the issue of feasibility in [63], where the set of allowable accelerations for agents is empty as well as the issue of coupling in [64], where satisfaction of the acceleration constraint of one agent is coupled to the acceleration of another agent rather than just the positions and velocities. In order to do this, we introduced a so-called apparent velocity where agents always assume that they are moving directly at each other when calculating their pairwise acceleration constraints. This idea of apparent velocity readily extends to agents avoiding collisions with convex polytopic obstacle, where the closest point on
the polytope is assumed to be another agent with zero velocity.
Chapter 7

Conclusion and Future Works

We explored the motivating asset protection scenario in Section 1.1, decomposing it into two separate phases with the overarching goal of computing both phases with computation distributed amongst agents. The first phase was the assignment problem, where agents are assigned to their tasks. The second phase was to provide collision avoidance guarantees for mobile agents as they move to complete their tasks. We also considered the link between the two phases and discussed how solving certain assignment problems like the LexBAP leads to intrinsic collision avoidance certificates.

The asset protection scenario was cast as typical example of the BAP, since it requires minimisation of the worst-case positioning time of agents. In Chapter 3, we investigated various structural properties of the BAP for the purpose of developing a distributed BAP algorithm. Specifically, important tools like a pruned edge set, a critical edge, price of absence, a matching-sublevel set and a bottleneck cluster were introduced and several properties of these tools were derived that were used in subsequent chapters. Furthermore, properties of the BAP itself were discussed and conditions for when merging two separate BAPs into a combined problem could be done efficiently were provided. Once again, augmenting paths were shown to play a vital role as these conditions were related to the existence of a specific augmenting path between particular agents and tasks in the combined BAP. There are several directions for future work related to structure in the BAP. We considered clustering agents in order to allow the efficient merging of two BAPs into a combined problem. However, there is potential future work in the investigation of how the sensitivity analysis using robustness margins or the price of absence affects these conditions on merging. This opens up the possibility of optimality in terms of partitioning of agents since we note that a critical edge is not necessarily unique. The choice of which critical edge is used in a bottleneck cluster as well as its price of absence may provide a measure for
comparing two bottleneck clusters formed from the same bipartite graph. Another interesting
direction might be to examine the structural properties of the SAP for merging two SAPs into a
combined problem. In the same way the merging problem lead to the warm-starting property of
PruneBAP, this direction would be valuable for considering how existing SAP algorithms may
also be amenable to warm-starting.

In Chapter 4, we introduced an algorithm, PruneBAP, for solving the BAP that was amenable
to having computation distributed amongst agents. As expected, when benchmarked against an
existing distributed greedy algorithm, PruneBAP was found to require longer on average to con-
verge to a solution. This is expected since the purpose of PruneBAP is to find an optimal BAP
solution while greedy algorithms do not produce optimal BAP solutions in general. However, it
is possible to exploit the assignment found via a greedy algorithm to warm-start PruneBAP, i.e.,
we can start with an assignment found via a greedy algorithm and use PruneBAP to iteratively
find better assignments with respect to the BAP objective function. Since each iterative assign-
ment is a valid pairing of agents to tasks, this allows the flexibility of trading-off computational
effort with optimality gap. The more computational effort we apply with PruneBAP, the lower
the weight of the largest edge in the resulting MCM. In the context of the asset protection
scenario, this means that it is possible to terminate PruneBAP as soon as agents are assigned
to intercept threats in such a way that they are all able to reach the interception points in time
to seduce the threats. We need to use the BAP to compare assignments, but it may not be
necessary to obtain an optimal solution in this case.

One possibility for future work is applying PruneBAP for solving BAPs with heterogeneous
agents, where certain tasks can only be carried out by a subset of agents. The capability of an
agent to carry out a particular task is represented by the presence or lack of an edge connecting
that agent to the task. Once again, making use of the fact that PruneBAP can be warm-started
with an arbitrary assignment and does not require a connected bipartite graph as an input,
PruneBAP can be applied to solve this type of problem. The challenge in this case is to develop
a method for comparing the costs of heterogeneous agents in an equitable manner since the
function for calculating the cost of one agent to complete a task may not be the same as the
function for calculating the cost of a different agent to complete that task.

Another one of the main contributions in Chapter 4 was to search for an augmenting path with
distributed computation. Aside from their use for solving the BAP, augmenting path searches
are applied in many approaches for solving other combinatorial optimisation problems. These
include the problems of finding an MCM of a bipartite graph, finding the maximum cardinality intersection of a matroid, and finding the minimum of submodular functions, see [84, 108–110].

The distributed augmenting path search methods derived in this thesis have the potential to provide benefits for solving other combinatorial optimisation problems. The extension of the methods to other applications is the subject of future work.

Inter-agent collision avoidance was tauted as being one of the reasons for not just solving the BAP but further solving a special case of it, the LexBAP. Given this motivation, we introduced a greedy reformulation of the LexBAP in Chapter 5, i.e., the SeqBAP. We derived conditions for when this particular greedy approach produces an assignment that is also an exact solution to the LexBAP. A distributable algorithm, PruneSeq, was introduced to produce assignments according to the SeqBAP. PruneSeq relies on the warm-starting property of PruneBAP and it exploits the price of absence of edges to produce a solution with lower worst-case complexity than both an exact LexBAP algorithm and a naive SeqBAP algorithm that simply solves \( n \) sequential BAPs. In exploiting this structure in the LexBAP, the trade-off is that the explicit price of absence is not computed; only the positivity of the price of absence of an edge is checked. In order to compute the \( k \)th order robustness margins required for producing the safe position sets for collision avoidance in Chapter 6, an additional BAP must be computed for every \( k \)th order bottleneck edge. Since the price of absence of a bottleneck edge is related to the matching-sublevel set, one future direction would be find bounds on the \( k \)th order robustness margins rather than their explicit values, which is a cheaper prospect as that may only require augmenting path searches rather than computating additional BAPs.

Given the foreshadowing throughout this thesis on the connections between the concepts of bottleneck edges, critical edges, price of absence, robustness margin and bottleneck clusters, another future direction is a unifying analysis on the relationships between these key concepts in the BAP as well as their relationships to the sensitivity analysis in [95]. It would also be an interesting direction to discover the relationships between these concepts and existing centralised BAP algorithms such as the threshold algorithm [26–28].

In Chapter 6, we shifted focus to the second phase of the asset protection scenario and considered inter-agent collision avoidance. Assigning agents according to the BAP does not provide collision avoidance guarantees but we showed that solving the LexBAP, which is a special case of the BAP, does lead to safe sets of positions for agents to avoid colliding with each other based on robustness margins.
Alternatively, the collision avoidance guarantees can be considered separately to the assignment problem. We extended existing CBF approaches for guaranteeing collision avoidance between mobile agents. We derived constraints on the sets of accelerations that each agent was able to compute locally using only information about the positions and velocities of other agents and not their accelerations or full trajectories. One thing to note is that the authors in [63, 64] also provide a so-called minimally invasive controller, which allows agents to choose the acceleration from within their safe acceleration set that is closest, in a least-squares sense, to its nominal acceleration. This nominal acceleration may for example be related to the original trajectory of the agent for task completion without taking into consideration collision avoidance. Therefore, one possible future direction may be to investigate the performance of the CBF method derived in this thesis when using different schemes for choosing acceleration points within the safe set and compare that to when the minimally invasive scheme is applied. This is important because it is desirable to ensure that the time-varying safe sets remain as large as possible as an agent moves to complete its task, i.e., we wish to avoid having agents frequently entering braking mode and slowing down.
Bibliography


