# Heuristics for routing and scheduling of spatio-temporal type problems in industrial environments.

Jayanth Krishna Mogali

CMU-RI-TR-21-69

September 2021

*Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.*

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

**Thesis Committee:**
Stephen F. Smith, Chair
Willem-Jan van Hoeve
Maxim Likhachev
J. Christopher Beck, Univ. of Toronto

*To my family*

# Abstract

Spatio-temporal problems are fairly common in industrial environments. In practice, these problems come with different characteristics and are often very hard to solve optimally. So, practitioners prefer to develop heuristics that exploit mathematical structure specific to the problem to obtain good performance. In this thesis, we will present work on heuristics for 3 different classes of problems that commonly appear in industrial settings. The methods in this thesis generally view a region in space-time as a discrete resource, and so a recurring theme in this thesis is to view these problems from under a discrete optimization lens.

The first problem we will look at is the Multi-agent path-finding (MAPF) problem. MAPF is a useful abstraction for modeling pick-up and delivery problems within automated warehouses, where conflict-free paths for a set of robots need to be computed from pre-specified start and goal locations for robots while minimizing travel costs. A lot of prior work has been done by AI researchers, predominantly search techniques that are extensions of A* or conflict-driven binary search. Departing from existing approaches, we take a polyhedral view of the problem and propose a cutting plane procedure for computing a lower bound on the optimal solution to the MAPF problem, which is then incorporated into existing search-based methods as an evaluation function via Lagrangians. A novel feature of our approach lies in the cut generation procedure, which we show to be reminiscent of the template matching technique from image processing. Empirically, we show that our procedure outperforms a state-of-the-art search based approach for MAPF.

We then move on to problems such as the movement of products across the factory floor using conveyor belts, railway tracks etc. At a high level, products need to be transported on railway tracks along a pre-specified route, where each track can appear on the routes for different products, and so each track is like a shared resource. While a product occupies some track, no other product may occupy the same track. The overall objective is to transport all products to their respective destinations in the shortest time (makespan). Such problems are typically abstracted as Blocking Job Shop (BJS) problems. While makespan minimization for BJS is NP-Hard, most local search heuristics for BJS also suffer from a high computational cost per local search move. We propose several algorithmic improvements for existing local search procedures by leveraging structural properties (some existing and some new) of the BJS polytope. With those efficient updates, we are able to achieve new best results on a large fraction of existing BJS benchmark problems.

The third problem deals with situations where robots work in proximity. We present some work done for an aircraft manufacturer, where robots assemble the fuselage of an aircraft. The problem can be abstracted as a multiple-traveling salesman problem with collision and enabling constraints and makespan objective. Collision constraints prohibit robots from occupying certain regions simultaneously. Enabling constraints impose a weak ordering in which robots can perform tasks. We present 2 complementary heuristics which emphasize the routing and scheduling aspects in the problem to different degrees, with both heuristics relying on an efficient implementation of a scheduling sub-solver. We describe the overall system and present empirical results.

# Acknowledgements

First and foremost, I would like to thank my thesis adviser, Prof. Stephen Smith. I am very thankful for his flexibility and freedom, which enabled me to explore many research directions. His relaxed demeanor often calmed me down during many stressful situations. As a researcher, he has a very diverse portfolio and never shies away from complicated real-world problems. His research approach has been both inspirational and has significantly influenced my outlook on real-world problems. Thank you, Steve, for taking me in your lab since my early time as a Master's student, and for supporting me all along.

Next, I like to thank the rest of my committee, Prof. Willem-Jan van Hoeve, Prof. Christopher Beck, and Prof. Maxim Likhachev, for their time and valuable suggestions. In particular, I would like to thank Prof. Willem, with whom I was fortunate to collaborate on one of the chapters in this thesis. I am thankful for his guidance and generosity with his time.

CMU has been a wonderful place to do research. The large and diverse research conducted at the School of Computer Science has broadened my horizon. The excellent courses at the Tepper School of Business, and the seminar series in the Chemical Engineering department were probably the most important sources of information relevant to my thesis research. There are too many friends and researchers from these departments for me to name and thank. I am very grateful to the late Prof. Egon Balas, who has been an enormous influence on my research methodology. Very early during my PhD, I was unsure whether to work on Machine Learning or Discrete Optimization. It was his courses that helped me make up my mind to choose the latter. In all the 3 projects that are presented in this thesis, some of the ideas that he developed during his distinguished career manifests itself in some form or the other.

I would next like to thank my lab members. Thank you, Dr. Zachary Rubinstein, for all the fun chats and collaborations on multiple projects. Also, thank you for serving on my qualifier committees. Thank you, Dr. Laura Barbulescu, for collaborating with me on the Job Shop project. Thank you for your diligence and patience. Thank you, Prof. Joris Kinable, for your friendship, and for being a mentor, and for collaborating with me on the Quadbot project. Also, thank you for introducing me to Decision Diagrams. I thank Achal Arvind, Chung-Yao Chuang, Allen Hawkes, Isaac Isukapati, Richard Goldstein, Viraj Parimi, and Suryansh Saxena for their friendship and support. A special thanks to Achal Arvind, whose passion for combinatorial problems in industrial settings rubbed onto me. A special thanks to Viraj Parimi for his help in the preparation of my research papers and for always agreeing to play Table Tennis at whatever time of the day.

The Robotics Institute has been a fun department to be a part of. Many thanks to Suzanne Lyons Muth, Keyla Cook, and Barbara Jean Fecich for taking care of all the administrative work for the students. I have made several friendships these past seven years at Pittsburgh, which I hope will continue for a lifetime. I would like to thank Rajshekar Prabhakar, Sri Vignesh Rajendran, Gobinath Velliappan, Sandeep Konam, Saurabh Sharma, Ishani Chatterjee, Wenhao Luo, Dilip Krishnamurthy, Shashank Sripad, Adithya Pediredla, Sudershan Boovaraghavan, Neeha Dev, Akhila Kolapalli, Alankar Kotwal, Aparajita Sahoo, and Sravani Hotha.

I would also like to thank my first research mentor, Prof. Chandrasekhar Seelamantula

from IISc Bangalore, and my undergraduate adviser, Prof. Somasekhar Veeramraju from NIT Warangal.

Last but not least, I wish to express a deep sense of gratitude to my beloved parents and my brother for their endless support and love throughout my life that made me who I am today. I am forever indebted to them.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In the last few decades, robots are increasingly being employed to assist humans in industrial environments. As the physical space is shared by robots and humans, unsurprisingly, interactions between robots, and between robots and humans occur. Typically, in these problems, agents (robots, humans) must collaborate in order to complete the task, by sharing resources. The physical space in which the agents operate is an example of such a shared resource. The shared resources need to be allocated over time in order for the agents to complete their respective tasks, naturally the allocation of these resources leads to optimization problems. In this thesis, we present work on some specific optimization problems with a spatio-temporal flavor arising in industrial environments.

Spatio-temporal interactions between agents are not limited to industrial environments, they arise in other applications such as autonomous driving in urban environments etc... Industrial environments, unlike other application areas where spatio-temporal interactions arise, distinguish themselves by a few characteristics. Sometimes, the problems in industrial environments are the result of some manufacturing process, where the same procedure is executed day in and day out. Secondly, the systems involved in industrial problems are usually very reliable, and the environment in which the agents operate is usually almost fully controllable. On the flip-side, the spatio-temporal problems in industrial environments typically need planning over long horizons, so the corresponding optimization problems they generate tend to be large scale i.e., many variables and constraints. Owing to these distinguishing characteristics of spatio-temporal problems in industrial environments, the following objectives are very attractive from a practitioner's point of view while developing solution methodologies for such problems: (1) Computing an optimal or near-optimal solution for problems that are solved repeatedly and where reliable data exists; (2) Developing problem-specific procedures that exploit underlying mathematical structure to cope with problem scale.

We begin by briefly introducing all the spatio-temporal problems considered in this thesis. In the subsequent chapters, we provide precise mathematical formulations, literature review, contributions and results obtained for each of those problems. The three problems are:

(A) Multi-Agent Path-Finding (MAPF) problem: MAPF is a useful abstraction for modeling pick-up and delivery problems within automated warehouses [Wurman *et al.*, 2008; Stern, 2019]. Given pre-specified start and goal locations for a set of robots, the problem is to

generate conflict-free paths while minimizing an objective that aims to balance travel costs and makespan.

(B) Blocking Job Shop (BJS) problem: BJS is a useful abstraction for modeling movement of products\material across the factory floor using railway tracks, conveyor belts, etc..., see [Mati *et al.*, 2001a; Klinkert, 2001; Poppenborg *et al.*, 2012]. We use an example representative of BJS to explain the problem. At a high level, products need to be transported along pre-specified routes. At any given time, at most one product can be transported along a track. Even after reaching the end of the track, the product remains on the track until the downstream track on its route is not occupied by any other product. While a product occupies a track, all other products are blocked from accessing that track. The overall objective is to transport all products to their respective destinations in the shortest time (makespan). The example described above occurs in real world applications where there are no intermediate buffers to store the products during transportation.

(C) Multi-Robot scheduling with Blocking and Enabling (MRSBE) constraints: MRSBE is a new class of problems introduced in this thesis. We explain this problem class using a real-world example that is representative of MRSBE. The task of attaching the skin of an airplane fuselage requires fastening at multiple locations, and these locations are distributed across the surface in multiple clusters. The order in which the fastening locations can be serviced needs to obey a weak ordering specified in terms of enabling constraints (a generalization of the precedence constraint). A robot is used to reach each fastening location, and we are given a set of robots to complete attaching the skin to the fuselage. The overall problem is to allocate fastening locations to robots and determine the order in which each robot visits their allocated locations, such that, at all times, robots do not collide with each other and obey enabling constraints. Solutions with lower makespan are preferred. At a high level, we can view MRSBE as a mTSP (multiple traveling salesman problem) with collision constraints.

MAPF, BJS, and MRSBE, all have one attribute in common. All three problems contain constraints that are referred to in scheduling literature as *Blocking* type constraints. Observe that in the MAPF problem, when a robot occupies a location, it blocks that location from being accessed by other robots until the moment in time the robot moves to a different location. Similarly, in the BJS problem, a track is blocked until the moment in time the product that is currently on it moves to a downstream track. Similarly, for the MRSBE problem, the space occupied by a robot is blocked to other robots to ensure that the robots do not collide. While it is difficult to imagine MAPF without blocking, it is not the same for BJS and MRSBE. Without blocking, BJS reduces to the well studied classical Job Shop problem.

## 1.1 Philosophy pursued in this work

### 1.1.1 Discrete optimization view of the problems

In this thesis, we adopt the view that regions in space-time are discrete resources. These resources need to be allocated to agents and scheduled for solving the optimization problem.

By taking such a view, we can model all three problems as discrete optimization problems. For example, in the MAPF problem, locations in the warehouse are viewed as resources that the robots need to share to get to their goal locations. In BJS, the machines (conveyor belts, railway tracks, etc...) needed to process the jobs are the resources. In MRSBE, the physical space in which the robots operate is the shared resource. Viewing spatio-temporal problems through a discrete optimization lens is quite common in optimization literature, see [Hall and Sriskandarajah, 1996] for other examples.

### 1.1.2 Solution approach

The spatio-temporal optimization problems considered in this work are all NP-Hard in the strong sense. One can attempt to obtain the optimal solution to those problems by modeling and solving them using Mathematical Programming technologies such as Mixed Integer Linear Programming, and Constraint Programming. However, obtaining optimal solutions for realistically sized problem instances for our problem classes within practical time limits has been challenging[1]. While these solvers can still be used to obtain good quality solutions using their built-in generic primal feasibility procedures, the quality of the solutions have not generally been competitive when compared to dedicated heuristics developed for each problem separately. Practitioners commonly forgo computing the optimal solution, instead, they develop a heuristic that is specific to each problem type to obtain good quality solutions. This should not be suprising, since the three problems have typically been used to model different applications in the real world, and so the problem characteristics vary. Typically, the state of the art approaches for these problems have very little in common. Often, the heuristics developed exploit the underlying mathematical structure in the problem class to enhance search performance in different ways. In this thesis, we adopt a similar philosophy for the BJS and MRSBE problem. For the MAPF problem, though, we try to speed up an exact search algorithm, through the use of a heuristic lower bounding procedure that exploits structure.

## 1.2 This thesis

In this section, we briefly glimpse through our work in this thesis and identify its main contributions.

Previous work on MAPF has largely been focused on developing methods that compute the optimal solution to the problem, using best first search approaches. The performance of these approaches, crucially, depends on the use of good quality node evaluation functions. However, the problem of developing good quality node evaluation function has received very little attention in the MAPF literature. In this work, we propose a polyhedral cutting plane procedure for computing a lower bound on the optimal solution to the MAPF problem. The lower bound will serve as the node evaluation function for best first search procedures. By incorporating our node evaluation function into a state-of-the-art search based approach for

[1] We will elaborate on the challenges for each of the spatio-temporal problems separately, in their respective chapters.

MAPF, called Conflict-based search, we were able to improve the performance of the search procedure.

Research effort on heuristics for the BJS has largely focused on tabu-search neighborhood based heuristics. Briefly, they are iterative methods, moving from one solution to another solution chosen from the neighborhood of the current solution. The main research challenge of such schemes is in providing a neighborhood definition, such that, near optimal solutions can be reached by only considering solutions within that neighborhood. Unfortunately, popular approaches for obtaining neighbors have relied on procedures that scale poorly with the dimensionality of the input problem. Due to the high computational complexity, the search performance has been very poor. In our work, we improve the computational complexity of obtaining neighbors by leveraging insights from the polyhedral view of the BJS problem. In certain cases, we also provide very efficient procedures for evaluating the makespan objective of neighboring solutions. With those improvements, we report new best results with the makespan objective on a large fraction ($\approx \frac{3}{4}$) of existing benchmarks for BJS.

The MRSBE problem as mentioned earlier is a new problem class. We formalize this general problem and analyze its complexity. We develop a system comprising of a scheduler and two neighborhood based local search operators for obtaining good quality solutions. Our scheduler is a hybrid approach that leverages constraint programming techniques to accelerate a heuristic search approach. We analyze the performance of the system developed on a set of synthetically generated problem instances, some of which capture the real world problem that motivated our work on MRSBE, and others that generalize to other application settings. We provide a differential analysis of our local search procedure and provide a comparison to other approaches to demonstrate the efficacy of the proposed heuristic.

**Thesis Statement:** This thesis explores the general hypothesis that scheduling problems with blocking constraints can be effectively addressed by exploiting the problem structure that such constraints provide. For each of the three problem variants identified above, we aim to obtain our structural insights through polyhedral and decomposition techniques and demonstrate this hypothesis by producing new state of the art results.

## 1.3 Outline of the thesis

In **Chapter** 2, we introduce some helpful concepts and mathematical tools that have been employed in the development of the heuristics in this work. In **Chapter** 3, we present our work on the MAPF problem. Parts of this chapter previously appeared in [Mogali *et al.*, 2020]. In **Chapter** 4, we present our work on the BJS problem. This chapter is based on our paper [Mogali *et al.*, 2021a]. In **Chapter** 5, we present our work on the MRSBE problem. This chapter is based on our paper [Mogali *et al.*, 2021b]. In **Chapter** 6, we provide a summary of the work completed in this thesis, more general takeaways and future research directions. The contents in Chapters 3, 4 and 5 are not connected to each other, and so a reader interested in a specific problem can directly read the appropriate chapter he\she is interested in.

# Chapter 2

# Mathematical tools used in the heuristics

In this chapter, we introduce the tools that have been employed for the development of the heuristics. The reader is assumed to have basic familiarity with Linear and Integer Programming [Conforti *et al.*, 2014]. The tools that have been used for the MAPF problem are described in Section 2.1. The tools used for the work on BJS and MRSBE problems are described in Section 2.2.

## 2.1 Tools for MAPF

Recall from Section 1.2, our work on the MAPF problem involves developing a node evaluation function for search based methods based on polyhedral approaches. Central to our approach will be the construction of polyhedral relaxations to the projections of the MAPF polytope, which in our approach is done using Decision Diagrams (DDs). DDs are introduced in Section 2.1.2. A reader familiar with DDs can skip Section 2.1.2. A second concept that we'll use for computing the node evaluation function is the Lagrangian dual, we introduce it in Section 2.1.1.

### 2.1.1 Lagrangian dual

Consider an optimization problem of the form shown below:

$$P : \min_{x \in \{0,1\}^n} \{c^\mathsf{T} x | x \in \mathcal{X}\} \tag{2.1}$$

where $\mathcal{X} \subseteq \{0,1\}^n$. $P$ is the prototypical optimization problem that we will frequently encounter in this thesis. It may be difficult to obtain an exact description of $\mathcal{X}$ for our problem, however we may have with us a set $\mathcal{X}' = \{x \in \mathbb{R}^n | Ax \leq b\}$, such that $\mathcal{X} \subseteq \mathcal{X}'$. Then a trivial lower bound to $Opt(P)$ i.e., the optimal value of $P$, can be obtained by solving the following optimization problem:

$$D : \max_{\lambda \geq 0} \min_{x} c^\mathsf{T} x + \lambda^\mathsf{T}(Ax - b) \tag{2.2}$$

By Lagrangian duality, it follows that $Opt(D) \leq Opt(P)$. For a detailed introduction to Lagrangian duality, the reader can refer to [Boyd *et al.*, 2004]. $Opt(D)$ will be referred to as

the Lagrangian dual lower bound to the problem. The Lagrangian dual is used in designing our node evaluation function for the MAPF problem.

### 2.1.2 Decision diagrams

Decision diagrams (DDs) have become an emerging tool for discrete optimization problems. Let $\mathcal{X} \subseteq \{0,1\}^n$ denote the feasible set of a 0-1 discrete optimization problem. Note that cardinality of $\mathcal{X}$ may be $\mathcal{O}(2^n)$. In this section, we will be interested in computing the optimal value to the problem shown in Equation (2.1), i.e., $Opt(P)$, using DDs. At a high level, DDs provide a compact representation of the members in $\mathcal{X}$, that will facilitate efficient computation of $Opt(P)$. For a thorough introduction to DDs for optimization, the reader can refer to [Bergman *et al.*, 2016].

Borrowing notation from [Davarnia and van Hoeve, 2020], a DD for $\mathcal{X}$, represented by $\mathcal{D}(\mathcal{X}) = (\mathcal{U}, \mathcal{A}, l)$ is a top-down directed multi-graph, where $\mathcal{U}$ represents the set of nodes, and $\mathcal{A}$ represents a set of arcs. $\mathcal{D}(\mathcal{X})$ can be decomposed into $n+1$ node layers $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{n+1}$, and $n$ arc layers $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$. The node layer $\mathcal{U}_1$ contains a single source node $s$, and the node layer $\mathcal{U}_{n+1}$ contains a single terminal node $t$. The tail of any arc in layer $j$ is connected to a node in $\mathcal{U}_j$, and its head to a node in $\mathcal{U}_{j+1}$. The label $l(a)$ of each arc $a \in \mathcal{A}_j$ , for $j \in [n]$, represents the value of $x_j \in \{0,1\}$. Each node in the layer $\mathcal{U}_j$ has a maximum out-degree equal to the size of the domain of variable $x_j$. Owing to such a structure in the DD, each $s-t$ path represents a point in $\{0,1\}^n$. A DD is said to be an exact representation of $\mathcal{X}$, if there is a 1:1 correspondence between the points represented by the $s-t$ paths in $\mathcal{D}(\mathcal{X})$ and members in the set $\mathcal{X}$. The reader may refer to [Bergman *et al.*, 2016], for procedures to construct $\mathcal{D}(\mathcal{X})$. We provide the DD for a Knapsack problem in Figure 2.1 as an example.

Given a DD $\mathcal{D}(\mathcal{X})$ which exactly represents $\mathcal{X}$, we now describe the procedure for computing $Opt(P)$. The first step is to assign a cost to each arc in $\mathcal{A}$. Suppose $a \in \mathcal{A}_j$, then set the cost of $a$ to $l(a) \cdot c(x_j)$, where $c(x_j)$ is the value corresponding to $x_j$ in the vector $c$. Then $Opt(P)$ is simply the cost of the shortest $s-t$ path in $\mathcal{D}(\mathcal{X})$. Note that such a path can be computed in $\mathcal{O}(|\mathcal{A}|)$ complexity.

We make a few remarks about DDs. The DD to exactly represent $\mathcal{X}$ is non-unique. For example, previously in the description of $\mathcal{D}(\mathcal{X})$, we mapped arcs in $\mathcal{A}_j$ to the variable $x_j$, however we could have chosen a different mapping from arc layer to variable. A different mapping leads to a different DD than the one described earlier. It is possible for the new DD to be more compact, and thereby allowing us to compute the shortest $s-t$ path more efficiently. Secondly, note that in the description of $\mathcal{D}(\mathcal{X})$, we labeled each arc in $\mathcal{A}$ to the value of one variable in the set $\{x_j\}_{j=1}^n$, and so we ended up having $n$ arc layers. However, depending on the definition of $\mathcal{X}$, it may be possible to create a highly compact DD where we label each arc in $\mathcal{A}$ to the values of multiple variables in the set $\{x_j\}_{j=1}^n$. The main challenge with this new encoding is to ensure that there is a 1:1 correspondence between the members in the set $\mathcal{X}$, and the points corresponding to the $s-t$ paths in the DD. In general, it may be highly non-trivial to come up with such designs for every candidate $\mathcal{X} \subseteq \{0,1\}^n$. For the MAPF problem, fortunately, there is a very intuitive and simple way to construct a DD with such a compact encoding. We will elaborate on this in Chapter 3.

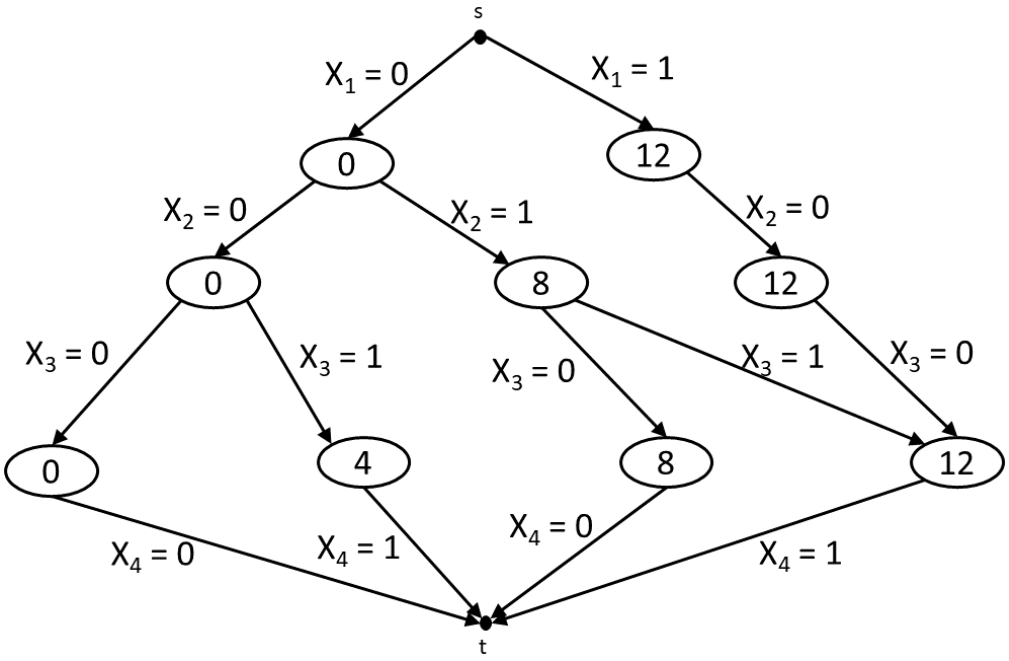



**Figure 2.1:** Decision diagram for a Knapsack constraint where $\mathcal{X} = \{x \in \{0,1\}^4 | 12x_1 + 8x_2 + 4x_3 + 2x_4 \leq 17\}$. The numbers marked in the nodes is the sum of the weights on each path originating from $s$ to that node.

## 2.2 Concepts and tools for BJS and MRSBE

The heuristics developed for both the MAPF and MRSBE are local search methods. In Section 2.2.1, we give a high level sketch as to how such a scheme is implemented. A tool that gets heavily used in the development of these heuristics is the Alternative graph [Mascis and Pacciarelli, 2002]. We introduce this tool in Section 2.2.2.

### 2.2.1 Local search framework

The heuristics procedures developed for both BJS and MRSBE are implemented as local search schemes. Local search procedures are iterative methods, which start from a feasible solution to the problem, and then iteratively move to a neighbor solution. To facilitate this movement across solutions, the designer of the heuristic must provide a way to obtain the neighbors of a solution. Typically, the neighborhood of a solution is defined as those solutions that are structurally very similar to the input solution. The definition for structural similarity is left to the designer to choose. The designer specified procedure to obtain a neighbor is loosely referred to as a “move" in local search terminology.

The pseudo-code for a typical local search scheme is provided in Algorithm 1. In line 6, the user has considerable choice in selecting the neighbor $s'$. If the neighborhood is not too large, then it may be possible to evaluate the objective of all neighbors in the neighborhood, and then pick the best neighbor among them. If the neighborhood is very large, an alternative

**Algorithm 1** Local search heuristic framework

1: **Given:**
2: Initial feasible solution $s$.
3: **Initialize:**
4: best solution = $s$.
5: **repeat**
6: $s'$ := Select a neighbor from the neighborhood of $s$.
7: **if** obj($s'$) $<$ obj(best solution) **then**
8: best solution := $s'$.
9: **if** Accept($s'$) **then**
10: $s := s'$
11: **until** Termination Criterion
12: **return** best solution

is to randomly select one of the neighbors from the neighborhood. In line 9, a criterion is used whether to accept $s'$ as the solution for the next iteration, and typically only used if the neighbor selected in line 6 is not necessarily the best neighbor in the neighborhood.

Algorithm 1 provides a high level framework of how local search schemes are typically implemented. Often in practice, designers choose to enhance its performance with the help of meta-heuristic procedures. Meta-heuristics are procedures that are embedded within the local search procedures, and they typically help in escaping local minima. Generally, these meta-heuristics operate by either modifying the way in which the worth of a neighbor is measured in line 6, or by explicitly specifying a scheme to decide whether to accept a neighbor in line 9. Tabu search [Glover and Laguna, 1998] is an example of the former, and Late acceptance [Burke and Bykov, 2017] is an example of the latter type. In our work, we use Tabu search in the heuristic for BJS, and Late acceptance for the MRSBE heuristic. We describe these procedures in greater detail in their respective chapters.

### 2.2.2 Alternative graphs

The Alternative graph (AG) is a special kind of temporal network for representing optimization problems with blocking constraints. The AG representation allows us to represent partial and complete solutions to those problems graphically, a property that is useful for describing neighborhoods in local search, and for developing constructive heuristics. The AG was first introduced in the context of BJS in Mascis and Pacciarelli [2002]. Below, we provide a slightly general version of the AG representation, to facilitate its usage for MRSBE heuristic as well. Before we describe the AG representation, we describe the class of optimization problems with blocking constraints for which the AG representation is used in this thesis.

Consider a problem with $M$ robots, and let $Seq_1, \ldots, Seq_M$ denote a sequence of landmarks for each robot, where $Seq_i \cap Seq_j = \emptyset$ if $i \neq j$. The robot needs to visit each landmark in its respective sequence, and in the same order as provided in the sequence. Corresponding to robot $i \in [M]$ and landmark $u \in Seq_i$, let $p_u$ denote the minimum time that robot $i$ needs to be physically located at landmark $u$. Generally the problems of our interest would be to com-

pute a schedule for the robots to visit all landmarks, subject to certain additional constraints that will specified shortly. Let $next_u$ denote the landmark following $u$ in $Seq_i$. Let $S_u$ denote the time at which robot $i$ first visits $u$. The first constraint that any feasible schedule to our problem must satisfy is:

$$S_u + p_u \leq S_{next_u}, \quad \forall i \in [M], \forall u \in Seq_i \tag{2.3}$$

We now introduce the blocking component in the problem. Blocking means that after $S_u$, robot $i$ continues to block the physical space robot $i$ requires for visiting $u$, until the moment in time robot $i$ moves to visit $next_u$ , i.e., until $S_{next_u}$. To ensure that robots do not collide while they block the physical space, we introduce a set of collision constraints $\mathbf{C}$. Each collision constraint (CC) in $\mathbf{C}$ is specified in terms of a tuple $(u, i, v, j)$ , where $i, j \in [M]$, and $u \in Seq_i, v \in Seq_j$. To ensure no collisions occur, we must ensure that the blocking intervals $[S_u, S_{next_u}]$ and $[S_v, S_{next_v}]$ do not overlap. So appropriately, the following set of constraints are introduced:

$$(S_v \geq S_{next_u}) \vee (S_u \geq S_{next_v}), \quad \forall (u, i, v, j) \in \mathbf{C} \tag{2.4}$$

Finally, we introduce enabling constraints (also known as OR constraints in literature [Möhring *et al.*, 2004]) into our problem. In order to service a landmark $u \in Seq_i$ by robot $i$, a set of enablers denoted by $\mathcal{E}_u \subseteq \bigcup_{j \in [M] \setminus i} Seq_j$ is specified. At least one among the enablers should be serviced prior to $u$ being serviced, or equivalently, the condition in Equation (2.5) must be satisfied. Equation (2.5) is referred to as the enabling constraint (EC) for $u$.

$$\bigvee_{v \in \mathcal{E}_u} (S_u \geq S_{next_v}) \tag{2.5}$$

So our problem also contains the following set of ECs:

$$\bigvee_{v \in \mathcal{E}_u} (S_u \geq S_{next_v}), \quad \forall u \in \bigcup_{j \in [M]} Seq_j \tag{2.6}$$

While there can be infinitely many schedules that are feasible to our constraints in Equations (2.3), (2.4), (2.5), generally in this thesis we will be interested in solutions that minimize makespan. A schedule that minimizes makespan to our problem can be obtained by solving an optimization problem (BJ-OPT) shown below:

**BJ-OPT**

$$\min_{S \geq 0} \max_{i \in [M]} \max_{u \in Seq_i} S_u + p_u$$

s.t. Equations $(2.3), (2.4)$ and $(2.5)$ are satisfied..

Before moving on to the AG representation, note that BJ-OPT is a fairly general problem class. For instance, we can represent the example for BJS involving movement of products across the factory floor on tracks in Chapter 1 as a BJ-OPT. To see why, for each railway track $j$ traversed by product $i$, we can introduce a landmark $l_i^j$ in $Seq_i$. If a track is used by more than one product, we simply introduce CCs between the appropriate pair of landmarks

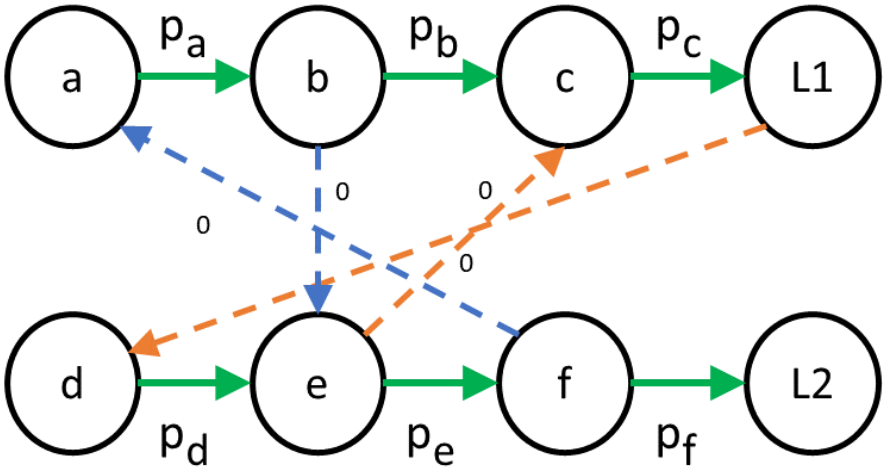



**Figure 2.2:** Alternative graph for a problem with 2 robots. Green arrows are the arcs in $F$. Alternative arcs, belonging to the same CC, have been marked with the same color.

representing the same track. For example, if track $j$ is used by products, $i, k$ we introduce the CC $(l_i^j, i, l_k^j, k)$. No ECs will be needed for this problem.

We are now ready to represent BJ-OPT using an AG. Basically, the temporal constraints in BJ-OPT are represented as arcs in the AG. For example, to represent a constraint of the form $S_v \geq S_u + l(u, v)$, we will have an arc $(u, v)$ with length $l(u, v)$ in our AG. Formally, an AG is a directed, weighted graph $G(\mathcal{N}, F, \mathcal{A}, En)$, where $\mathcal{N}$ represents the nodes in the graph. Each node in $\mathcal{N}$ corresponds to a landmark visited by a robot. $F$ is the set of arcs traversed by the robots in their respective itineraries. There is a 1:1 correspondence between arcs in $F$ and temporal equations shown in Equation (2.3). Before defining $\mathcal{A}$ and $En$, let us look at an example. Consider a problem with 2 robots, and let robot 1 visit landmarks $a, b, c$ in the same order. Similarly, for robot 2, let $Seq_2 = (d, e, f)$. The AG for this example is shown in Figure 2.2, with the arcs in $F$ shown in green. We have also included 2 dummy nodes, namely $L1, L2$, for reasons which will become self-explanatory when we describe $\mathcal{A}$.

For every CC in Equation (2.4), there are 2 disjunctions. Analogously, for every CC in $\mathbf{C}$, the AG provides a set of arcs to choose. For instance, for the CC $c = (v, i, w, j)$ the set of arcs to choose from is $\mathcal{A}_c = \{(next_v, w), (next_w, v)\}$, one arc corresponding to each disjunction. Both arcs are set to length 0, in order to match the temporal relation in those disjunctions. The arcs in the set $\mathcal{A}_c$ are called alternative arcs. Going back to the AG definition, we define $\mathcal{A} = \{\mathcal{A}_c\}_{c \in \mathbf{C}}$ as a set of pairs of arcs. For our earlier example, assume the CCs are $(a, 1, e, 2)$ and $(c, 1, d, 2)$. Observe in Figure 2.2, the alternative arcs corresponding to the two CCs present in our example are shown as dashed arcs.

For each node $u \in \mathcal{N}$, we define a set $En_u = \{(next_v, u) | v \in \mathcal{E}_u\}$, one arc corresponding to each enabler of $u$. The arcs in the set $En_u$ are used to represent the enabling constraint in Equation (2.5) for $u$. The length of each arc in the set $En_u$ is 0. The set $En_u$, like in the case of collision constraints, offers a set of alternative arcs to choose from. Going back to the AG definition, we define $En = \{En_u\}_{u \in \mathcal{N}}$. Sometimes, to distinguish between the alternative arcs in $\mathcal{A}$ and $En$, we will refer to the arcs in $\mathcal{A}$ as collision alternative arcs (CAA), and the

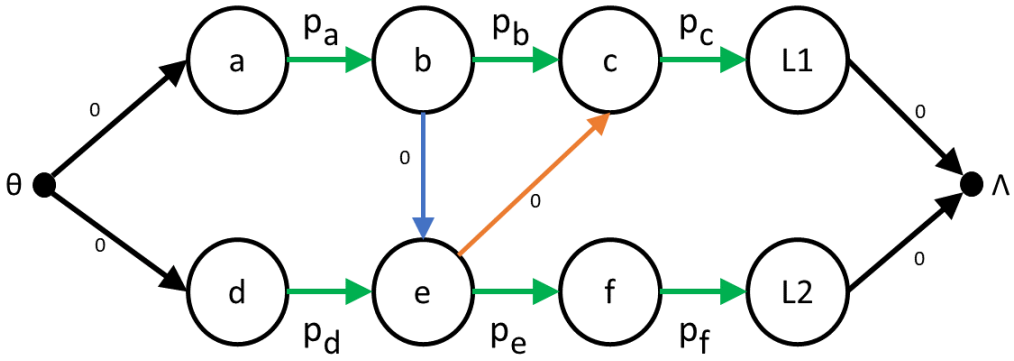



**Figure 2.3:** $\theta, \Lambda$ have been introduced for computing start times.

arcs in the set $En$ as enabling alternative arcs (EAA).

All that remains to be shown is how feasible solutions to BJ-OPT are represented graphically on the AG. In any feasible schedule to BJ-OPT, observe that exactly one of the two disjunctions corresponding to each $c \in$ C in Equation (2.4) can be active, i.e., evaluates to true. Likewise, at least one of the disjunctions in Equation (2.5) evaluates to true in any feasible solution to BJ-OPT. Let $Sol$ be some feasible solution to BJ-OPT, and denote the set of alternative arcs corresponding to the active disjunctions (computed using $Sol$) by $Cs$. In the case of an enabling constraint for a node (see Equation (2.5)), more than one disjunction can be active in $Sol$, but we will assume that $Cs$ only contains the arc corresponding to the disjunction that became active the earliest (ties resolved arbitrarily). The AG representation for solution $Sol$ is given by the graph $G(\mathcal{N}, F \cup Cs)$ , i.e., only arcs in the set $F \cup Cs$ are included in the AG. For our running example, consider a solution where for the CC $(a, 1, e, 2)$ the disjunction $(S_e \geq S_b)$ is active, and for the CC $(c, 1, d, 2)$ the disjunction $(S_c \geq S_e)$ is active. We represent such a solution on the AG with the appropriate selection in Figure 2.3. The makespan of the solution is given by the longest $\theta - \Lambda$ path on the graph.

# Part I

# Chapter 3

# Multi-Agent Path Finding

Multi-agent path finding (MAPF) is the problem of finding paths for individual robots (agents), given a start and end vertex for each robot on some layout (graph), such that the paths are spatio-temporally conflict-free and an objective resembling travel costs is minimized. MAPF has found many applications in warehouse logistic systems [Wurman *et al.*, 2008], robotics and video games [Silver, 2005]. MAPF is known to be NP-Hard to solve optimally on general graphs [Yu and LaValle, 2013b].

Many techniques have been proposed for solving the MAPF problem, and they come in different flavors. Current approaches for solving MAPF optimally can be broadly classified into search based methods [Sharon *et al.*, 2015; Wagner and Choset, 2011], constraint programming approaches [Gange *et al.*, 2019], and solution methods that rely on polyhedral techniques such as the integer programming formulation of [Yu and LaValle, 2013a], and the branch-cut-price method [Lam *et al.*, 2019]. A significant challenge for all these approaches is in developing strong lower bounding techniques required for proving optimality. From the point of view of search based methods, this typically translates into developing strong admissible heuristics. For polyhedral techniques, strong lower bounds are typically obtained by developing cutting plane procedures that tighten the linear description of the feasible region [Conforti *et al.*, 2014].

In this work, we propose a cut generation scheme that can be incorporated into search based methods for MAPF as well as polyhedral approaches. Incorporating cuts into techniques that use polyhedral approaches is common, but incorporating cuts into search based methods for MAPF is somewhat rare, and will be the focus of this work. The key technical tools that we will use in this work, are the Lagrangian Relax-and-Cut (LRC) procedure, and Decision Diagrams. Lagrangian Relax-and-Cut (LRC) [Escudero *et al.*, 1994; Lucena, 2005] is a popular Lagrangian based approach for obtaining strong lower bounds for discrete optimization problems, and is based on the idea of dualizing cuts.

Although the target application is the MAPF problem, this chapter simultaneously makes contributions to the literature on LRC, Decision Diagrams (DDs) and MAPF. All the contributions are presented in the context of the MAPF problem, but some ideas may be more broadly applicable. For the MAPF problem, we introduce 1) a new polyhedral approach for MAPF based on lower-dimensional 'templates' that can be translated spatio-temporally over the input graph, 2) a new cut generation scheme that utilizes decision diagram representations

of templates, 3) a Lagrangian Relax-and-Cut procedure to compute the lower bound, and 4) the idea of using the resulting lower bound as a node evaluation function in a conflict-based search (CBS) procedure. Experimental evaluation shows that our lower bounds can be very effective when the MAPF problem is more constrained.

The rest of the chapter is organized as follows. In Section 3.1 we introduce the MAPF problem and provide an Integer Programming formulation for the problem. In Section 3.2 we introduce the LRC scheme in the context of the MAPF problem. In Section 3.3 we present the cut generation scheme and introduce the necessary tools for performing cut generation, i.e., the projection polytope and the relaxation of the projection polytope. In Section 3.4 we introduce the DD representation of the relaxation to the projection polytope. The reader may choose to skip Section 3.4 without any implications for understanding the later sections. In Section 3.5 we briefly introduce CBS, and present how the LRC scheme is incorporated within CBS as a node evaluation function. In Section 3.6 we present the experimental setup and results. In Sections 3.7 and 3.8, we present future and related work, respectively. Parts of this chapter appeared previously in Mogali *et al.* [2020].

## 3.1 MAPF Problem Description

We consider the makespan constrained version of the MAPF problem in this chapter. We are given an undirected graph $G = (V, E)$, a set of $\mathbf{N}$ robots $\mathcal{R} = \{r_1, ..., r_{\mathbf{N}}\}$, and a makespan upper bound $\mathbf{T} \in \mathbb{Z}_+$, where $\mathbb{Z}_+$ represents the set of positive integers. Corresponding to each robot $r_i \in \mathcal{R}$, we are given a start vertex $s_i \in V$, and goal vertex $g_i \in V$. The task is to find a path for each robot, such that the robot paths do not conflict, while minimizing the cumulative sum of path costs. A path $p$ can be viewed as a function $p : \{0, 1, ..., \mathbf{T}\} \rightarrow V$, where $p(t)$ returns a vertex in $V$ corresponding to time $t$. If $\mathcal{P} = \{p_1, ..., p_{\mathbf{N}}\}$ is a set of robot paths containing a path for every robot, then $\mathcal{P}$ is feasible to the MAPF problem iff:

1. $p_i(0) = s_i$ and $p_i(\mathbf{T}) = g_i$, $\forall i \in \{1, 2, ..., \mathbf{N}\}$.
2. For each robot $r_i \in \mathcal{R}$ and for all $t \in \{0, 1, ..., \mathbf{T} - 1\}$, we require $p_i(t) = p_i(t + 1)$, or $(p_i(t), p_i(t+1)) \in E$. The robot either stays in its current vertex or moves to a neighbor.
3. To prevent vertex collisions, we require that $p_i(t) \neq p_j(t)$, for all pairwise distinct $i, j \in \{1, ..., \mathbf{N}\}$ and time $t \in \{0, 1, ..., \mathbf{T}\}$.
4. To prevent edge collisions, there should not exist a pair of robots $r_i, r_j$ and time $t \in \{0, 1, ..., \mathbf{T} - 1\}$, such that, $p_i(t) = p_j(t + 1)$ and $p_i(t + 1) = p_j(t)$.

We refer to any path $p_i$ satisfying 1 and 2 as a **start-end path** for robot $r_i$. The cost of start-end path $p_i$ is given by $c_i(p_i) = \sum_{t=0}^{\mathbf{T}-1} c_i(p_i(t), p_i(t + 1))$, where

$$c_i(p_i(t), p_i(t + 1)) = \begin{cases} 0, \text{ if } p_i(t) = p_i(t + 1) = g_i \\ 1, \text{ otherwise.} \end{cases} \tag{3.1}$$

Equation (3.1) assigns a cost of 0 if the robot is waiting at its goal vertex, else assigns a cost of 1. The goal of MAPF is to find a set of conflict-free robot paths $p_1, ..., p_{\mathbf{N}}$ that minimizes the objective $\sum_{i=1}^{\mathbf{N}} c_i(p_i)$. The objective we are minimizing is slightly different from the objective being minimized in other literature [Felner *et al.*, 2018; Li *et al.*, 2019a]. The difference is

that, in the other objective, a free wait at the goal for a given time step is allowed iff the robot remains at the goal during all later time steps until time $\mathbf{T}$. The techniques presented in this chapter can be adapted to work with the other objective, however, it complicates the presentation of the templates, and so we prefer to work with the objective in Eqn (3.1).

### 3.1.1 Integer programming model for MAPF

We next provide a multi-commodity flow based Integer Programming (IP) model for the MAPF problem, similar to [Yu and LaValle, 2013a]. The IP model will be useful in deriving valid inequalities for the lower bounding procedure we propose in later sections. Below, for any $n \in \mathbb{Z}_+$, we will use the notation $[n]$ to denote the set $\{1, 2, ..., n\}$.

The IP model will make use of a time expanded graph. The time expanded graph is an arc-weighted directed acyclic graph defined for each robot, where the nodes can be partitioned into $\mathbf{T}+1$ layers, and arcs into $\mathbf{T}$ layers. We shall denote the time expanded graph for robot $r_i$ by $F_i(N_i, A_i)$. Denote the nodes in layer $t \in \mathbb{Z}_+ \cup \{0\}$ by $N_i(t)$. Corresponding to each vertex $v \in V$, there exists a node in $N_i(t)$ if the shortest path from $s_i$ to $v$ in $G(V, E)$ uses at most $t$ edges, and the shortest path from $v$ to $g_i$ uses at most $\mathbf{T} - t$ edges. With a slight abuse of notation, we shall denote the node corresponding to the vertex $v \in V$ in $N_i(t)$ by $v_i^t$. Throughout this chapter, if a node from any of the graphs in the set $\{F_i\}_{i\in[\mathbf{N}]}$ is specified, we will assume that the vertex, time, and robot associated with that node can be deduced from our notation. Arcs in $A_i$ connect nodes between adjacent layers, with the tail of the arc connected to the node belonging to the lower indexed layer. Denote the arcs in level $t$ by $A_i(t)$. If $u_i^t \in N_i(t)$ and $v_i^{t+1} \in N_i(t+1)$, then there exists an arc $(u_i^t, v_i^{t+1}) \in A_i(t)$ iff $u = v$, or $(u, v) \in E$. We let $c_i(u_i^t, v_i^{t+1})$ denote the cost of the arc:

$$c_i(u_i^t, v_i^{t+1}) = \begin{cases} 0, \text{ if } u = v = g_i \\ 1, \text{ otherwise.} \end{cases} \tag{3.2}$$

There is a 1:1 correspondence between start-end paths for robot $r_i$ and $s_i^0 - g_i^{\mathbf{T}}$ paths in $F_i(N_i, A_i)$.

In describing our IP model, we use the following notation. For any node $v_i^t \in N_i(t)$, we denote $\delta^+_{F_i}(v_i^t)$ as the set of arcs in $A_i$ whose tail is the node $v_i^t$, and $\delta^-_{F_i}(v_i^t)$ as the set of arcs in $A_i$ whose head is the node $v_i^t$. For any vertex $u \in V$, we introduce the set $\overline{V}^t(u) = \{i \in [\mathbf{N}] | u_i^t \in N_i(t)\}$ for representing vertex collision constraints. For representing edge collision constraints, we define, for $(u, w) \in E$:

$$\overline{E}^t(u, w) = \{(i, j) \in [\mathbf{N}] \times [\mathbf{N}] | (u_i^t, w_i^{t+1}) \in A_i(t), (w_j^t, u_j^{t+1}) \in A_j(t), i \neq j\}.$$

For each robot $r_i \in \mathcal{R}$, and each arc $a \in A_i$, we introduce a binary variable $x(a) \in \{0, 1\}$ to indicate whether the robot $r_i$ traverses the arc $a$ in a feasible solution to the MAPF problem. Let $|A| = \sum_{i\in[\mathbf{N}]} |A_i|$, where $|A_i|$ denotes cardinality of set $A_i$. The 0-1 IP formulation for the MAPF problem is:

$$\underset{x\in\{0,1\}^{|A|}}{\text{minimize}} \sum_{i=1}^{\mathbf{N}} \sum_{a\in A_i} c_i(a)x(a) \tag{3.3}$$

$$\text{s.t.} \sum_{a \in \delta^+_{F_i}(s_i^0)} x(a) = 1,\ \forall i \in [\mathbf{N}] \tag{3.4}$$

$$\sum_{a \in \delta^-_{F_i}(u_i^t)} x(a) = \sum_{a \in \delta^+_{F_i}(u_i^t)} x(a),\ \forall i \in [\mathbf{N}],\ \forall t \in [\mathbf{T}-1], \forall u_i^t \in N_i(t) \tag{3.5}$$

$$\sum_{i \in \overline{V}^t(u)} \sum_{a \in \delta^{-1}_{F_i}(u_i^t)} x(a) \leq 1,\ \forall u \in V, \forall t \in [\mathbf{T}] \tag{3.6}$$

$$x(u_i^t, w_i^{t+1}) + x(w_j^t, u_j^{t+1}) \leq 1,\ \forall (u,w) \in E, \forall t \in \{0,1,...,\mathbf{T}-1\}, \forall (i,j) \in \overline{E}^t(u,w) \tag{3.7}$$

Equations (3.4) and (3.5) (a.k.a. flow balance constraints) ensure that a **start-end path** is chosen for every robot, while Eqns (3.6) and (3.7) prevent vertex and edge collisions respectively.

## 3.2 Lagrangian Relax-and-Cut

In this section, we will describe the LRC procedure for generating lower bounds in the context of the MAPF problem, but first we require a few preliminaries. Let $\mathbf{P}$ denote the MAPF polytope as shown below:

$$\mathbf{P} = \mathbf{conv}(x \in \{0,1\}^{|A|} | x \text{ satisfies } (3.4) - (3.7)), \text{ where } \mathbf{conv} \text{ denotes convex hull.} \tag{3.8}$$

Each vertex of $\mathbf{P}$ corresponds to a set of start-end paths, one for each robot in $\mathcal{R}$, such that the paths are mutually conflict-free. Denoting the objective in Eqn (3.3) by $c^\top x$, since $\mathbf{P}$ is the convex hull and the objective in Eqn (3.3) is linear, we can represent the optimization problem shown in Eqns (3.3) - (3.7) as:

$$Opt = \min_x \{c^\mathsf{T} x | x \in \mathbf{P}\} \tag{3.9}$$

A cut $a^\top x \leq b$ is valid for $\mathbf{P}$ if $\max_{x \in \mathbf{P}} a^\top x \leq b$. Given a valid cut $a^\top x \leq b$ for $\mathbf{P}$ and a point $y \notin \mathbf{P}$, the cut separates $y$ from $\mathbf{P}$ if $b < a^\top y$. Given a set of start-end paths with one path for each robot in $\mathcal{R}$, such that at least one pair of robots are conflicting, we will assume in this section that we are provided with an oracle that can generates cuts which separates such a set of paths from $\mathbf{P}$. We defer the development of such a cut generation oracle to a later section.

Next, we provide an overview of the LRC lower bounding procedure. LRC is an iterative procedure which at the $0^{th}$ iteration is provided with a relaxation $P_0$ of $\mathbf{P}$ as input, i.e., $P_0 \supseteq \mathbf{P}$. As the iterations progress, LRC computes tighter relaxations of $\mathbf{P}$, and so $P_0 \supseteq P_1 \supseteq \cdots \supseteq P_i \supseteq \cdots \supseteq \mathbf{P}$, where $P_i$ is the relaxation constructed at iteration $i$. Let

$$Opt(i) = \min_x \{c^\mathsf{T} x | x \in P_i\} \tag{3.10}$$

then clearly $Opt(0) \leq Opt(1) \leq \cdots \leq Opt(i) \leq \cdots \leq Opt$. $Opt(i)$ is the value of the lower bound obtained from LRC at iteration $i$. For obtaining $P_i$, LRC generates a set of

valid inequalities $E_i x \leq f_i$ for $\mathbf{P}$ using the cut generation oracle, and tightens $P_{i-1}$, i.e., $P_i = P_{i-1} \cap \{E_i x \leq f_i\}$. From a computational perspective, LRC shares similarity with other Lagrangian methods in the sense that, $Opt(i)$ is not computed using the formulation shown in the RHS of Eqn (3.10), instead it is computed using the Lagrangian dual. The procedure for generating $E_i x \leq f_i$ is also based on the Lagrangian dual, both these aspects are explained below.

The steps of the LRC procedure are provided in Algorithm 2. At iteration 0, the algorithm is initialized with $P_0 = \{x \in \{0,1\}^{|A|} | x \text{ satisfies Eqns } (3.4) - (3.5)\}$, i.e., the relaxation obtained by omitting all the collision constraints in the IP formulation in Section 3.1.1. Observe from the earlier discussion, $P_i = P_0 \underset{k<i}{\cap} \{E_k x \leq f_k\}$. So if we dualize all the inequalities generated by the oracle up to iteration $i$, we can alternatively compute $Opt(i)$ using the Lagrangian dual formulation shown in line 4 of Algorithm 2. $Opt(i)$ is the optimal objective value of the max-min problem shown. By dualizing the inequalities generated by the oracle, the inner minimization problem in line 4 reduces to an easy to solve min-cost flow problem for each robot, since the feasible region is just $P_0$. In line 5 we recover $\bar{x}_i$ by applying any shortest path algorithm on the time expanded graph (refer to $F$ in Section 3.1.1) for each robot, with the arc costs set according to the Lagrangian objective in line 4. Note that $\bar{x}_i$ corresponds to a set of start-end paths for the robots, and potentially contains conflicts. If $\bar{x}_i$ contains conflicts, the oracle is used to generate inequalities separating $\bar{x}_i$ from $\mathbf{P}$ as shown in line 11, and the inequalities generated (i.e., $E_i x \leq f_i$) are incorporated into the Lagrangian dual problem (line 4) in subsequent iterations.

Generating inequalities that separate $\bar{x}_i$ from $\mathbf{P}$ in line 11 perturbs the Lagrangian objective and is an attempt to obtain a different minimizer in lines 4 and 5 in subsequent iterations. A different minimizer in line 5 allows us to continue generating more separation inequalities, since the new minimizer may also contain conflicts, and by extension help us obtain tighter lower bounds. While all inequalities in the system $E_i x \leq f_i$ generated by the oracle in line 11 is required to be valid for $\mathbf{P}$, we do not however require them all to separate $\bar{x}_i$ from $\mathbf{P}$.

In our implementation, we computed the optimal $\lambda$ in line 4 using a first order gradient method with the step sizes set according to the scheme in Baker and Sheasby [1999]. As is common in Lagrangian implementations, we also simultaneously try to construct a valid upper bound (see $UB$ in Algorithm 2), see lines 6, 12. Our heuristic repair procedure tries to modify the set of conflict-containing paths, $\bar{x}_i$ in line 5, to a conflict-free set of paths. The heuristic procedure is described in Section 3.2.1. If the $\bar{x}_i$ is already conflict-free, then, we used the solution to update the upper bound as shown in line 8. However, since our goal is to generate lower bounds, in our implementation we tried to find a different minimizer in line 5 that contained conflicts so that we can use that solution to generate cuts as shown in line 11 and tighten the relaxation. Finally, the termination condition in Algorithm 2 was set to a combination of time and a bound on the number of iterations.

The scheme described in Algorithm 2 is referred to as Delayed LRC in literature, since at every iteration the Lagrangian dual problem is solved to optimality. In an alternate implementation of LRC called Non-Delay LRC [Lucena, 2005], $\bar{\lambda}$ in line 4 is obtained by taking a step along the direction of the sub-gradient. In Mogali *et al.* [2020], we previously implemented Non-Delay LRC. We switched to Delayed LRC since the empirical performance was better.

**Algorithm 2** Lagrangian Relax-and-Cut algorithm

1: **Output:** Inequalities $Ex \leq f$, optimal Lagrangian multipliers $\bar{\lambda}$, and upper bound(UB).
2: **Initialize:** $i \leftarrow 0$, $\lambda \leftarrow \emptyset$, UB $\leftarrow \infty$, $Ex \leq f \leftarrow \emptyset$
3: **repeat**
4: $\bar{\lambda} \leftarrow \underset{\lambda \geq 0}{\text{argmax}} \underset{x \in \{0,1\}^{|A|}}{\min} \{L(x, \lambda) | x \text{ satisfies Eqns } (3.4) - (3.5)\}$ , where,
$L(x, \lambda) = c^{\mathsf{T}} x + \sum_{0 \leq k < i} \lambda_k^{\mathsf{T}} (E_k x - f_k)$.
5: $\bar{x}_i \leftarrow \underset{x \in \{0,1\}^{|A|}}{\text{argmin}} \{L(x, \bar{\lambda}) | x \text{ satisfies Eqns } (3.4) - (3.5)\}$
6: **if** $\bar{x}_i$ is conflict free **then**
7: **if** $c^\top \bar{x}_i <$ UB **then**
8: UB $\leftarrow c^\top \bar{x}_i$
9: **return**
10: **else**
11: Generate cuts $E_i x \leq f_i$ separating $\bar{x}_i$ from $\mathbf{P}$ using the cut generation oracle
12: Repair $\bar{x}_i$ to generate non-conflicting paths. If the repair is successful and cost of the repaired solution is less than UB, then update UB.
13: Augment the inequalities $E_i x \leq f_i$ to the system $Ex \leq f$.
14: $i \leftarrow i + 1$
15: **until** Termination criterion is met

### 3.2.1 Primal repair procedure

Our procedure identifies a maximal independent set (MIS) of non-conflicting robots from the paths provided in $\bar{x}_i$, and fixes the path of the robots in the MIS to as they are in $\bar{x}_i$ (see line 5 in Algorithm 2). The paths for the remaining robots are computed sequentially in a fixed order. For each robot that is not part of the MIS, we compute the number of available start-end paths that do not conflict with the paths fixed for the robots in the MIS, and order the robots in increasing order of available paths. Proceeding in that fixed order, we compute the shortest start-end path that does not collide with any path fixed previously to some other robot. If such a path exists, then we fix that path for that robot, and move to the next robot in the ordering. Otherwise, the primal repair procedure returns unsuccessfully.

## 3.3 Projections and cut generation

In this section, we will develop the tools necessary for the cut generation oracle. Clearly, since any conflict is either a vertex or an edge conflict, a simple oracle that satisfies our requirement is one that simply returns the appropriate inequality violated from Eqns (3.6) and (3.7). For designing our cut-generation oracle, we would like to go beyond the simple oracle by taking into account additional considerations. Notice from Eqn (3.10) that the lower bound outputted by the LRC procedure depends on both the tightness of the relaxation $P_i$ and the MAPF objective (i.e., $c^\top x$). For generating tight relaxations, we prefer that our oracle generates separation inequalities that are deeper than Eqns (3.6), (3.7) when possible. While there can

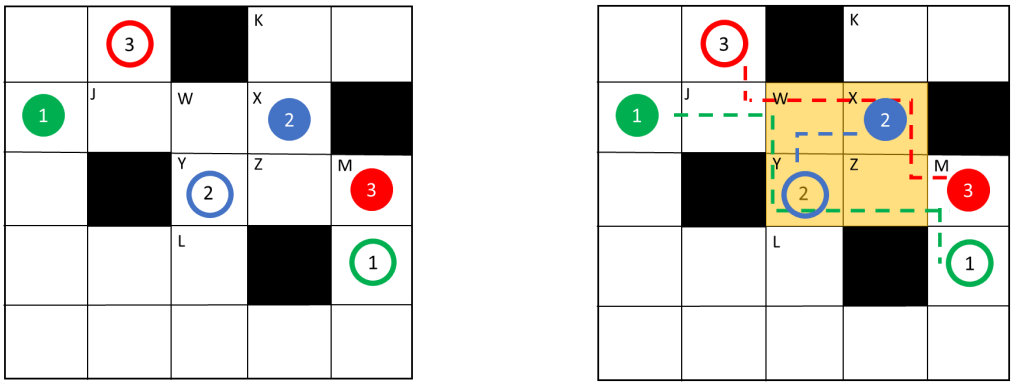



**Figure 3.1:** An MAPF instance with 3 robots. In 3.1a, the solid colored circles represent the robot's initial location at time 0, and the colored rings represent the goal location. In 3.1b, each dashed line represents one of the shortest paths between start and goal. For these set of paths, robots 1 and 2 conflict at location Y, at time 3. The yellow region is the template we have chosen.

be several candidates for such separation inequalities, we prefer that the oracle outputs those that leads to larger improvements in the lower bound. For that purpose, we account for the MAPF objective in the oracle's cut generation process.

### 3.3.1 A motivating example

We will motivate the key ideas of our cut generation oracle using the 3 robot MAPF instance shown in Figure 3.1. The layout shown in Figure 3.1 is a 4-connected grid, so between consecutive time steps a robot is allowed to either stay in its current cell or move to one of its adjacent cells by moving vertically or horizontally. In Figure 3.1b, a shortest path for each robot to their respective goal location is indicated. The shortest path costs are 6, 2 and 5 for robots 1, 2 and 3 respectively, and so a lower bound on the optimal MAPF objective is 13. For the displayed set of paths, robots 1 and 2 conflict at location Y at time 3.

Given that a set of the shortest paths for the robots are conflicting, a natural question to ask is whether there exists a feasible solution whose objective is 13? Suppose there is a conflict-free set of paths with objective 13, then the following conditions must hold:

I) The cost of robot 1's path is 6. Robot 1 must be at location W at time 2, reach X or Y at time 3, and reach Z at time 4.

II) Robot 2's path cost is 2, and it must reach Y at time 2 and remain stationary.

III) Robot 3's path cost is 5. Robot 3 must reach Z at time 1, reach X or Y at time 2 and W at time 3.

If there is a conflict-free solution satisfying conditions I), II) and III), we can find such a solution by analyzing the space of conflict-free paths that are at least partially overlapping with the yellow region (see Figure 3.1b) in the time interval $[2, 4]$. We leave it as an exercise

to the reader to verify that no conflict-free paths for the robots satisfying conditions I), II) and III) simultaneously exist. We can translate this inference into a valid inequality for $\mathbf{P}$ using the conditions I), II) and III). We explain this translation process in later sections. Through this analysis, we have inferred that a lower bound for the optimal MAPF objective for our example is at least 14.

Observe that I), II) and III) are conditions on the robot paths that are restricted to the yellow region (see Figure 3.1b) for the time interval $[2, 4]$. So in trying to determine whether a conflict-free solution not exceeding a certain cost exists (for our example it was 13), we translated it to a different problem which asks for the existence of a feasible solution satisfying certain conditions specified on a spatio-temporal neighborhood. Taking this observation as a motivation, we propose the following method. Given a conflict, the main idea of this work is to consider a set of robots (must include those in the conflict) and a spatio-temporal neighborhood around the conflict. To make inferences that potentially help improve the lower bound, we formulate a set of queries. These queries, much like our example, typically ask for the feasibility of a conflict-free path which satisfies certain conditions, where the conditions are specified using the spatio-temporal neighborhood. We answer those queries by analyzing the start-end mutually conflict-free paths for the robots projected over the spatio-temporal neighborhood, and based on the answer we derive a valid inequality for $\mathbf{P}$ and use it in Algorithm 2 for tightening the relaxation. We take this approach, since as it will be shown later in Section 3.3.2, we are able to provide a method to analyze paths over a projection.

We give an example for the proposed scheme using the MAPF instance shown in Figure 3.1. We begin by observing the vertex conflict at location Y between robots 1 and 2 at time 3. We conjecture that the spatial region in and around Y may be a bottleneck region that the robots need to carefully navigate to avoid collisions. For convenience, we will choose the yellow region marked in Figure 3.1b to be that bottleneck spatial region of our interest. Observe that the path of robot 3 also overlaps with the yellow region during that time interval, so we conjecture that the yellow region in the interval $[2, 4]$ is a spatio-temporal region that sees a lot of robot movement. This makes it a candidate neighborhood from where we can possibly understand how robots 1,2 and 3 constrain each other, and possibly lets us infer some non-trivial information about the lower bound. Using the knowledge in I), II) and III) we know that if there is an optimal solution whose MAPF objective is 13, then it is necessary that at time 3, robot 1 reaches X or Y, robot 3 reaches W, and robot 2 is stationary at Y from time 2. We turn this information into a question: at time 3 can robot 1 reach X or Y, robot 3 reach W, and robot 2 remain stationary at Y from time 2, all simultaneously? It turns out that the answer to our query is no, and as a certificate of infeasibility we obtain the inequality $x(W_1^2, X_1^3) + x(W_1^2, Y_1^3) + x(Y_2^2, Y_2^3) + x(Y_3^2, W_3^3) + x(X_3^2, W_3^3) \leq 2$.

We make a few observations regarding the proposed idea:

1. Given a conflict, to perform our analysis, we may have multiple options for selecting the robots and the spatio-temporal neighborhood. Some guidelines are needed in making these choices. We study this problem in Section 3.3.3.

2. We need to develop a query procedure, such that, answers to those queries can help improve the lower bound. We study this problem in Section 3.3.6.

3. For answering queries, we proposed to analyze the set of conflict-free start-end paths projected onto the spatio-temporal neighborhood only. Analyzing paths projected onto the neighborhood is akin to knowing the projection of $\mathbf{P}$ onto the dimensions of the problem associated with that neighborhood (i.e., appropriate dimensions of $x$ in Eqn (3.8)). Hence, we require an efficient method to analyze the conflict-free paths in the lower dimensional space. We study this problem in Section 3.3.2.

In the rest of this section, we will develop the necessary tools for generating and analyzing the conflict-free paths in the lower dimensional space. The fundamental object needed to develop these tools is the projection of the MAPF polytope $\mathbf{P}$ to lower dimensional spaces.

### 3.3.2 Projection polytopes

Recall from Section 3.1.1, $A$ denotes the set of all arcs in the time expanded graphs, i.e., $A = \bigcup_{i=1}^{\mathbf{N}} A_i$, and $\mathbf{P} \subset \{0,1\}^{|A|}$. Also, recall that we parameterized $\mathbf{P}$ in terms of $x$ variables in the 0-1 IP formulation. For some subset $S' \subseteq A$, let $x(S')$ denote the variables of $x$ corresponding to the arcs in $S'$. The projection of $\mathbf{P}$ onto dimensions spanned by $x(S')$ is defined as shown in Eqn (3.11). $\text{Proj}_{x(S')}(\mathbf{P})$ is simply the orthogonal projection of $\mathbf{P}$ onto the space spanned by the variables in $x(S')$.

$$\text{Proj}_{x(S')}(\mathbf{P}) = \{y \in \mathbb{R}^{|S'|} : \exists w \in \mathbb{R}^{|A|-|S'|}, \text{s.t. } (y,w) \in \mathbf{P}\} \tag{3.11}$$

We first show how projections are used for answering the *queries* mentioned in the previous section. Recall that each query asks about feasibility of conflict-free paths for a set of robots, satisfying certain conditions specified in terms of some spatio-temporal neighborhood. The robots and the spatio-temporal neighborhood in the query essentially specify a subset of arcs $S \subseteq A$. In the rest of the chapter, we refer to such a subset of $A$, i.e., $S$, as a rst-neighborhood, short for robot-spatio-temporal neighborhood. Assuming that $S$ is the rst-neighborhood in the query, it will be shown later that the answer to the query can be obtained by solving an optimization problem of the form shown in Eqn (3.12), where $K_0$ and $d$ are constant (vector). We explain how to specify $K_0$ and $d$ in Section 3.3.6.

$$\text{Answer} = \begin{cases} \text{Feasible, if } Val(S) \geq K_0 \\ \text{Infeasible, otherwise} \end{cases}, \text{ where } Val(S) = \max_x \{d^\mathsf{T} x(S) \,|\, x \in \mathbf{P}\} \tag{3.12}$$

Notice in the objective of Eqn (3.12), the co-efficient corresponding to any variable in $x(A \backslash S)$ is 0. Hence, we can alternatively compute $Val(S)$ using $\text{Proj}_{x(S)}(\mathbf{P})$ as shown in Eqn (3.13).

$$Val(S) = \max_{x(S)} \{d^\mathsf{T} x(S) \,|\, x(S) \in \text{Proj}_{x(S)}(\mathbf{P})\} \tag{3.13}$$

Computing $Val(S)$ using Eqn (3.12) is hard, however computing $Val(S)$ using Eqn (3.13) can be more tractable, especially when we have constructed $\text{Proj}_{x(S)}(\mathbf{P})$.

Computing $\text{Proj}_{x(S)}(\mathbf{P})$ exactly is hard due to 2 reasons: (i) $\mathbf{P}$ is not known, (ii) even if $\mathbf{P}$ was known, computing projections is computationally expensive. If $S$ is chosen judiciously,

however, we can construct a tight relaxation for the projection. More formally, we can construct a polytope $P(S) \subset \mathbb{R}^{|S|}$ such that $\text{Proj}_{x(S)}(\mathbf{P}) \subseteq P(S)$. If $P(S)$ is tight, we can compute a tight upper bound for $Val(S)$ by replacing $\text{Proj}_{x(S)}(\mathbf{P})$ by $P(S)$ in Eqn (3.13) as shown in Eqn (3.14).

$$\max_{x(S)} \{d^{\mathsf{T}} x(S) \mid x(S) \in P(S)\} \tag{3.14}$$

It turns out that a tight upper bound for $Val(S)$ is good enough, since as it will be explained later in Section 3.3.6, we only care about the case when the answer to our query is *Infeasible* (see Eqn (3.12)). If the upper bound to $Val(S)$ is $< K_0$, we are assured that $Val(S) < K_0$.

Different choices for $S$ give rise to different $P(S)$, but computing tight relaxations for $\text{Proj}_{x(S)}(\mathbf{P})$ can be challenging. In Section 3.3.3 we describe the general guideline we followed in selecting $S$ for a given conflict, and in Section 3.4 we describe the process of compactly representing the relaxation $P(S)$ using decision diagrams.

Previously in Mogali *et al.* [2020], we used $P(S)$ in the following way. In order to generate the cuts that separates $\bar{x}_i$ from $\mathbf{P}$ in line 11 of Algorithm 2, we presented a procedure inspired from [Davarnia and van Hoeve, 2020; Tjandraatmadja and van Hoeve, 2019] that outputs a face of $P(S)$, which separates $\text{Proj}_{x(S)}(\bar{x}_i)$ and $P(S)$. The face generated, i.e., $\bar{w}^\top x \leq h(\bar{w})$, is obtained by solving Eqn (3.15).

$$\bar{w} = \underset{\|w\|_2 \leq 1}{\text{argmax}}\ H(w), \text{ where } H(w) = w^{\mathsf{T}}(\text{Proj}_{x(S)}(\bar{x}_i)) - h(w),\ h(w) = \max_{y \in P(S)} \{w^{\mathsf{T}} y\} \tag{3.15}$$

In this work, we propose a different cut generating scheme. We also take into account the MAPF objective while generating the cut.

### 3.3.3 On selecting $S$ for a conflict

The choice of $S$ for a given set of start-end robot paths containing a conflict will not be made arbitrarily. An arbitrary choice for $S$ can lead to poor cuts, and computing a tight relaxation to $\text{Proj}_{x(S)}(\mathbf{P})$ is challenging. We parameterize $S$ by the sets $\mathcal{R}(S), T(S), L(S)$. We first introduce the sets $\mathcal{R}(S), T(S), L(S)$ using the notation in Section 3.1, and then describe how to design these sets based on the conflict. $\mathcal{R}(S) \subseteq [\mathbf{N}]$ is a set of indices of robots, $T(S) = \{\mathbf{l}, \mathbf{l}+\mathbf{1}, \ldots, \mathbf{u}\}$ is a discrete interval where $\mathbf{l}, \mathbf{u} \in \mathbb{Z}_+$ and $\mathbf{l} \leq \mathbf{u} < \mathbf{T}$. For each $i \in \mathcal{R}(S)$ and each time $t \in T(S)$, $L_i^t(S)$ is a non-empty subset of $N_i(t)$. Denote $L(S) = \bigcup_{i \in \mathcal{R}(S), t \in T(S)} L_i^t(S)$, then $S$ is defined as the set of all incoming and outgoing arcs associated with nodes in $L(S)$ as shown in Eqn (3.16).

$$S = \bigcup_{i \in \mathcal{R}(S), t \in T(S)} \left( \bigcup_{v_i^t \in L_i^t(S)} \left( \delta_{F_i}^+(v_i^t) \cup \delta_{F_i}^-(v_i^t) \right) \right) \tag{3.16}$$

**Example 1.** *We will illustrate the parameters of $S$ discussed in Section 3.3.1 for the rst-neighborhood (marked in yellow) chosen in Figure 3.1b. Clearly $\mathcal{R}(S) = \{1, 2, 3\}$ and $T(S) = \{2, 3, 4\}$. Note that $W$ is the only location that overlaps with the yellow region and reachable at time 2 by robot 1, and so $L_1^2(S) = \{W_1^2\}$. Similarly, $L_1^3(S) = \{W_1^3, X_1^3, Y_1^3\}$, and $L_1^4(S) = \{W_1^4, X_1^4, Y_1^4, Z_1^4\}$. For robot 2, $L_2^2(S) = \{W_2^2, X_2^2, Y_2^2, Z_2^2\}$, $L_2^3(S) = \{W_2^3, X_2^3, Y_2^3, Z_2^3\}$,*

*and* $L_2^4(S) = \{W_2^4, X_2^4, Y_2^4, Z_2^4\}$. *Lastly, for robot 3, we have* $L_3^2(S) = \{X_3^2, Y_3^2, Z_3^2\}$, $L_3^3(S) = \{W_3^3, X_3^3, Y_3^3, Z_3^3\}$ *and* $L_3^4(S) = \{W_3^4, X_3^4, Y_3^4, Z_3^4\}$.

W.l.o.g let us assume that our conflict involves robots $r_1, r_2$ at time $t_c$, and let us denote the set of nodes from $N_1, N_2$ involved in the conflict by $Z_{cf}$. So, for instance, $Z_{cf} = \{Y_1^3, Y_2^3\}$ for the vertex conflict shown in Figure 3.1b. Note that if the conflict is an edge conflict, then $Z_{cf}$ would contain 4 nodes. We say that $S$ is an appropriate choice for the conflict iff $1, 2 \in \mathcal{R}(S)$, $t_c \in T(S)$ and $Z_{cf} \subseteq L(S)$. Loosely, our requirement can be interpreted as: $S$ must contain arcs relevant to the conflict location.

Different choices to $S$ give rise to different $P(S)$, so different cuts can be derived by varying $S$. From the perspective of projections, the collision avoidance constraints in Eqns (3.6), (3.7) contain variables belonging to a small rst-neighborhood. For instance, if the conflict is an edge conflict, i.e., Eqn (3.7) is violated, then Eqn (3.7) can be viewed as a cut generated using the rst-neighborhood $S$, where $\mathcal{R}(S) = \{i, j\}$, $T(S) = \{t\}$ $L_i^t(S) = \{u_i^t\}, L_j^t(S) = \{w_j^t\}$. To generate deep cuts, we will typically choose larger rst-neighborhoods, such as the one shown in Figure 3.1b for the vertex conflict at $Y$.

### 3.3.4 Relaxation polytope $P(S)$

Given a rst-neighborhood $S \subset A$ parameterized by $\mathcal{R}(S), T(S), L(S)$ (see Section 3.3.3), we construct the relaxation polytope $P(S)$ by retaining as many relevant inequalities from the IP model in Sec. 3.1.1, for providing a tight relaxation for $\text{Proj}_{x(S)}(\mathbf{P})$. For defining $P(S)$, we will reuse the notation from Section 3.1.1 and from Section 3.3.3 for $T(S)$, i.e., $T(S) = \{\mathbf{l}, \mathbf{l}+\mathbf{1}, ..., \mathbf{u}\}$. We define the relaxation polytope $P(S)$, as:

$$P(S) = \textbf{conv}\,(x(S) \in \{0,1\}^n | x \text{ satisfies } (3.18) - (3.21)) \tag{3.17}$$

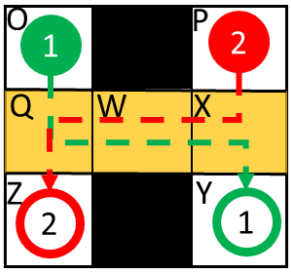
$$\sum_{a \in \delta^-_{F_i}(v_i^t)} x(a) = \sum_{a \in \delta^+_{F_i}(v_i^t)} x(a),\ \forall i \in \mathcal{R}(S),\ \forall t \in T(S), \forall v_i^t \in L_i^t(S) \tag{3.18}$$

$$\sum_{j \in \overline{V}^t(v) \cap \mathcal{R}(S)} \sum_{a \in \delta^{-1}_{F_j}(v_j^t)} \mathbb{1}(v_j^t \in L_j^t(S)) x(a) \leq 1,\ \forall t \in T(S), \forall i \in \mathcal{R}(S), \forall v_i^t \in L_i^t(S) \tag{3.19}$$

$$x(u_i^t, w_i^{t+1}) + x(w_j^t, u_j^{t+1}) \leq 1,\ \substack{\forall t \in \{\mathbf{l}-1\} \cup T(S), \forall (i,j) \in \{(k,m) \in \mathcal{R}(S) \times \mathcal{R}(S) | k \neq m\}, \\ \forall (u,w) \in \{(p,q) \in E | (p_i^t, q_i^{t+1}), (q_j^t, p_j^{t+1}) \in S\}} \tag{3.20}$$

$$\sum_{a \in S \cap A_i(t)} x(a) \leq 1,\ \forall i \in \mathcal{R}(S), \forall t \in \{\mathbf{l}-1\} \cup T(S) \tag{3.21}$$

Equation (3.18) ensures flow balance at all nodes in $L(S)$. In Eqn (3.19), $\mathbb{1}(\cdot)$ denotes the 0-1 indicator function. Equation (3.19) prohibits vertex collisions on nodes in $L(S)$. Equation (3.20) prohibits edge collisions over arcs in $S$. Equation (3.21) ensures no two arcs present in $S$, and belonging to the same arc layer in the time expanded graph of a robot, are simultaneously selected. Clearly, Eqns (3.19) and (3.21) are implied from the MAPF IP formulation, while Eqns (3.18) and (3.20) are present in the IP formulation, hence $\text{Proj}_{x(S)}(\mathbf{P}) \subseteq P(S)$.




Interpret the start and goal for each robot using the same conventions as adopted in Figure 3.1. The yellow region is the rst-neighborhood selected in Example 2.

**Figure 3.2**

### 3.3.5 Cut generation through querying

Recall from Section 3.3.2 that given a set of start-end paths for the robots (such as $\bar{x}_i$ in line 5 of Algorithm 2) containing a conflict, we proposed to generate cuts using a query procedure. The process of obtaining the answer to the query is to first select a rst-neighborhood $S \subset A$ for the conflict, and then solve an optimization problem over $P(S)$ as shown in Eqn (3.14). In this section, we will explain the query procedure and the optimization problem involving $P(S)$ through use of an example. We will formalize the procedure in the next section.

**Example 2.** *Consider the 2 robot problem on a 4-connected grid shown in Figure 3.2. The unique shortest path for each robot to reach their respective goal is indicated in Figure 3.2, and clearly at time 2 these paths conflict at $W$. The shortest start-end path cost for robots 1 and 2 is 4. For simplicity, let us assume these paths correspond to $\bar{x}_i$ in line 5 of Algorithm 2, and so we are interested in generating cuts for the conflict-containing $\bar{x}_i$. We choose the rst-neighborhood $S$ (highlighted in yellow in Figure 3.2) with parameters $\mathcal{R}(S) = \{1, 2\}$, $T(S) = [1, 3]$, $L_1^1(S) = \{Q_1^1\}$, $L_1^2(S) = \{Q_1^2, W_1^2\}$, $L_1^3(S) = \{Q_1^3, W_1^3, X_1^3\}$, $L_2^1(S) = \{X_2^1\}$, $L_2^2(S) = \{X_2^2, W_2^2\}$, $L_2^3(S) = \{Q_2^3, W_2^3, X_2^3\}$.*

*We now generate queries for cut generation. The queries will be in terms of existence of a non-conflicting set of start-end paths for the robots satisfying certain criterion. Throughout this example, $p_1$ refers to a start-end path for robot 1, and $p_2$ refers to a start-end path for robot 2. Since the unique set of the shortest start-end paths for the robots are conflicting, we know that there can be no conflict-free $p_1, p_2$ satisfying $cost(p_1) = cost(p_2) = 4$, where $cost(\cdot)$ simply returns cost of the input path evaluated using Eqn (3.2). Naturally, we will be interested to know whether there exists conflict-free start-end paths $p_1, p_2$ with $cost(p_1) + cost(p_2) = 9$. So we ask :*
***Q1:** Is there a conflict-free $p_1, p_2$ such that $cost(p_1) \leq 5$, $cost(p_2) = 4$ ?*
***Q2:** Is there a conflict-free $p_1, p_2$ such that $cost(p_1) = 4$, $cost(p_2) \leq 5$ ?*
*We will try to answer **Q1** in **A1**, our approach for answering **Q2** will be analogous.*

***A1:** Observe that if a conflict-free $p_1, p_2$ satisfying the conditions in **Q1** existed, then those paths must satisfy (i) and (ii) shown below. So if we are able to show that (i) or (ii) is infeasible, then it implies that conflict-free $p_1, p_2$ satisfying **Q1** does not exist.*

*(i) At time 2, robot 1 must be at Q or W, and robot 2 at $W$.*

*(ii) At time 3, robot 1 must be at W or X, and robot 2 at $Q$.*

*We begin by analyzing (i). For robot 1 to reach $Q$ at time 2, robot 1 must have either reached $Q$ at time 1 so that it can use the arc $(Q_1^1, Q_1^2)$ to remain at $Q$ at time 2. Or robot 1 can remain at $O$ at time 1, and then use the arc $(O_1^1, Q_1^2)$ to reach $Q$ at time 2. Similarly, it can be showed that for robot 1 to reach $W$ at time 2, it must use the arc $(Q_1^1, W_1^2)$. For robot 2 to reach $W$ at time 2, it must use the arc $(X_1^2, W_2^2)$. In other words, if there exists a $p_1, p_2$ satisfying (i), then $p_1$ must contain exactly one among the arcs $(O_1^1, Q_1^2)$, $(Q_1^1, Q_1^2)$, $(Q_1^1, W_1^2)$, and $p_2$ must contain the arc $(X_1^2, W_2^2)$. Recall from Section 3.3.2, to check the feasibility of such a conflict-free $p_1, p_2$ we proposed to use $P(S)$. We attempt to answer the feasibility problem through the optimization problem shown in Eqn (3.22). If there exists conflict-free $p_1, p_2$ satisfying (i), then the projection of $p_1, p_2$ onto the space spanned by $x(S)$ will lie in $P(S)$, and the optimum value of Eqn (3.22) will be $\geq 2$. Instead, if the optimal value of Eqn (3.22) is $< 2$, then we can safely conclude that no $p_1, p_2$ satisfying (i) exists. Note that the paths $q_1 = (O_1^0, Q_1^1, Q_1^2, Q_1^3)$, $q_2 = (P_2^0, X_2^1, W_2^2, W_2^3)$ are conflict-free paths for robots 1 and 2, and feasible to the problem in Eqn (3.22) with objective 2. The optimal value of Eqn (3.22) is indeed $\geq 2$, and so we cannot conclude that (i) is infeasible.*

$$\underset{x(S)\in P(S)}{\textit{maximize}} \quad x(O_1^1, Q_1^2) + x(Q_1^1, Q_1^2) + x(Q_1^1, W_1^2) + x(X_2^1, W_2^2) \tag{3.22}$$

*We next check if we can find paths in $P(S)$ that satisfy (ii). Using similar arguments as before, the feasibility problem can be turned into the optimization problem shown in Eqn (3.23). Observe that all the variables in the objective of Eqn (3.23) corresponds to arcs between time steps 2 and 3, and so by Eqn (3.21), we can deduce that the optimal objective value of Eqn (3.23) is $\leq 2$. Further, if there exists $p_1, p_2$ satisfying (ii), then an objective value of 2 in Eqn (3.23) is attainable by the projection of $p_1, p_2$ onto $x(S)$.*

$$\underset{x(S)\in P(S)}{\textit{maximize}} \quad x(W_1^2, W_1^3) + x(Q_1^2, W_1^3) + x(W_1^2, X_1^3) + x(W_2^2, Q_2^3) \tag{3.23}$$

*Note that there is at least one vertex of $P(S)$ at which the optimum of (3.23) is attained. The reader can verify from Eqn (3.23) that to achieve an objective of 2 at a vertex of $P(S)$, $x(W_2^2, Q_2^3)$ must be 1, and exactly one among $x(W_1^2, W_1^3)$, $x(Q_1^2, W_1^3)$, $x(W_1^2, X_1^3)$ must be a 1 at the vertex. If there is a vertex with $x(W_2^2, Q_2^3) = 1$, and exactly one among $x(W_1^2, W_1^3)$, $x(W_1^2, X_1^3)$ is also equal to 1, then trivially we have a vertex conflict on $W$ at time 2, violating Eqns (3.18), (3.19). Similarly, it can be argued that Eqn (3.20) excludes points from $P(S)$ where $x(W_2^2, Q_2^3)$ and $x(Q_1^2, W_1^3)$ are both 1. Hence, the optimal value for Eqn (3.23) cannot be 2, in fact the optimal value is 1. The optimal value of 1 implies that no conflict-free paths $p_1, p_2$ can satisfy (ii). Using the objective and the optimal value of Eqn (3.23), we derived Eqn (3.24). Although Eqn (3.24) was derived from $P(S)$, any conflict-free set of start-end robot paths for our problem must obey Eqn (3.24) since $Proj_{x(S)}(\mathbf{P}) \subseteq P(S)$.*

$$x(W_1^2, W_1^3) + x(Q_1^2, W_1^3) + x(W_1^2, X_1^3) + x(W_2^2, Q_2^3) \leq 1 \tag{3.24}$$

*Note that the shortest paths shown in Figure 3.2 clearly violates Eqn (3.24) because both arcs $(W_1^2, X_1^3)$ and $(W_2^2, Q_2^3)$ are present. Hence, Eqn (3.24) is can be viewed as a cut that separates the shortest paths from the feasible region, i.e., the polytope $\mathbf{P}$.*

*By showing that condition (ii) is infeasible using $P(S)$, we inferred that the answer to **Q1** is false, and the certificate for infeasibility is given by Eqn (3.24). Similarly, by symmetry we can deduce that the answer to **Q2** is also false, and the certificate for infeasibility is given by Eqn (3.25). Like Eqn (3.24), Eqn (3.25) is also a valid inequality for the problem that separates the shortest paths from $\mathbf{P}$. We leave it as an exercise to the reader to derive Eqn (3.25).*

$$x(W_1^2, X_1^3) + x(W_2^2, W_2^3) + x(X_2^2, W_2^3) + x(W_2^2, Q_2^3) \leq 1 \tag{3.25}$$

*Based on our answers to **Q1** and **Q2**, we can conclude that there are no conflict-free paths $p_1, p_2$ such that $cost(p_1) + cost(p_2) = 9$. While Eqns (3.24), (3.25) separate the paths shown in Figure 3.2 from $\mathbf{P}$, we can possibly derive stronger separation inequalities by querying about the existence of conflict-free paths $p_1, p_2$ such that $cost(p_1) + cost(p_2) = 10$. Through queries **Q3, Q4, Q5** shown below, we ask is there a conflict-free $p_1, p_2$ such that:*
***Q3:*** *$cost(p_1) \leq 5$, $cost(p_2) \leq 5$ ?* ***Q4:*** *$cost(p_1) \leq 6$, $cost(p_2) = 4$ ?* ***Q5:*** *$cost(p_1) = 4$, $cost(p_2) \leq 6$ ?*
*We will address **Q4** in **A4** to highlight an important detail that was not seen in **A1**. The procedure for answering **Q3** and **Q5** will be analogous, which we will skip.*

***A4:*** *If a conflict-free $p_1, p_2$ satisfying the conditions in **Q4** exists, then they must satisfy:*

*(iii) At time 2, robot 1 must be at O or Q or W, and robot 2 at $W$.*

*(iv) At time 3, robot 1 must be at Q or W or X, and robot 2 at $Q$.*

*For robot 1 to reach $Q$ or $W$ at time 2, we know from **A1** which arcs need to be used between times 1 and 2. For robot 1 to reach $O$ at time 2, it must use either $(O_1^1, O_1^2)$ or $(Q_1^1, O_1^2)$. However, note that $O_1^2 \notin L_1^2(S)$, and so the arc $(O_1^1, O_1^2) \notin S$. Like earlier, we can pose the feasibility problem for $p_1, p_2$ satisfying (iii) as the optimization problem shown in Eqn (3.26), albeit $x(O_1^1, O_1^2)$ is absent from the objective. Denoting the optimal objective value of Eqn (3.26) by opt_cost, Eqn (3.21) implies that opt_cost $\leq 2$.*

$$opt\_cost = \underset{x(S)\in P(S)}{\textit{maximize}} \quad x(Q_1^1, O_1^2)+x(O_1^1, Q_1^2)+x(Q_1^1, Q_1^2)+x(Q_1^1, W_1^2)+x(X_2^1, W_2^2) \tag{3.26}$$

*If opt_cost is $< 2$, unlike arguments used in **A1**, we cannot however claim that **Q4** is infeasible. The claim cannot be made because there may be a conflict-free $p_1, p_2$ satisfying (iii), with $p_1$ passing through the arc $(O_1^1, O_1^2)$, but the answer obtained by solving Eqn (3.26) does not account for such a possibility due to the limitations of the chosen $S$. Despite this limitation, the inequality $x(Q_1^1, O_1^2) + x(O_1^1, Q_1^2) + x(Q_1^1, Q_1^2) + x(Q_1^1, W_1^2) + x(X_2^1, W_2^2) \leq$ opt_cost, may still be useful. If opt_cost $< 2$, then the inequality is not implied by the flow balance inequalities of the MAPF problem (i.e., Eqns (3.4) and (3.5)), and so including it in Algorithm 2 for tightening the relaxation of $\mathbf{P}$ is beneficial. Unfortunately, in this example, it turns out that opt_cost is 2, and hence not useful for tightening the relaxation. The paths $q_1, q_2$ mentioned in **A1** attains the optimum value of 2 in Eqn (3.26). Next, moving onto (iv), using arguments similar to those in **A1** for the time 3 case, it can be shown that (iv) is infeasible. By proving infeasibility of (iv), we also obtain the following separation inequality:*

$$x(O_1^2, Q_1^3)+x(Z_1^2, Q_1^3)+x(Q_1^2, Q_1^3)+x(W_1^2, Q_1^3)+x(W_1^2, W_1^3)+x(Q_1^2, W_1^3)+x(W_1^2, X_1^3)+x(W_2^2, Q_2^3) \leq 1 \tag{3.27}$$

*Equation (3.27) is a certificate for infeasibility of **Q4**, and note that the separation inequality Eqn (3.24) derived earlier is implied by Eqn (3.27). We skip the proofs for infeasibility of **Q3, Q5** using $P(S)$ here.*

It is worth noting that the complexity (size) of the optimization problems generated for each query, i.e., Eqns (3.22), (3.23) and (3.26), are each only dependent on the size of the rst-neighborhood $S$, and so the complexity does not scale with the makespan constraint $\mathbf{T}$.

### 3.3.6 Objective Cuts

We next formalize the cut generation procedure from the previous section in Algorithm 3, and frequently refer to Example 2 while explaining the steps of the algorithm. Algorithm 3 takes as input the set of start-end paths $\bar{x}_i$ from line 5 in Algorithm 2, and a positive integer Max Offset, and outputs inequalities for use in line 11 of Algorithm 2.

Algorithm 3 considers conflicts in $\bar{x}_i$ one after the other, and generates cuts for each conflict separately. For each conflict, we first select an appropriate rst-neighborhood $S$ (see line 4) satisfying the conditions mentioned in Section 3.3.3. For the robots associated to $\mathcal{R}(S)$, through lines 5 - 12, we emulate the process of querying and generating cuts that we saw earlier in Example 2 (compare to ***Q1*** - ***Q5***).

To explain lines 5 - 8, we first recall a few facts from Example 2. Observe that we progressively queried for existence of solutions with increasing sum of start-end path costs for the robots. Through queries ***Q1*** and ***Q2***, we asked for the existence of solutions where the sum of path costs was 9. Since the start-end path cost for any robot cannot be lower than 4 in Example 2, there are only 2 legal ways to partition 9 into individual path costs for the robots, and these are the costs considered in ***Q1*** and ***Q2***. We then extended this process where the sum of start-end path costs was 10. There are only 3 legal ways to partition 10 into individual robot start-end path costs, and these were the costs considered in ***Q3*** - ***Q5***. To replicate this process from Example 2 in Algorithm 3, we introduce a variable *offset*. Within the loop spanning lines 5 - 12, we are interested in querying about conflict-free paths for robots associated to $\mathcal{R}(S)$, where the sum of the individual robot path costs does not exceed $SP(\mathcal{R}(S)) +$ *offset*. $SP(\mathcal{R}(S))$ is the sum of the shortest start-end path cost for the robots associated to $\mathcal{R}(S)$. Previously in Example 2, we had $\mathcal{R}(S) = \{1, 2\}$, $SP(\mathcal{R}(S)) = 4 + 4$, and Max Offset was 2. In line 6, we generate all the legal ways to partition the cost $SP(\mathcal{R}(S)) +$ *offset* among the robots associated to $\mathcal{R}(S)$. In line 7, we iterate over all the partitions, and for each partition we generate a *query* as shown in line 8. These queries are comparable to **Q1** - **Q5** in Example 2. For instance, **Q1** is the query concerning existence of conflict-free solutions corresponding to the partition $9 = 5 + 4$.

Cuts are generated in the innermost loop (lines 9 - 11) of Algorithm 3. Recall from Example 2, after formulating a *query*, we tried to disprove the existence of a feasible solution satisfying the costs in the *query* by further breaking it down into a set of feasibility problems (compare to (i), (ii) in ***A1***, or (iii), (iv) in ***A4***). Recall, each of those feasibility problems asks if there is a way for the robots to reach any one location from a set of locations at a particular time without conflicting, and depending on the answer, a cut may be generated. The earliest time corresponding to which there is a feasibility problem is the conflict time, and the last time

**Algorithm 3** Pseudo-code for objective-based cuts

1: **Given:** Start-end paths for robots $\bar{x}_i$, containing at least one conflict, Max Offset
2: **Initialize:** *offset* = 0, Cut_Container = $\emptyset$
3: **for each** *conflict* in $\bar{x}_i$ **do**
4: Select rst-neighborhood $S$ for *conflict*.
5: **for** *offset* $<$ Max Offset **do**
6: Partitions $\leftarrow$ Compute_legal_partitions($\mathcal{R}(S)$, *offset*).
7: **for each** *partition* $\in$ Partitions **do**
8: Generate *query* based on *partition*
9: **for** time $t \in T(S)$ and $t \geq$ *conflict time* **do**
10: Translate *query* into optimization problem at time $t$:

$$opt\_cost = \max_{x(S)\in P(S)} \sum_{j\in\mathcal{R}(S)} \sum_{a\in S\cap A_j(t-1)} d(a)x(a) \tag{3.28}$$

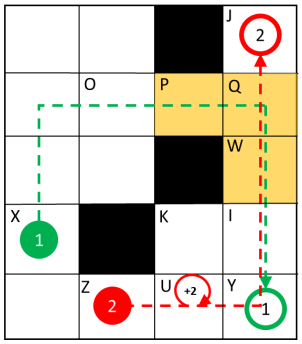
where $d(a)$ is set according to Eqn (3.29).
11: If the inequality $\sum_{j\in\mathcal{R}(S)} \sum_{a\in S\cap A_j(t-1)} d(a)x(a) \leq opt\_cost$ is not implied by Eqn (3.21), then store the inequality in Cut_Container.
12: *offset* $\leftarrow$ *offset* + 1.
13: Discard redundant inequalities from Cut_Container.
14: **return** Cut_Container

corresponding to which there is a feasibility problem is the largest time in $T(S)$. In Algorithm 3, the loop in line 9 iterates over the times for which we generate feasibility problems. The feasibility problem for a particular time and the cut generation step is set up in lines 10 - 11.

We will explain the process of setting up the feasibility problem from the *query* in line 8 at time $t$, and the process of extracting a cut in this paragraph. Assume that the *query* asks whether there is conflict free solution for robots associated to $\mathcal{R}(S)$, where the path cost for robot $r_j$ (where $j \in \mathcal{R}(S)$) does not exceed $len_j$. We first translate the question into an optimization problem of the form shown in Eqn (3.28) (compare to Eqns (3.22), (3.23), (3.26) in Example 2). The co-efficient $d(a)$ in Eqn (3.28) is defined as shown below:

$$d(a) = \begin{cases} 1, & \text{if } Sh(a) \leq len_j \\ 0, & \text{otherwise.} \end{cases} \tag{3.29}$$

where, $Sh(a)$ in Eqn (3.29) returns the cost of the shortest $s_j^0 - g_j^{\mathbf{T}}$ path in $F_j(N_j, A_j)$ passing through arc $a$. Based on the optimal objective value of Eqn (3.28), we derive the inequality shown in line 11 (compare to Eqns (3.24), (3.25) and (3.27) in Example 2). Recall from Example 2, the inequality may or may not be a certificate for proving that a solution satisfying the robot path costs in the *query* is infeasible, mainly due to the limitations of the chosen $S$. However, the inequality derived may still be useful for tightening the relaxation in Algorithm 2. Recall from Section 3.2, Algorithm 2 progressively tightens the flow balance constraints polytope $P_0$, and so the inequality derived in line 11 is useful for tightening if it is not valid for $P_0$. Inequalities valid for $P_0$ are inconsequential to Algorithm 2 for generating strong lower




The loop on the path of robot 2 at location U is to indicate that robot 2 waits for 2 time units at U, before moving to Y at time 4. A vertex conflict occurs at location W at time 6. The yellow region is the chosen rst-neighborhood.

**Figure 3.3**

bounds, and so in line 11 we used the following procedures for identifying and discarding such inequalities. The check is similar to our procedure from Example 2, where we compute an upper bound for the LHS of the inequality in line 11 using Eqn (3.21). If the upper bound computed equals $opt\_cost$, then we discard the inequality. Otherwise, we store the inequality in the Cut_Container as shown in line 11.

Not all the inequalities derived during Algorithm 3 are expected to form an irredundant system. We previously saw such a case in Example 2, where Eqn (3.27) implies Eqn (3.24). Depending on the parameter $L(S)$ of neighborhood $S$, inequalities generated for different values of time $t$ (see line 9) but corresponding to the same *query* may also be redundant. Although not essential, we recommend discarding redundant inequalities from Cut_Container before using them in Algorithm 2.

We make a few observations regarding Algorithm 3. Note that the inequalities derived in Algorithm 3 are based on querying for solutions satisfying certain costs (objective). Owing to this flavor, we call the inequalities generated by Algorithm 3 as objective cuts. Note that there is no guarantee that the objective cuts for a conflict separate the input conflict-containing start-end robot paths $\bar{x}_i$ from $\mathbf{P}$. The most expensive step of Algorithm 3 is computing $opt\_cost$ in line 10. It remains to be shown how the optimization problem shown in Eqn (3.28) is solved. In Section 3.4, we will provide a compact construction for $P(S)$ using a DD, and using the DD we will provide a simple optimization procedure.

### 3.3.7 Delay-and-Long Inequalities

In this section, we will present another method to generate valid inequalities for $\mathbf{P}$, for the purposes of tightening the relaxation in Algorithm 2. Similar to the approach in Section 3.3.6, we will generate each inequality by making use of a conflict. Previously in objective cuts, we generated inequalities by making use of the conflict time, and the robots involved in the conflict. In this section, we will use some additional information present in the conflict-containing start-end robot paths while generating cuts. The inequalities that will be presented in this section will be called the delay-and-long inequalities for reasons that will become clear shortly. We will motivate the need for these inequalities and explain the inequality generation procedure with the help of Example 3.

**Example 3.** *Consider the scenario shown in Figure 3.3. The shortest start-end path cost for robot 1 is 4, and for robot 2 it is 6. In the scenario shown, instead of taking the shortest path from X to Y, robot 1 takes a longer path of cost 8. On the other hand, robot 2 waits for 2 seconds at location $U$ en route to its goal, thus resulting in a vertex conflict at $W$ involving robots 1 and 2 at time 6. Let us assume that the paths shown in Figure 3.3 correspond to $\bar{x}_i$ in line 5 of Algorithm 2, and so we are interested in generating cuts for the conflict-containing $\bar{x}_i$. We choose the rst-neighborhood $S$ (highlighted in yellow in Figure 3.3) with parameters $\mathcal{R}(S) = \{1, 2\}$, $T(S) = [4, 7]$, $L_1^4(S) = \{P_1^4\}$, $L_1^5(S) = \{P_1^5, Q_1^5\}$, $L_1^6(S) = \{P_1^6, Q_1^6, W_1^6\}$, $L_1^7(S) = \{P_1^7, Q_1^7, W_1^7\}$, $L_2^4(S) = \{W_2^4\}$, $L_2^5(S) = \{Q_2^5, W_2^5\}$, $L_2^6(S) = \{P_2^6, Q_2^6, W_2^6\}$, and $L_2^7(S) = \{P_2^7, Q_2^7, W_2^7\}$.*

*For the chosen $S$, suppose we choose to generate objective cuts using Algorithm 3 with Max Offset set to 2, then note we could not have derived any cut that separates the paths shown in Figure 3.3 from the polytope $\mathbf{P}$. To see why, observe that there are no start-end paths for robot 1 which uses at least one arc in $S$, and the path cost not exceeding 6 (i.e., $SP(\{1\})$ + Max Offset). Hence, when we construct each query and formulate the optimization problem shown in Eqn (3.28), the objective will not contain any terms corresponding to robot 1. Although one work-around is to expand $S$, we will show in Section 3.4 that increasing the size of $S$ is computationally unattractive. Taking into cognizance the limitations on the size of $S$, we need an alternate method to generate cuts for these types of scenarios.*

*To generate a cut for the scenario shown in Figure 3.3, we will first formulate a 3 part hypothesis concerning the conflict, and then generate queries assuming all the 3 parts of the hypothesis to be true. We will then follow the same strategy of using these queries to recover inequalities as we did earlier with objective cuts. Our first part of the hypothesis is that, in order to avoid conflicts with robots omitted in Figure 3.3, robot 1 was initially forced to take a longer route to reach its goal, but along this longer route, robot 1 encounters a vertex conflict at time 6 with robot 2. The second part of our hypothesis is that, robot 2 was initially forced to wait at some location along its shortest path to its goal, to avoid collisions with other robots. This wait delays robot 2's arrival at $W$, leading to a vertex conflict with robot 1. Our third part of the hypothesis is that, there exists an optimal solution where robots 1 and 2 have to navigate through the rst-neighborhood $S$ (i.e., there exists an optimal solution where some arcs of $S$ are used). Based on this 3-part hypothesis, through our queries, we wish to understand how robots 1 and 2 constrain each other in the rst-neighborhood $S$, for strengthening the lower bound.*

*We next motivate our query methodology for understanding how robots 1 and 2 constrain each other. If we assumed that robot 2 was absent from the problem, and the first part of the hypothesis is true, we may expect that robot 1 passes through $W$, and that robot 1 follows the shortest path from $W$ to its goal location $Y$. Similarly, if the $2^{nd}$ part of our hypothesis is true, and robot 1 was removed from the problem, then beginning from time 6, we may expect robot 2 to trace a path that is the shortest from $W$ to $J$. Intuitively, our queries will be indirectly asking whether those expectations on the robot's paths still hold when robots 1 and 2 are both present in the problem. If we can prove that our expectation is false, we obtain a cut as a certificate of infeasibility. We generate more queries by relaxing the requirement that robots follow the shortest path to their goal starting from the conflict time, and check whether more cuts can be generated.*

We next specify how to generate delay-and-long inequalities for a given conflict. Let $r_1$ and $r_2$ denote the robots in the conflict, and let $t_{con}$ denote the conflict time. Let $loc_1 \in V$ (resp. $loc_2 \in V$) denote the location of robot $r_1$ (resp. $r_2$) at time $t_{con}$. Let $sh_1$ (resp. $sh_2$) denote the length of the shortest start-end path for $r_1$ (resp. $r_2$) passing through $loc_1$ (resp. $loc_2$). Let $del_1$ (resp. $del_2$) denote the difference in time between $t_{con}$ and the earliest time $r_1$ (resp. $r_2$) can reach location $loc_1$ (resp. $loc_2$) from its respective start location at time 0. The delay-and-long inequality generation process will be identical to Algorithm 3, except for the way line 6 is implemented to generate partitions, and how we interpret the query in line 8, which ultimately affects how the objective function in Eqn (3.28) is defined. Since we assumed that the conflicting robots are $r_1, r_2$, note that $\{1, 2\} \subseteq \mathcal{R}(S)$, where $S$ is the rst-neighborhood for the conflict chosen in line 4. Previously, for objective cuts, we computed legal partitions of $SP(\mathcal{R}(S))$+*offset* in line 6 of Algorithm 3. For delay-and-long inequalities, we will instead compute partitions of $SP(\mathcal{R}(S))$+*offset*+$(sh_1 - SP(\{1\}))$+$(sh_2 - SP(\{2\}))$ in line 6. For conflicting robot indices i.e $j \in \{1, 2\}$ (resp. robots in $\mathcal{R}(S)$ other than those in the conflict, i.e., $j \in \mathcal{R}(S)\backslash\{1, 2\}$), a partition is said to be illegal for robot $r_j$, if the cost associated to the robot in the partition is $< sh_j$ (resp. $< SP(\{j\})$). Analogous to Eqn (3.29), we next describe how $d(a)$ in Eqn (3.28) is specified. In line 8 of Algorithm 3, assume that the path cost specified for robot $r_j$ (where $j \in \mathcal{R}(S)$) in the *partition* is $len_j$. For arc $a = (u_j^{t-1}, v_j^t) \in A_j(t-1) \cap S$, we define the co-efficient $d(a)$ in Eqn (3.28) as:

$$d(a) = \begin{cases} 1, & \text{if } j \in \{1,2\},\ \&\ a' = (u_j^{t-1-del_j}, v_j^{t-del_j}) \in A_j(t-1-del_j),\ \&\ Sh(a') \leq len_j \\ 1, & \text{if } j \in \mathcal{R}(S)\backslash\{1,2\}\ \&\ Sh(a) \leq len_j. \\ 0, & \text{otherwise.} \end{cases} \tag{3.30}$$

**Example 3 (continued).** *In the example shown in Figure 3.3, we have $t_{con} = 6$, $loc_1 = loc_2 = W$, $sh_1 = 8$, $sh_2 = 6$, $del_1 = 0$, and $del_2 = 2$. For the case offset = 0, and $t = 7$, we will explain the construction of the objective function in Eqn (3.28) in accordance with Eqn (3.30). Note that, $SP(\mathcal{R}(S)) = 4 + 6$, and so in line 6 of Algorithm 3, we need to compute partitions for 10 (i.e., $SP(\mathcal{R}(S))$) + 0 (i.e., offset) + (8 (i.e., $sh_1$) - 4 (i.e., $SP(1)$)) + (6 (i.e., $sh_2$) - 6 (i.e., $SP(2)$)). There is only 1 legal way to partition such a cost between the 2 robots, and that is 8 for robot 1 and 6 for robot 2. We will construct the objective function using Eqn (3.30), beginning with the variables for robot 2. From $L_2^6(S), L_2^7(S)$, it can be verified that the arcs in $A_2(6) \cap S$ are $(O_2^6, P_2^7), (P_2^6, O_2^7), (P_2^6, P_2^7), (P_2^6, Q_2^7), (Q_2^6, J_2^7), (Q_2^6, P_2^7), (Q_2^6, Q_2^7), (Q_2^6, W_2^7)$, $(J_2^6, Q_2^7)$ $(W_2^6, W_2^7)$, $(W_2^6, Q_2^7)$, $(W_2^6, I_2^7)$, $(I_2^6, W_2^7)$. Of the arcs in $A_2(6) \cap S$, we next have to identify those arcs for which the co-efficient is 1 in the objective function. Since robot 2 is part of the conflict, according to the top row in Eqn (3.30), first we need to check if these arcs are present 2 (i.e., $del_2$) time steps earlier. So for instance, for the coefficient of $(O_2^6, P_2^7)$ to be 1, we first require the arc's time-shifted counterpart, i.e., $(O_2^4, P_2^5)$, to be present in $A_2(4)$. Since $O$ is not reachable to robot 2 earlier than time 5, the arc $(O_2^4, P_2^5) \notin A_2(4)$. Using similar arguments, we can argue that the only time shifted counterparts of $A_2(6) \cap S$ that are present in $A_2(4)$ are $(W_2^4, W_2^5)$, $(W_2^4, Q_2^5)$, $(W_2^4, I_2^5)$, and $(I_2^4, W_2^5)$. According to top row of Eqn (3.30), we need to check whether the shortest start-end path for robot 2 through these arcs are $\leq 6$ (i.e., $len_2$). It turns out that $(W_2^4, Q_2^5)$ is the only arc that satisfies this criterion,*

*and so among the arcs in $A_2(6) \cap S$, the co-efficient of $(W_2^6, Q_2^7)$ in the objective is set to 1, and the rest to 0. Repeating this exercise with robot 1, the reader can verify that the only arc in $A_1(6) \cap S$ with co-efficient 1 in the objective is $(W_1^6, I_1^7)$. So the optimization problem is given by:*

$$\underset{x(S)\in P(S)}{\textit{maximize}} \quad x(W_1^6, I_1^7) + x(W_2^6, Q_2^7) \tag{3.31}$$

*Clearly $P(S)$ cannot contain a point where $x(W_1^6, I_1^7)$ and $x(W_2^6, Q_2^7)$ are both 1, since otherwise, we will trivially have a vertex collision on $W$ at time 6, thus contradicting Eqn (3.19). It can be shown that the optimal objective value of Eqn (3.31) is 1, and the inequality we derive is : $x(W_1^6, I_1^7) + x(W_2^6, Q_2^7) \leq 1$. Although in this example, the cut derived is weaker than the simple vertex conflict inequality (i.e. Eqn (3.6)) at $W$, in many cases we may be able to obtain tighter inequalities.*

## 3.4 Decision Diagrams for $P(S)$

In sections 3.3.6 and 3.3.7, we obtained our cuts by solving a maximization problem over $P(S)$, see Eqn (3.28). In this section, we will present a simple procedure to perform the maximization. We will represent $P(S)$ using a decision diagram (DD). A DD is simply a layered directed acyclic multigraph. We will show that our task of maximizing over $P(S)$ is equivalent to computing the longest path cost on the DD, which is procedurally simple.

We begin with an outline of DD data structure. Borrowing notation from Davarnia and van Hoeve [2020], we denote the DD for $P(S)$ by $\mathcal{D}(S) = (\mathcal{U}, \mathcal{A}, f)$, where $\mathcal{U}$ represents a set of nodes, $\mathcal{A}$ represents arcs in a top-down multi-graph, $f$ labels each arc in $\mathcal{A}$ to some subset of arcs in $S$. Given an arc $a \in \mathcal{A}$, for convenience, we will refer to the arcs from $S$ in the set $f(a)$ as the labels of $a$. $\mathcal{U}$ can be decomposed into $|T(S)| + 2$ layers $\mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_{|T(S)|+1}$, and $\mathcal{A}$ into $|T(S)| + 1$ layers $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{|T(S)|}$. $\mathcal{U}_0$ contains a single node $\mathbf{sr}$ called source, and $\mathcal{U}_{|T(S)|+1}$ contains a single node $\mathbf{sk}$ called sink. For any arc in layer $j$, its tail is connected to a node in $\mathcal{U}_j$, and its head to a node in $\mathcal{U}_{j+1}$. Consequently, any arc directed path from the source to the sink node (a.k.a. $\mathbf{sr}$ - $\mathbf{sk}$ path) in the DD encodes a subset of arcs from $S$.

At a high level, we will be representing the vertices of $P(S)$ as $\mathbf{sr} - \mathbf{sk}$ paths in $\mathcal{D}(S)$. The graphical representation of $\mathcal{D}(S)$ will enable us to compactly represent the vertices of $P(S)$. In Section 3.4.3, we will explain how the compact representation of the vertices through $\mathcal{D}(S)$ helps us to solve the maximization problem in Eqn (3.28) efficiently. We will begin this section by describing our strategy for constructing $\mathcal{D}(S)$.

### 3.4.1 Strategy for constructing $\mathcal{D}(S)$

In constructing $\mathcal{D}(S)$, since our goal is to compactly represent the vertices of $P(S)$, we will first make some observations regarding those vertices. Let us assume that $T(S) = \{\mathbf{l}, \mathbf{l}+\mathbf{1}, \ldots, \mathbf{u}\}$. Given any vertex $x_p$ of $P(S)$, we can infer the following information from $x_p$. For each $i \in \mathcal{R}(S)$ and $t \in \{\mathbf{l}-1\} \cup T(S)$, there is at most one arc $a \in S \cap A_i(t)$, such that, $x_p(a) = 1$, due to Eqn (3.21). If $t \in \{\mathbf{l}-1, \mathbf{l}, \ldots, \mathbf{u}-1\}$, we can observe that:

A) If $head(a) \in L_i^{t+1}(S)$, then because of Eqn (3.18), there must have been an arc $b \in A_i(t+1) \cap S$, such that, $x_p(b) = 1$ and $tail(b) = head(a)$.

B) If $head(a) \notin L_i^{t+1}(S)$, then exactly one of the following is true:

(a) No arc $b \in A_i(t+1) \cap S$ can be found, such that, $x_p(b) = 1$.

(b) There exists an arc $b \in A_i(t+1) \cap S$, such that, $x_p(b) = 1$ and $tail(b) \in N_i(t+1) \backslash L_i^{t+1}(S)$.

So our first observation is that, for each robot $r_i$ (where $i \in \mathcal{R}(S)$), we can extract a set of arcs from the vertex $x_p$, with at most one arc for every time step. If these arcs are arranged temporally, then they form a traversable path for $r_i$ if situation A) occurs $\forall t \in \{\mathbf{l}-1\} \cup T(S)$. Otherwise, if B) occurs for some $t \in \{\mathbf{l}-1\} \cup T(S)$, nevertheless the arranged arcs can be interpreted as a set of broken paths for $r_i$. Our second observation is that, the set of (possibly) broken paths for the robots associated to $\mathcal{R}(S)$ do not contain vertex and edge conflicts of the kind prohibited by Eqns (3.19) and (3.20). So the problem of constructing $\mathcal{D}(S)$ to compactly represent the vertices of $P(S)$, can instead be viewed as a problem of compactly representing a set of non-conflicting (in a certain sense) robot paths.

A popular method for compactly representing the set of all non-conflicting robot paths is through the use of a state diagram. The main challenge with interpreting $\mathcal{D}(S)$ as a state diagram is that, since some vertices of $P(S)$ may correspond to broken robot paths, we may be unable to pin down the state of the robot. For e.g., if B)b occurs, then the vertex $x_p$ implies that robot $r_i$ is simultaneously in 2 different states, namely $head(a)$ and $tail(b)$ at time $t+1$. Such a dual state representation is not permissible using a state diagram. Below, we will show that we can address this difficulty by introducing an auxiliary state for a robot, while continuing to interpret $\mathcal{D}(S)$ as a state diagram. We will label each arc in the DD using arcs from $S$, that enable the robots to transition from the arc's tail to its head state. This labeling allows us to read off vertices of $P(S)$ from the labels on the $\mathbf{sr} - \mathbf{sk}$ paths of $\mathcal{D}(S)$. Interpreting the DD as a state diagram is not new, see [Bergman *et al.*, 2016] for more examples.

### 3.4.2 Construction of $\mathcal{D}(S)$

To construct $\mathcal{D}(S)$, we will interpret each node in $\mathcal{U}$ as a state. Formally, a state specifies a time $t$ and maps each $i \in \mathcal{R}(S)$ to a robot-state. For $i \in \mathcal{R}(S)$, a robot $r_i$ occupying location $v \in V$ at time $t$, is said to be in the robot-state:

$$\begin{cases} v_i^t, & \text{if } t \in T(S) \text{ and } v_i^t \in L_i^t(S) \\ \mathbf{o}\ , & \text{otherwise} \end{cases} \tag{3.32}$$

Throughout this section, if $k \notin T(S)$, all references to $L_i^k(S)$ should be interpreted as the empty set. We provide an example demonstrating the utility of the robot-state $\mathbf{o}$ shortly.

**Construction of $\mathcal{U}$:** We intend to construct $\mathcal{U}$, such that, there is a 1:1 correspondence between nodes in $\mathcal{U}$ and states that are realizable by $P(S)$. Intuitively, a state at time $t$ is said to

be realizable by $P(S)$ if there exists a vertex $x_v$ of $P(S)$, such that, for every robot associated to $\mathcal{R}(S)$, the location (in $V$) occupied by the robot as implied by $x_v$ at time $t$, coincides with the location implied by the state [1].

Recall that $T(S) = \{\mathbf{l}, \mathbf{l}+\mathbf{1}, ..., \mathbf{u}\}$. At time $t \in T(S)$, observe, there are at most $\prod_{i \in \mathcal{R}(\mathcal{S})} (|L_i^t(S)| + 1)$ distinct states realizable as a consequence of Eqn (3.32). For each one of those $\prod_{i \in \mathcal{R}(\mathcal{S})} (|L_i^t(S)| + 1)$ states at time $t \in T(S)$, we introduce a node in the layer $t - \mathbf{l} + 1$ of $\mathcal{D}(S)$, i.e., $\mathcal{U}_{t-\mathbf{l}+1}$. For node $u \in \mathcal{U}$, we shall use the notation $u[i]$ to denote the robot-state of $r_i$ in $u$. The node $\mathbf{sr}$ in layer $\mathcal{U}_0$ and node $\mathbf{sk}$ in layer $\mathcal{U}_{T(S)+1}$, both correspond to the state $\mathbf{sr}[i] = \mathbf{sk}[i] = \mathbf{o}$, $\forall i \in \mathcal{R}(S)$. Some nodes (states) populated in $\mathcal{U}$, may contain vertex collisions. A node $u \in \mathcal{U}$ is said to contain a vertex collision iff $\exists i, j \in \mathcal{R}(S)$, such that, both, $u[i]$ and $u[j]$ are different from $\mathbf{o}$ (i.e., $u[i], u[j] \in L(S)$), and $u[i], u[j]$ correspond to the same location in $V$. We remove all nodes that contain such vertex collisions from $\mathcal{U}$, as those states are not realizable by any vertex of $P(S)$ owing to Eqn (3.19).

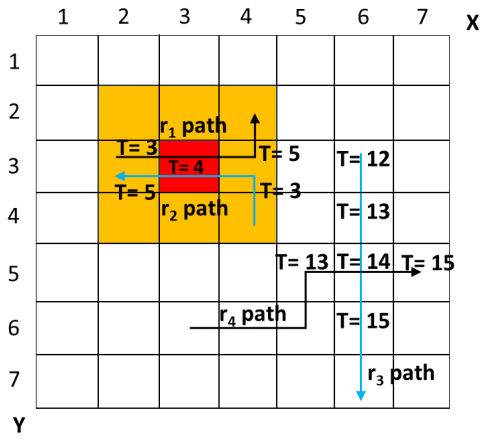
**Motivation for $\mathbf{o}$ as a robot-state:** Say robot $r_1$ traverses from location $a$ at time 1 to $b$ at time 2, i.e., $r_1$ traverses the arc $(a_1^1, b_1^2) \in A_1(1)$ (refer $A_i(t)$ notation from Section 3.1.1), and then the arc $(c_1^2, d_1^3) \in A_1(2)$. Firstly, observe that such a path for $r_1$ is infeasible for the MAPF problem. However, if $b_1^2, c_1^2 \notin L_1^2(S)$, then recall that $P(S)$ does not enforce flow balance at the nodes $b_1^2, c_1^2$. Consequently, it is possible for $P(S)$ to contain a vertex $x_v$, where, $x_v(a_1^1, b_1^2) = x_v(c_1^2, d_1^3) = 1$. Since $x_v$ implies that $r_1$ is at two different locations at time 2, it is not possible to capture this situation as a state. To reconcile this possibility with our state space approach, by introducing $\mathbf{o}$, it allows us to interpret the situation as: $r_1$ traverses robot-state $a_1^1$ to robot-state $\mathbf{o}$ using the arc $(a_1^1, b_1^2)$, and then transitions from $\mathbf{o}$ to robot-state $d_1^3$ using the arc $(c_1^2, d_1^3)$.

**Construction of $\mathcal{A}$:** Continuing with the state diagram interpretation of $\mathcal{D}(S)$, arcs in $\mathcal{A}$ represent feasible state transitions. The labels on each arc indicate which subset from $S$ enable the robots associated to $\mathcal{R}(S)$ to transition between the pair of states connected to the arc. The state (node) $u_1 \in \mathcal{U}$ is connected to a state $u_2 \in \mathcal{U}$ by an arc in $\mathcal{A}$ iff the state transition from $u_1$ to $u_2$ is legal. For the transition from $u_1$ to $u_2$ to be legal, it is required that for every $i \in \mathcal{R}(S)$, transitioning from the robot-state of $r_i$ in $u_1$ to the robot-state of $r_i$ in $u_2$ is legal, and the transition does not cause an edge collision between robots. We begin by specifying the legal robot-state transitions, and we also specify which arc in $S$ enables the transition, so that we can use this information for labeling the arcs of $\mathcal{A}$.

**Feasible robot-state transitions:** The conditions listed in A) and B) characterize the robot paths (vertices) of $P(S)$, so we use them for identifying legal robot-state transitions.

• At time $t \in T(S)$, the robot $r_i$, in the robot-state $v_i^t \in L_i^t(S)$, is restricted to use only an arc from $\delta_{F_i}^+(v_i^t)$ for transitioning, refer A) for the reason. Suppose $r_i$ transitions using the arc

[1] When the robot-state is $\mathbf{o}$, the location in $V$ implied by the robot-state is unspecified. We will ignore this fact for the moment.




One possible choice of $S$ for the edge conflict between $r_1$ and $r_2$ is: $\mathcal{R}(S) = \{1, 2\}$, and $T(S) = \{3, 4, 5\}$. For all $i \in [1, 2]$ and $\forall t \in T(S)$, $L_i^t(S)$ is populated using the nodes in $N_i(t)$ corresponding to all locations in the $3 \times 3$ grid centered at $(3, 3)$ (yellow and red squares). $S$ for our example can be obtained from the parameters specified by applying Eqn (3.16). Clearly, the arcs in the edge conflict are present in $S$.

**Figure 3.4**

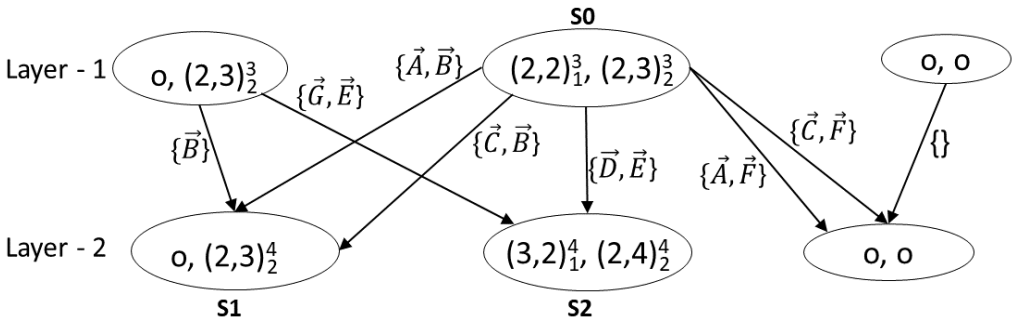



**Figure 3.5:** A portion of the DD for the choice of $S$ in Fig. 1 is shown above, where $\vec{A}$= $((2,2)^3_1, (2,1)^4_1)$, $\vec{B}$= $((2,3)^3_2, (2,3)^4_2)$, $\vec{C}$= $((2,2)^3_1, (1,2)^4_1)$, $\vec{D}$= $((2,2)^3_1, (3,2)^4_1)$, $\vec{E}$= $((2,3)^3_2, (2,4)^4_2)$, $\vec{F}$= $((2,3)^3_2, (1,3)^4_2)$, $\vec{G}$= $((3,1)^3_1, (3,2)^4_1)$.

$(v_i^t, w_i^{t+1})$, then the robot can transition to the robot-state:

$$\begin{cases} w_i^{t+1}, & \text{if } w_i^{t+1} \in L_i^{t+1}(S). \\ \mathbf{o}, & \text{otherwise.} \end{cases}$$

• At time $t$, the robot $r_i$ in the robot-state $\mathbf{o}$, using arc $(v_i^t, w_i^{t+1}) \in A_i(t) \cap S$, where $v_i^t \notin L_i^t(S)$, can transition to the robot-state $w_i^{t+1}$, if $w_i^{t+1} \in L_i^{t+1}(S)$. Otherwise, without using any arc in $A_i(t) \cap S$, the robot $r_i$ can transition into the robot-state $\mathbf{o}$ at time $t + 1$. Refer B) for why both these cases are possible. By our choice of designing $S$ using $\mathcal{R}(S), T(S), L(S)$ as shown in Eqn (3.16), the reader can verify that there is no arc in $A_i(t) \cap S$ that takes the robot $r_i$ from robot-state $\mathbf{o}$ at time $t$ to $\mathbf{o}$ at time $t + 1$.

**Example 4.** *Before continuing with the construction of $\mathcal{A}$, we take a brief detour. For the choice of parameters of $S$ in Figure 3.4, a portion of the corresponding DD that we want to construct using our procedure is depicted in Figure 3.5. Only a few states in layers $\mathcal{U}_1, \mathcal{U}_2$, and a few arcs in $\mathcal{A}_1$ with their labels, are illustrated. For robot $r_1$ to transition from location*

*$(2,2)$ at time 3 (denoted by state $(2,2)_1^3$) to $(3,2)$ at time 4, $r_1$ needs to transition using the arc $\vec{D}$ (refer Figure 3.5 for definition of $\vec{D}$). Likewise, for transitioning from robot-state $(2,3)_2^3$ to $(2,4)_2^4$, $r_2$ needs to transition using arc $\vec{E}$. Since transitioning between those robot-states for $r_1$ and $r_2$ does not cause a collision, we connect states **S0**, **S2** with an arc in $\mathcal{A}_1$ and label the arc with the set $\{\vec{D},\vec{E}\}$ as shown in Figure 3.5. From the definition of $L_1^4(S)$ provided in the caption of Figure 3.4, the reader can verify that both, $(2,1)_1^4$ and $(1,2)_1^4$ are $\notin L_1^4(S)$. So $r_1$ can transition to robot-state* **o** *at time 4 from $(2,2)_1^3$ using either $\vec{A}$ or $\vec{C}$. Consequently, $r_1, r_2$ can transition from **S0** to **S1** in 2 different ways as shown in Figure 3.5.*

With the help of the feasible robot-state transition rules, we are now equipped to populate $\mathcal{A}$. Consider any node $v \in \mathcal{U}_k$ and any node $w \in \mathcal{U}_{k+1}$. To decide whether we should connect $v$ to $w$ by arcs in $\mathcal{A}_k$, we need to first determine whether $w$ is a feasible (legal) state transition of $v$. We determine this by first checking, for each $i \in \mathcal{R}(S)$, whether the transition from $v[i]$ to $w[i]$ is legal. If the transition from $v[i]$ to $w[i]$ is illegal for any $i \in \mathcal{R}(S)$, then we can skip introducing any arc connecting $v$ to $w$ in $\mathcal{A}_k$. Otherwise, if we find all robot state transitions to be feasible, we will determine all the different ways in which the robots may transition from $v$ to $w$, and for each such possibility, we will consider introducing an arc in $\mathcal{A}_k$ connecting $v$ to $w$. By repeating this process for all pairs of nodes in $\mathcal{U}$ occurring between consecutive layers, we can populate $\mathcal{A}$. For $i \in \mathcal{R}(S)$, we will use the set $h(v,w,i) \subset S$ to establish all the different ways in which robot $r_i$, can transition from the robot-state $v[i]$ to $w[i]$ by traversing some arc in $S$. We can populate the set $h(v,w,i)$, by following the rules shown below:

1. If $k = 0$ and $w[i] \in L_i^{\mathbf{1}}(S)$, then first observe that $v[i] = \mathbf{o}$ since $v$ is the **sr** node. Robot $r_i$, can use any one of the arcs from the set $\delta_{F_i}^-(w[i])$ to transition to $w[i]$, and so $h(v,w,i) = \delta_{F_i}^-(w[i])$.
2. If $1 \leq k < |T(S)|$ , we consider separately the following 3 cases:

- If $v[i] \in L_i^{k+\mathbf{1}-1}(S)$, $w[i] \in L_i^{k+\mathbf{1}}(S)$ and $(v[i],w[i]) \in \delta_{F_i}^+(v[i])$, then: $h(v,w,i) = \{(v[i],w[i])\}$.
- If $v[i] = \mathbf{o}$ and $w[i] \in L_i^{k+\mathbf{1}}(S)$, then: $h(v,w,i) = \{(p_i^{k+\mathbf{1}-1}, w[i]) \in \delta_{F_i}^-(w[i]) | p_i^{k+\mathbf{1}-1} \notin L_i^{k+\mathbf{1}-1}(S)\}$.
- If $v[i] \in L_i^{k+\mathbf{1}-1}(S)$ and $w[i] = \mathbf{o}$, then: $h(v,w,i) = \{(v[i], p_i^{k+\mathbf{1}}) \in \delta_{F_i}^+(v[i]) | p_i^{k+\mathbf{1}} \notin L_i^{k+\mathbf{1}}(S)\}$.

3. If $k = |T(S)|$ (i.e., the penultimate layer of $\mathcal{D}(S)$) and $v[i] \in L_i^k(S)$, then $h(v,w,i) = \delta_{F_i}^+(v[i])$. Also note that $w$ is **sk** node, and so $w[i] = \mathbf{o}$.
4. If $v[i] = w[i] = \mathbf{o}$, then $h(v,w,i) = \emptyset$.

As $h(v,w,i)$ corresponds to the different ways in which the robot $r_i$ transitions from state $v[i]$ to $w[i]$, it is only natural that the elements of the set $H(v,w) = \prod_{i \in \mathcal{R}(S)} h(v,w,i)$ correspond to all the different ways in which robots associated to $\mathcal{R}(S)$ can transition from $v$ to $w$. Note that $H(v,w)$ is a Cartesian product of sets, and so each element of $H(v,w)$ is itself a subset (the empty set is also a subset) of $S$. For any $i \in \mathcal{R}(S)$, note that each element of $H(v,w)$ can contain at most one arc from $A_i$. Some elements of $H(v,w)$ may contain arcs from $S$, such that, robots transitioning using those arcs will result in an edge collision, and hence needs to be removed due to Eqn (3.20). Corresponding to each element remaining in $H(v,w)$ after the

previous edge collision filtration step, we add an arc $a$ from $v$ to $w$ in $\mathcal{A}_k$ and label (cf. $f$ function in definition of $\mathcal{D}(S)$) $a$ by the arcs from $S$ present in the element of $H(v, w)$.

The relationship between the DD $\mathcal{D}(S)$ we constructed, and $P(S)$, can be characterized as shown in Theorem 1. The 1:1 correspondence in the Theorem means that, if $x_v$ is a vertex of $P(S)$ and let $Q = \{a \in S | x_v(a) = 1\}$, then there is a $\mathbf{sr} - \mathbf{sk}$ path in $\mathcal{D}(S)$ such that the labels occurring on the path coincide exactly with $Q$. Conversely, for any $\mathbf{sr} - \mathbf{sk}$ path, if $\bar{S} \subset S$ are labels occurring on the path, then there is a vertex $x_v$ in $P(S)$ such that $x_v(\bar{S}) = 1, x_v(S \backslash \bar{S}) = 0$. We can easily prove Theorem 1, so we skip the formal derivation.

**Theorem 1.** *There is a 1:1 correspondence between vertices of $P(S)$, and $\mathbf{sr} - \mathbf{sk}$ paths in $\mathcal{D}(S)$.*

### 3.4.3 Performing the maximization in Eqn (3.28) over $P(S)$ using $\mathcal{D}(S)$

The problem in Eqn (3.28) is a linear optimization problem, so we know that at least one optimal solution lies at a vertex of $P(S)$. To compute the optimal objective value efficiently, we can exploit the correspondence between vertices of $P(S)$ and $\mathbf{sr} - \mathbf{sk}$ paths in $\mathcal{D}(S)$. Algorithmically, we assign a cost to each arc in $\mathcal{A}$ depending on the labels on the arc. For instance, if arc $a \in \mathcal{A}$ is labelled with $b_1, b_2$, where $b_1, b_2 \in S$, then we simply assign a cost of $d(b_1) + d(b_2)$ to $a$, where $d(b_i)$ is the co-efficient of $b_i \in S$ in the objective $d$ of Eqn (3.28). If $a$ is not labeled with any arcs from $S$, then we assign a cost of 0. After setting costs to all arcs in $\mathcal{A}$ in the manner just described, computing the maximum is equivalent to obtaining the longest $\mathbf{sr} - \mathbf{sk}$ path cost in this weighted DD $\mathcal{D}(S)$. The complexity for obtaining the longest path cost is $\mathcal{O}(|\mathcal{A}|)$, since $\mathcal{D}(S)$ is a directed acyclic graph.

### 3.4.4 Tightening the construction of $\mathcal{D}(S)$:

While our goal in constructing $\mathcal{D}(S)$ was to compactly represent the vertices of $P(S)$, recall that our interest in $P(S)$ was in providing a tight relaxation for $\text{Proj}_{x(S)}(\mathbf{P})$. Observe that $\text{Proj}_{x(S)}(\mathbf{P}) \subseteq P(S)$, and since both $P(S)$ and $\text{Proj}_{x(S)}(\mathbf{P})$ are $0-1$ polytopes, consequently, all vertices of $\text{Proj}_{x(S)}(\mathbf{P})$ are also vertices of $P(S)$. So we can obtain a tighter relaxation for $\text{Proj}_{x(S)}(\mathbf{P})$, by removing vertices of $P(S)$ lying outside $\text{Proj}_{x(S)}(\mathbf{P})$. In this section, we will perform such a tightening, but directly operating on $\mathcal{D}(S)$, by suitably removing some arcs from $\mathcal{A}$ and nodes from $\mathcal{U}$. We can then use the pruned $\mathcal{D}(S)$, instead, for performing the maximization in Section 3.4.3. Our strategy for pruning $\mathcal{D}(S)$ is as follows. Recall, from Theorem 1, we know that every vertex of $P(S)$ corresponds to some $\mathbf{sr} - \mathbf{sk}$ path in $\mathcal{D}(S)$. Given an arc $a \in \mathcal{A}$, consider the set of all vertices of $P(S)$ corresponding to $\mathbf{sr} - \mathbf{sk}$ passing through $a$, and suppose we can argue that all those vertices lie outside $\text{Proj}_{x(S)}(\mathbf{P})$, then eliminating those vertices from $P(S)$ is equivalent to removing $a$ from $\mathcal{D}(S)$.

We can prune the arc $a$ from $\mathcal{A}$, if it satisfies at least one of the conditions shown below:

AA) $a$ contains labels of the form $(v_i^{t-1}, w_i^t)$, $(u_j^{t-1}, w_j^t)$, where $i, j \in \mathcal{R}(S)$ are pairwise distinct. Note, $v, u$ are not restricted to correspond to different locations in $V$.

AB) $a$ contains labels of the form $(w_i^{t-1}, v_i^t)$, $(w_j^{t-1}, u_j^t)$, where $i, j \in \mathcal{R}(S)$ are pairwise distinct. Note, $v, u$ are not restricted to correspond to different locations in $V$.

If arc $a$ satisfies AA), we will explain why $a$ can be pruned from $\mathcal{A}$. The validity of AB) as a pruning criterion is analogous to AA). Observe that there can be no feasible point $x_p$ in $\text{Proj}_{x(S)}(\mathbf{P})$, such that, $x_p(v_i^{t-1}, w_i^t) = 1$, $x_p(u_j^{t-1}, w_j^t) = 1$, since it will trivially violate the vertex collision constraint (i.e., Eqn (3.6)) at $w$ at time $t$. Hence, any $\mathbf{sr} - \mathbf{sk}$ path in $\mathcal{D}(S)$ that contains the labels $(v_i^{t-1}, w_i^t)$, $(u_j^{t-1}, w_j^t)$, will correspond to a vertex of $P(S)$ that lies outside $\text{Proj}_{x(S)}(\mathbf{P})$, and so $a$ can be pruned. Despite the obvious vertex collision implied by the labels in $a$, we briefly mention why arc $a$ may have been included in the construction of $\mathcal{D}(S)$. $P(S)$ may have contained a vertex $x_q$, where, $x_q(v_i^{t-1}, w_i^t) = 1$, $x_q(u_j^{t-1}, w_j^t) = 1$, and $x_q$ does not violate Eqn (3.19), if $w_i^t \notin L_i^t(S)$ or $w_j^t \notin L_j^t(S)$.

For $i \in \mathcal{R}(S)$ and $t \in T(S)$, consider the case when $L_i^t(S) = N_i(t)$. Then, recalling Eqn (3.32), observe that all the $\mathbf{sr} - \mathbf{sk}$ paths in $\mathcal{D}(S)$ corresponding to vertices of $\text{Proj}_{x(S)}(\mathbf{P})$ do not pass through the robot-state $\mathbf{o}$ of robot $r_i$ at time $t$. Hence, if $L_i^t(S) = N_i(t)$, we can remove all those nodes (states) from the layer of $\mathcal{U}$ corresponding to time $t$, where the robot-state of $r_i$ in the node is $\mathbf{o}$.

## 3.5 Integration in Conflict Based Search

In this section, we will briefly introduce the conflict-based search (CBS) method, which is currently amongst the state-of-the-art algorithms for solving the MAPF problem optimally. Then, we will demonstrate how we can incorporate the Lagrangian Relax-and-Cut (LRC) scheme (refer Section 3.2) into CBS, to boost the performance of CBS.

### 3.5.1 Conflict-based search

CBS [Sharon *et al.*, 2015] is an algorithm to compute the optimal solution for the MAPF problem. CBS performs best first search on a search tree. Each node in the search tree is characterized by a set of arcs in $A$ (refer Section 3.1.1) that the robots are prohibited from using, and considered as constraints in the search tree node. The shortest path for each robot obeying the constraints in the search tree node is either computed or inherited from its parent, and based on those paths, the pairwise conflicts between robots are identified and stored in the tree node. During search, the leaf nodes in the search tree are evaluated using an evaluation function, and the most promising node is selected for exploration. Once a leaf node is selected for exploration, one of the pairwise conflicts in the node is chosen by the algorithm with the hope of resolving it. In case the conflict chosen for resolution is a vertex conflict (resp. edge conflict), we know that at least one of the conflicting robots must be prohibited from accessing the conflict location (resp. conflict edge) at the conflict time in the optimal solution. So the optimal solution must satisfy the 2 term disjunctive constraint, where each term (constraint) prohibits exactly one of the conflicting robots from accessing the conflict location (resp. edge) at the conflict time. Two child nodes, one for each constraint, is created in the search tree. Additionally, the child nodes inherit all the constraints from their parent also. The search ends when the search tree node chosen for exploration contains no conflicts.

Several innovations have been proposed to improve the performance of the CBS scheme described above. The first improvement is to the node evaluation function. The node evalu-

ation function takes a search tree node as input, and basically outputs a lower bound to the cost of the optimal MAPF solution, but also subject to the constraints in the search tree node. The expectation is that, if the output of the node evaluation function closely matches the true lower bound at every search tree node, then the search would expand fewer nodes in the search tree. Many node evaluation functions have been proposed, see [Felner *et al.*, 2018; Li *et al.*, 2019a]. Amongst them, the WDG measure proposed in Li *et al.* [2019a] outputs the tightest lower bound. In this work, we incorporate LRC into CBS as a node evaluation function, so that in combination with WDG, the ensemble is a stronger node evaluation function. More details concerning WDG and the LRC based evaluation function is presented in Section 3.5.2.

In the basic version of CBS, an arbitrary pairwise conflict stored in the node was chosen for branching. A better strategy was proposed in Boyarski *et al.* [2015], where before branching, they classify the conflicts in the search tree node into different priorities. The idea is to assign a higher priority to those conflicts, that generate child nodes that can help improve the lower bound at a faster rate. In this work, we derive additional conflicts from start-end robot paths minimizing a Lagrangian objective, and consider them also during branching. Details are presented in Section 3.5.2.

The third type of improvement to CBS falls in the category of constraint generation schemes for resolving conflicts. In the basic scheme, depending on whether the conflict was an edge or a vertex conflict, a disjunctive constraint was generated. In Li *et al.* [2020a], for the case when a conflict occurs at the goal location of one of the conflicting robots, or at a corridor location, the authors demonstrate that the conflict resolution scheme of CBS would generate a very large number of conflicts, and create a very large search tree. The authors in Li *et al.* [2020a] propose a more sophisticated constraint generation scheme for both these cases, that effectively preempts the explosion in the number of conflicts. In Section 3.7.3, we will show some preliminary work on a constraint generation scheme that makes use of the information present in the objective cuts, and delay-and-long cuts.

### 3.5.2 Integrating Lagrangian Relax-and-Cut with CBS

There are 3 steps to integrating the LRC scheme introduced in Section 3.2 into CBS. We will begin with a high level description of these steps.

1. Before we apply CBS to our problem, we apply Algorithm 2, and let inequalities $\hat{E}x \leq \hat{f}$, optimal Lagrangian multipliers $\hat{\lambda}$, and upper bound UB denote the outputs of the algorithm.

2. We will formulate a new node evaluation function called LR-WDG, which makes use of the output from Algorithm 2. LR-WDG is described below. Given a search tree node $sn$, we use LR-WDG in conjunction with WDG to evaluate the node, so the evaluation of $sn$ is given by max(WDG($sn$), $\lceil$LR-WDG($sn$)$\rceil$), where $\lceil\cdot\rceil$ denotes the ceiling function.

3. In each search tree node, note that the basic CBS procedure stores the shortest start-end path for each robot subject to the constraints in the node. Pairwise conflicts are derived by analyzing these shortest paths. We will store an additional start-end path for each robot, subject to the constraints in the search tree node, that instead minimizes the Lagrangian objective $L(x, \lambda)$ shown in Algorithm 2. Given a search node $sn$ in the search tree, let us denote the set of arcs

that are prohibited in $sn$ by $\bar{\mathcal{A}}_{sn} \subset \cup_{i\in[\mathbf{N}]} A_i$, then the paths for the robots minimizing the Lagrangian objective is obtained by solving:

$$\hat{h}_1(sn) = \min_{\substack{x\in\{0,1\}^{|A|} \\ x(a)=0, \forall a\in\bar{\mathcal{A}}_{sn}}} \{c^\mathsf{T}x + \hat{\lambda}^\mathsf{T}(\hat{E}x - \hat{f}) | x \text{ satisfies Eqns } (3.4) - (3.5)\} \tag{3.33}$$

We compute conflicts from the Lagrangian optimal paths, and add them to the set of conflicts that we will consider for branching. Note that $\hat{h}_1(\cdot)$ in Eqn (3.33) is itself also a valid node evaluation function. In Boyarski *et al.* [2015], conflicts for branching were classified by quickly evaluating the child nodes that would have been created by branching on a conflict using the node evaluation function $h(\cdot)$. $\hat{h}(\cdot)$ and $h(\cdot)$ differ only in the objective that is minimized, $h(\cdot)$ optimizes only for $c^\mathsf{T}x$ in the objective of Eqn (3.33). The idea to include conflicts from the Lagrangian optimal paths is that, in case we are unable to identify conflicts that generate child nodes with a higher $h(\cdot)$ value than $h(sn)$, we may still be able to identify a conflict that generates child nodes with a higher $\hat{h}(\cdot)$ than $\hat{h}(sn)$.

### Lagrangian based node evaluation function

As motivated earlier in Section 3.2, we can use the Lagrangian information to derive strong lower bounds for our problem. In a similar vein, we will use it to devise a node evaluation function in this section. Such an approach is not new in Constraint Programming literature, see [Benoist *et al.*, 2001; Bergman *et al.*, 2015; Khemmoudj *et al.*, 2005] for other examples. While $\hat{h}_1(\cdot)$ is an example of such a Lagrangian based node evaluation function, in this section, we will present a stronger node evaluation function, inspired directly from the minimum vertex cover (MVC) function of [Felner *et al.*, 2018] and WDG function of [Li *et al.*, 2019a]. We first explain the WDG function, and then present our node evaluation function.

For a given search tree node $sn$, the WDG function from Li *et al.* [2019a], is based on pairwise costs for the robots. For each pair of robots we can solve a 2-Agent MAPF problem disregarding all other robots. For $i, j \in [\mathbf{N}]$, let us denote $l_{sn}(i, j)$ to be the sum of the shortest conflict-free path costs of robots $r_i, r_j$ that also satisfy the constraints in the search tree node $sn$. We can compute $l_{sn}(i, j)$ using the formulation shown below.

$\boxed{l_{sn}(i,j)}$

$$\min_{x\in\{0,1\}^{|A|}} \sum_{a\in A_i\cup A_j} c(a)x(a) \tag{3.34}$$

$$\text{s.t.} \sum_{a\in\delta^+_{F_k}(s^0_k)} x(a) = 1,\ k \in \{i, j\} \tag{3.35}$$

$$x \text{ satisfies Eqns (3.5) - (3.7)} \tag{3.36}$$

$$x(a) = 0, \forall a \in \bigcup_{k\in[\mathbf{N}]\setminus\{i,j\}} A_k \tag{3.37}$$

$$x(a) = 0, \forall a \in \bar{\mathcal{A}}_{sn} \cap \{A_i \cup A_j\} \tag{3.38}$$

$\boxed{\text{WDG}(sn)}$

$$\min_{cost(k)\in\mathbb{Z}_+} \sum_{k\in[\mathbf{N}]} cost(k) \tag{3.39}$$

$$\text{s.t.} \quad cost(i) + cost(j) \geq l_{sn}(i, j), \quad \forall i, j \in [\mathbf{N}], \text{ and } i < j \tag{3.40}$$

$$cost(k) \geq LB(k, sn),\ \forall k \in [\mathbf{N}] \tag{3.41}$$

For search tree node $sn$, any valid node evaluation function must output a lower bound to the problem shown in Eqn (3.42).

$$\min_{x \in \mathbf{P}} \{c^\mathsf{T} x | x(a) = 0,\ \forall a \in \bar{\mathcal{A}}_{sn}\} \tag{3.42}$$

Given an optimal solution for the problem shown in Eqn (3.42), for each $k \in [\mathbf{N}]$, denote the cost of the start-end path for robot $r_k$ in the optimal solution by $cost(k)$, then these costs must satisfy Eqn (3.40). $LB(k, sn)$ in Eqn (3.41) denotes the shortest start-end path cost for robot $r_k$ subject to the constraints in $sn$, and so $cost(k)$ must trivially satisfy Eqn (3.41). The WDG function value evaluated at $sn$ is obtained by finding integer assignments for the $cost(\cdot)$ variables, subject to Eqns (3.40) - (3.41), that minimizes the sum of the cost variables in Eqn (3.39). Computing WDG($sn$) is known to be NP-Hard, since it is at least as hard as solving a minimum vertex cover problem.

We next present our node evaluation function, which will refer to as LR-WDG, short for Lagrangian relaxation WDG. Similar to the approach taken in deriving the WDG bound, for each pair of robots we can solve a 2-Agent MAPF problem, but instead of minimizing the MAPF objective $c$ as we did in Eqn (3.34), we will instead minimize the Lagrangian objective. For $i, j \in [\mathbf{N}]$, analogous to $l_{sn}(i, j)$, we will denote the pairwise cost for robots $r_i, r_j$ minimizing the Lagrangian objective by $l^{\hat{\lambda}}_{sn}(i, j)$. We can use a 2-Agent MAPF solver to compute $l^{\hat{\lambda}}_{sn}(i, j)$. We simply have to modify the cost of each arc in $A_i, A_j$ to the arc's co-efficient in $c^\mathsf{T} x + \hat{\lambda}^\mathsf{T} \hat{E} x$, and then apply the solver to retrieve the optimal objective value.

$\boxed{l^{\hat{\lambda}}_{sn}(i, j)}$

$$\min_{x \in \{0,1\}^{|A|}} c^\mathsf{T} x + \hat{\lambda}^\mathsf{T} \hat{E} x \tag{3.43}$$

$$\text{s.t. } x \text{ satisfies Eqns (3.35) - (3.38)} \tag{3.44}$$

$\boxed{\text{LR-WDG}(sn)}$

$$\min_{y \in \mathbb{R}^{\mathbf{N}}} -\hat{f}^\mathsf{T} \hat{\lambda} + \sum_{k \in [\mathbf{N}]} y_k$$

$$\text{s.t. } y_i + y_j \geq l_{ij}(\hat{\lambda}, sn), \quad \forall i, j \in [\mathbf{N}] \text{ and } i < j \tag{3.45}$$

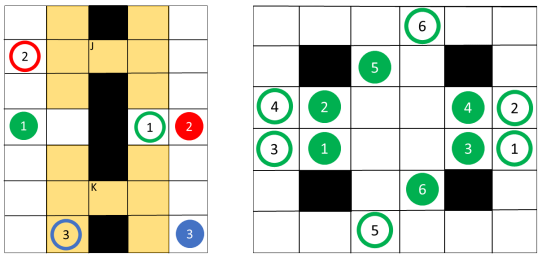
$$y_k \geq LB_{\hat{\lambda}}(k, sn), \forall k \in [\mathbf{N}] \tag{3.46}$$

Using the pairwise Lagrangian costs $l_{ij}(\hat{\lambda}, sn)$, we specify LR-WDG($sn$). Analogous to $LB(k, sn)$, let $LB_{\hat{\lambda}}(k, sn)$ denote the shortest start-end path cost for robot $r_k$, subject to the constraints in the search node $sn$, where the cost of using an arc $a \in A_k$ is given by the coefficient of $x(a)$ in $c^\mathsf{T} x + \hat{\lambda}^\mathsf{T} \hat{E} x$. We get LR-WDG($sn$) by solving the linear program shown above. Note that since the optimal value of Eqn (3.42) is integral, Proposition 1 still holds if we replace LR-WDG($sn$) by $\lceil$LR-WDG($sn$)$\rceil$. Finally, note that $\hat{h}_1(sn) = -\hat{f}^\mathsf{T} \hat{\lambda} + \sum_{k \in [\mathbf{N}]} LB_{\hat{\lambda}}(k, sn)$, and so LR-WDG($sn$) is $\geq \hat{h}_1(sn)$, owing to Eqn (3.46).

**Proposition 1.** *LR-WDG($sn$) is a valid lower bound to the problem shown in Eqn (3.42).*

*Proof.* Proof in Appendix, refer Section A.1. □

**Example 5.** *We show some examples comparing WDG and $\hat{h}_1(\cdot)$ evaluated at the root node of the search tree. In Figure 3.6a, note that locations J, K act as bottleneck regions. Robot 1 has 2 shortest routes, one via J and the other via K, while robots 2's (resp. 3's) shortest route is via J*




(a) (b)

$$cost(1) + cost(3) \geq 10 \quad (3.47)$$
$$cost(2) + cost(4) \geq 10 \quad (3.48)$$
$$cost(2) + cost(5) \geq 9 \quad (3.49)$$
$$cost(2) + cost(6) \geq 9 \quad (3.50)$$
$$cost(3) + cost(5) \geq 9 \quad (3.51)$$
$$cost(3) + cost(6) \geq 9 \quad (3.52)$$
$$cost(1) + cost(3) + cost(6) \geq 13 \quad (3.53)$$
$$cost(2) + cost(4) + cost(5) \geq 13 \quad (3.54)$$

**Figure 3.6:** The grids in both figures are 4-connected. Solid circles indicate start locations, and rings indicate goal locations. The yellow regions in 3.6a are representative of the rst-neighborhoods regions used for generating cuts. The neighborhood on top was parameterized with robots 1 and 2, while the neighborhood below was parameterized using robots 1 and 3. In Figure 3.6b, neighborhoods parameterized by 2 and 3 robots were used for generating cuts.

*(resp. $K$). Due to multiple shortest routes available for robot 1, it is easy to see that the WDG bound coincides with the sum of the shortest route costs for the robots, which is 18, whereas the $\hat{h}_1(\cdot)$ bound is 20, which is also the optimal solution cost to this problem. Intuitively, the cut generated by the rst-neighborhood centered at J, tells us that, if in the optimal solution robots 1 and 2 passes through J, then one among those robots must have waited for the other robot to pass through first. From the cut generated by the rst-neighborhood centered at K, a similar observation can be made for robots 1 and 3. Note that WDG is unable to make a similar such inference on this problem, and so the WDG bound is weaker than $\hat{h}_1(\cdot)$.*

*For the case shown in Figure 3.6b, the optimal cost for the problem is 30. The WDG value for the root node is 28, while the $\hat{h}_1(\cdot)$ value is 29. Since WDG is based on pairwise robot costs, the WDG value is based on Eqns (3.47) - (3.52). Refer to the description of WDG for the definition of $cost(\cdot)$ in those equations. When we applied LRC (i.e., Algorithm 2) to the problem in Figure 3.6b, some cuts (i.e., $\hat{E}x \leq \hat{f}$) used in Eqn (3.33) were generated from rst-neighborhoods parameterized with 3 robots, to obtain better bounds. Using those cuts, we were also able to infer Eqns (3.53) - (3.54). The procedure to make such inferences is described in the next section. Intuitively, the $\hat{h}_1(\cdot)$ value was based on more information than WDG in this case, and so, $\hat{h}_1(\cdot)$ was stronger.*

### Strengthening WDG via Lagrangians

Recall from Eqn (3.40), the WDG bound was based on pairwise robot costs. In this section, we will generate inequalities of the form shown in Eqn (3.55), and add them to the WDG($sn$) formulation, with the hope of strengthening it. To strengthen WDG($sn$), we present a simple greedy procedure to identify subsets $Cl$ in Eqn (3.55), such that, Eqn (3.55) is not implied by Eqns (3.40), (3.41). In Example 5, Eqns (3.53) - (3.54) were examples of such inequalities, with $Cl = \{1, 3, 6\}$ in Eqn (3.53). For convenience, we shall refer to $Cl$ as a cluster.

$$\sum_{k \in Cl} cost(k) \geq val, \text{ where } Cl \subseteq [\mathbf{N}], |Cl| > 2, \text{ and } val \in \mathbb{Z}_+ \quad (3.55)$$

Before presenting the algorithm, we introduce some notation. Given an inequality $\sum_{a\in A} e(a)x(a) \leq f$, let $Rob(e^{\mathsf{T}}x \leq f) \subseteq [\mathbf{N}]$ denote the cluster, such that, $k \in Rob(e^{\mathsf{T}}x \leq f)$, iff, $\exists\, a \in A_k$ with $e(a) \neq 0$. Observe that Eqn (3.33) provides a lower bound for the optimal sum of conflict-free start-end path costs of all $\mathbf{N}$ robots. In Eqn (3.55), we are interested in a lower bound for the optimal sum of conflict-free start-end path costs for a cluster of robots $\{r_k\}_{k\in[Cl']}$, where $Cl' \subseteq [\mathbf{N}]$. A straightforward way to compute such a lower bound, is to modify the optimization problem shown in Eqn (3.33). We discard from its objective all those inequalities $e^{\mathsf{T}}x \leq f$ from $\hat{E}x \leq \hat{f}$, such that, $Rob(e^{\mathsf{T}}x \leq f) \not\subseteq Cl'$. We also add the constraint $x(a) = 0, \forall\, a \in \underset{k\in[\mathbf{N}]\setminus Cl'}{\cup} A_k$. With those changes to the optimization problem in Eqn (3.33), we denote its optimal value by $\hat{h}_1(sn, Cl')$.

Algorithm 4 provides a simple iterative procedure for identifying clusters. The algorithm takes a search node $sn$, the number of iterations, and the output of Algorithm 2 as inputs. The algorithm starts with a set of clusters, stored in $Gr$, and as the iterations progress, each cluster in $Gr$ is enlarged by at most one element at every iteration. An obvious set of clusters to initialize $Gr$, is the set of robots associated with each cut that was generated in Algorithm 2, as shown in lines 4 - 5. In line 10, to check if a cluster $Cl$ can be enlarged by including some robot $r_k$, we compute the difference between the lower bounds computed using Lagrangians and the sum of the shortest start-end paths. If the difference is non-negative, then we know that Eqn (3.55) generated for cluster $Cl \cup \{k\}$ will not be implied by Eqn (3.41), and so we choose to enlarge $Cl$ by adding $k$ as shown in line 12.

**Example 6.** *In Example 5, the rst-neighborhood centered at J (resp. K) gave us a cut involving robots 1 and 2 (resp. 1 and 3). If we apply Algorithm 4 to that problem, $Gr$ in line 4 would have been initialized with the clusters $\{1, 2\}$ and $\{1, 3\}$. In the first iteration of Algorithm 4, both these clusters would have been enlarged to form the same cluster, i.e., $\{1, 2, 3\}$, since we previously noted in Example 5 that $\hat{h}(rn) = 20 > (7 + 6 + 5)$, where $rn$ is the root node, $LB(1, rn) = 7$, $LB(2, rn) = 6$ and $LB(3, rn) = 5$.*

A noticeable limitation of Algorithm 4 is that, owing to the greedy nature of our procedure, there may be a high degree of overlap between clusters, and its effect is detrimental to strengthening the WDG bound. To mitigate this behavior, in our implementation, we enforce the constraint that no two clusters in $Gr$ are permitted to overlap greater than a certain cardinality, where the threshold is specified as an input to Algorithm 4. This feature can be easily implemented by restricting the domain of $k$ in lines 10 and 12 appropriately.

Finally, observe that extracting clusters by executing Algorithm 4 at every node of the search tree is computationally prohibitive. Instead, in our implementation, we applied Algorithm 4 only to the root node $rn$ of the search tree, and extracted clusters. The inequalities of the form shown in Eqn (3.55) generated from each of these clusters, was then added to the constraints in the computation of WDG($sn$), at every node $sn$ in the search tree. Notice that for the problem shown in Figure 3.6a, the inequality $cost(1) + cost(2) + cost(3) \geq 20$ generated from the cluster $\{1, 2, 3\}$, when added to formulation of WDG($rn$), lifts the WDG bound for the root node from 18 to the optimal MAPF cost which is 20.

**Algorithm 4** Cluster computation for strengthening WDG

**Given:** Search node $sn$, Max Iter, Output of Algorithm 2, i.e., $\hat{E}x \leq \hat{f}$, $\hat{\lambda}$
**Initialize:** $Gr \leftarrow \emptyset$
**for each** inequality $e^{\mathsf{T}}x \leq f \in \hat{E}x \leq \hat{f}$ **do**
  $Cl \leftarrow Rob(e^{\mathsf{T}}x \leq f)$
  $Gr \leftarrow Gr \cup Cl$  { $Gr$ is a set of clusters}
**for** *iter* = 1 : Max Iter **do**
  $Gr' \leftarrow \emptyset$
  **for each** $Cl \in Gr$ **do**
    $Cl' \leftarrow Cl$
    $\Delta \leftarrow \max_{k \in [\mathbf{N}] \setminus Cl} In(sn, Cl, k)$, where $In(sn, Cl, k) = \hat{h}_1(sn, Cl \cup k) - \sum_{j \in \{Cl \cup k\}} LB(j, sn)$
    **if** $\Delta \geq 0$ **then**
      $Cl' \leftarrow Cl \cup \{\arg\max_{k \in [\mathbf{N}] \setminus Cl} In(sn, Cl, k)\}$
    $Gr' \leftarrow Gr' \cup Cl'$
  $Gr \leftarrow Gr'$
**return** $Gr$

## 3.6 Experimental Evaluation

### 3.6.1 Experimental Setup

In this section, we provide the description of the algorithms with their parameters that we tested, and the layouts on which the experiments were performed.

#### Algorithms and their parameters

We implemented and tested 4 variants of the CBS algorithm, namely WDG-CBS, LR-WDG, WDG-G, and LR-WDG-G. We explain what each of these variants mean separately below.

In the variant WDG-CBS, we implemented the basic version of the CBS i.e., [Sharon *et al.*, 2015], with the following enhancements. We implemented the conflict classification and branching rules from the ICBS algorithm proposed in Boyarski *et al.* [2015]. In case of ties in the conflict classification rule, following the suggestion in Li *et al.* [2020a], we prioritized corridor conflicts over other conflict types. From Li *et al.* [2019a], we implemented the WDG node evaluation function (see Section 3.5.2 for description) with memoization. We included the constraint generation scheme for corridor conflicts from Li *et al.* [2020a]. Recall that the MAPF objective we consider allows a robot to freely wait at its goal. This property rendered the target conflict constraint generation scheme from Li *et al.* [2020a] inapplicable for this work. With these enhancements to CBS, we believe our implementation of WDG-CBS is representative of the current state-of-the-art variant for CBS. For our experimental results comparison, WDG-CBS will serve as the baseline method, and we will compare the

performance of the other three algorithms against this variant.

For the LR-WDG variant, we integrated the LRC scheme into WDG-CBS, by following the three steps mentioned in Section 3.5.2, with one small modification to step 3. For any search node $sn$, we consider branching on conflicts derived from the Lagrangian optimal paths, only when LR-WDG($sn$) $>$ WDG($sn$). Since the cuts from Algorithm 2 were derived at the root node of the search tree, the utility of the cuts in determining the lower bound for a search tree node diminishes as the depth of the search tree node increases. Consequently, the gap between LR-WDG($sn$) and WDG($sn$) decreases as the depth of $sn$ increases. We also strengthened the WDG function using the procedure presented in Section 3.5.2. The rst-neighborhoods used for generating cuts are described in Section 3.6.1.

The WDG-G and LR-WDG-G variants are simple extensions of WDG-CBS and LR-WDG respectively. We introduce this algorithm to mitigate a limitation of CBS. In certain situations, during CBS, the same pair of agents generates a large number of conflicts, thereby resulting in a large search tree, see Target Conflicts in Li *et al.* [2020b] for an example. To mitigate this issue, in WDG-G and LR-WDG-G, we pair some robots and treat each pair as a single meta-agent. When planning a path for a meta-agent, we compute paths for both the robots in the pair, such that, the paths are mutually conflict-free. We compute this pairing of robots only once, and use the same pairing throughout the search. The pairing procedure is explained in the next paragraph. Our idea of pairing can be seen as a special case of meta-agent algorithms. Coupling groups of robots into meta-agents is not new in CBS, see Sharon *et al.* [2012]. To keep computational costs of WDG-G and LR-WDG-G comparable to WDG-CBS and LR-WDG, the node evaluation (i.e. WDG($\cdot$) and LR-WDG($\cdot$)) for each search tree node was performed identically to the way we evaluated in WDG-CBS and LR-WDG, i.e., instead of computing Eqns (3.40), (3.45) for pairs of (possibly) meta-agents, we compute them for only pairs of robots. Consequently, WDG-G is not naturally better that WDG-CBS in terms of lower bound. However, the same is not true between LR-WDG and LR-WDG-G. When performing the LRC step in LR-WDG-G, in lines 4, 5 of Algorithm 2, we have to additionally ensure that the paths computed for robots mapped to the same meta-agent, must be mutually conflict free. We give a geometric interpretation to understand the consequences. Recall from Section 3.2, in the LRC phase of LR-WDG, we are computing a tight relaxation to $\mathbf{P}$ by iteratively adding cuts to $P_0$ (refer Section 3.2 for definition) and tightening it. In LR-WDG-G however, we start with a tighter relaxation $Q_0 \subseteq P_0$, and in Algorithm 2 we are adding cuts to tighten $Q_0$. Empirically, the lower bound after the LRC step is generally higher in LR-WDG-G than compared to LR-WDG.

The pairs are obtained by first constructing a graph, where each node in the graph represents a robot. Nodes (robots) $i, j$ (where $i, j \in [\mathbf{N}], i \neq j$) are connected by an edge if $diff(i, j) > 0$, where $diff(i, j) = l_{rn}(i, j) - (LB(i, rn) + LB(j, rn))$, and $rn$ refers to the root node of the search tree. The reader can refer to Section 3.5.2 for the definitions of $l.(\cdot, \cdot), LB(\cdot, \cdot)$. The value $diff(i, j)$ represents how much robots $i, j$ constrain each other. After constructing the graph, we compute a maximum weighted matching. Robots that are not connected to any edge in the optimal matching are treated as singleton agents, while robots connected to an edge in the optimal matching are paired together to form a meta-agent. The motivation behind this pairing is that, a robot pair with a high $diff(\cdot, \cdot)$ score is likely to produce numerous conflicts during CBS involving each other, and so by pairing those robots, we

can avoid branching on those conflicts during CBS.

### Layouts and Problem Instances

For testing the algorithms, we considered 4 types of layouts, namely *Random*, *Empty Room*, and *Maze*. For generating the *Random* layouts, we considered a $30 \times 30$ 4-connected grid, and some % of locations on the grid were randomly chosen to be stationary obstacles. The obstacle percentages considered were 10, 15, 20 and 25. For the *Empty* layout, we tested on the layout *empty-32-32.map*, for the *Room* layout, we tested on the layout *room-32-32-4.map*, and for the *Maze* layout, we tested on the layout *maze-32-32-2.map*. Note that all 3 of those named layouts are 4-connected grids, and taken from Stern *et al.* [2019].

For all layout types, we considered problem instances with a varying number of robots. For the *Random* layout type, we generated 25 problem instances for a given number of robots and obstacle %. The layouts across the 25 instances differ, and on each instance the start and goal locations for the robots are randomly chosen independently of other instances. For the *Empty*, *Room*, and *Maze* layout types, we considered the problem instances provided in [Stern *et al.*, 2019], in particular we performed testing on the *even* scenarios.

For every problem instance, we computed the shortest time it takes for every robot to reach its goal location, assuming all other robots are absent. We then set the makespan constraint $\mathbf{T}$ to be 3 more than the time it took for the robot which took the longest to reach its goal. We confirmed that all the *Random* problem instances generated are feasible.

### Templates for 4-connected grids

Recall, that to generate cuts from a conflict, we need to specify a rst-neighborhood and construct the corresponding DD for the neighborhood. Constructing a DD is expensive, but fortunately, since all the layouts that we consider in our experiments are 4-connected grids, we can exploit some structure. When the underlying layout $G$ (refer Section 3.1) is a 4 or 8-connected grid, the neighborhood relative to any location on the grid is same across all locations on the grid, a property that allows us to build *Templates*.

To understand what we mean by *Templates*, consider the following. Let us denote the polytope $P(S)$ described in Figure 3.4 by $P_1$. Now consider the vertex conflict for robots $r_3$ and $r_4$ at time 14 shown in Figure 3.4. For this conflict, we can create a polytope $P(S_2)$ with parameters : $\mathcal{R}(S_2) = \{3, 4\}$, and $T(S_2) = \{13, 14, 15\}$. For all $i \in [3, 4]$ and $\forall t \in T(S_2)$, $L_i^t(S_2)$ is set to nodes in $N_i(t)$ corresponding to all locations in the $3 \times 3$ grid centered at $(6, 5)$. Clearly, $P(S_2)$ can also output a cut for the conflict between $r_3, r_4$. While polytopes $P_1, P(S_2)$ lie in different dimensions, the facial structure of both polytopes are identical. If we substitute $r_1$ for $r_3$, $r_2$ for $r_4$, advance the interval $T(S_2)$ by 10 time units, and translate all locations in $L_\cdot(S_2)$ by 3 units along the negative X-axis and by 2 units along the negative Y-axis, we get back all the parameters for $S$ described in Figure 3.4. Hence, we claim that both $P_1$ and $P(S_2)$ are manifestations of the same base template polytope, and the base template can be visualized as the square shaped yellow region in Figure 3.4. So suppose we previously constructed the DD $\mathcal{D}(S_1)$ for $P_1$, we can simply spatio-temporally shift the parameters of $\mathcal{D}(S_1)$ to get the DD $\mathcal{D}(S_2)$ for $P(S_2)$, without computing $\mathcal{D}(S_2)$ from scratch.

While working with structured graphs such as grids, we can precompute a library of different templates, along with their corresponding DDs. Given a conflict, we can simply choose one of the templates from the library for generating cuts. By spatio-temporally shifting the parameters of the template about the conflict, multiple cuts can be generated using the same base template. Note that some locations around the conflict location(s) may be physically blocked. For example, assume location $(7, 6)$ in Figure 3.4 is blocked. For the conflict between $r_3, r_4$, we can still use the yellow template in Figure 3.4 with the following adjustment to the cut generation procedure. When computing the RHS of a cut, i.e., when computing the longest $\mathbf{sr} - \mathbf{sk}$ path on the DD, we have to avoid arcs on the path that pass through infeasible states in the DD. A state in the DD is said to be infeasible if it corresponds to a location for a robot that is physically blocked.

For generating the results in Section 3.6.2, our template library consisted of 65 templates, of which one of the templates was parameterized by 2 robots. Denoting the 2 robot template by $S_1$, the parameters of the template were, $|T(S_1)| = 7$, $|L_i^t(S_1)| = 25, \forall i \in \mathcal{R}(S_1)$, and $\forall t \in T(S_1)$. This translated roughly into 80,000 arcs in the DD for the template. The remaining 64 templates were parameterized by 3 robots, where each template $S$ was designed with $|T(S)| = 5$, and $|L_i^t(S)| = 6$, $\forall i \in \mathcal{R}(S)$, and $\forall t \in T(S)$. Per template, this translated roughly into 0.2 million arcs in the DD. In the appendix section, we provide a visual representation of the templates in the library.

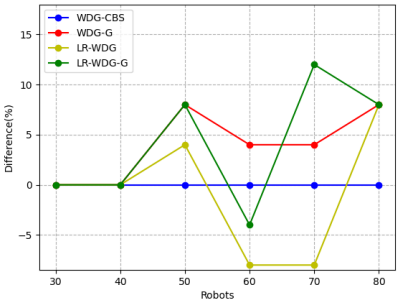
### 3.6.2 Experimental Results

All experiment results reported in this chapter were carried out on an Intel 4 core i7-4790 processor running at 3.6 GHz with 16 GB RAM, and the code was written in C++ as a single-threaded application. For generating objective and delay-and-long cuts, we set Max Offset in Algorithm 3 to 9. For a given a conflict, we used both 2 and 3 robot templates in line 4 of Algorithm 3 to generate objective cuts. For generating delay-and-long cuts, we only used the 2 robot template. The parameter $T(S)$ for the $S$ chosen in line 4 was such that, the *conflict time* in line 9 lies in the middle of the interval $T(S)$. While solving the optimization problem in line 4 of Algorithm 2 using projected subgradient ascent, we used A* to solve the inner min-cost flow problem efficiently. We allocated 15 minutes to each of the 4 algorithms for every problem instance. In case of LR-WDG and LR-WDG-CBS, the 15 minutes was further divided as follows. We set the termination criterion in Algorithm 2 to 20 iterations, and a time limit of 10 minutes, whichever terminated earlier.

#### Results on Random Instances

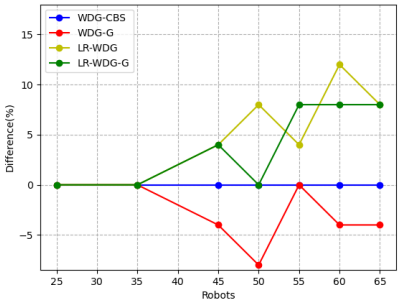
In terms of the number of problems that are solved to optimality, Figure 3.7 provides a comparison of the 4 algorithms with WDG-CBS treated as the baseline. We make the following observations from these plots. When the obstacle % is $> 10$, in most cases, LR-WDG dominates WDG-CBS, and LR-WDG-G dominates WDG-G. Further, as the number of robots increases, we generally find that LR-WDG performs better than both WDG-G and WDG. Although there is no algorithm that always dominates the other algorithms, on the whole, LR-WDG-G appears to be the best performing algorithm across all obstacle percentages. These

| Robots | Opt (%) |
|---|---|
| 30 | 100 |
| 40 | 92 |
| 50 | 76 |
| 60 | 40 |
| 70 | 48 |
| 80 | 12 |



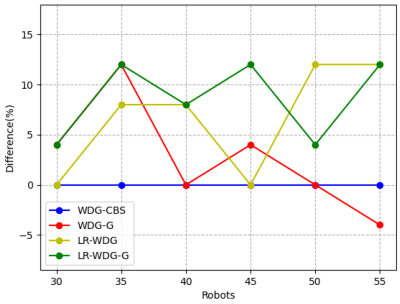

**(a)** 10% Obstacles

| Robots | Opt (%) |
|---|---|
| 25 | 100 |
| 35 | 92 |
| 45 | 68 |
| 50 | 72 |
| 55 | 44 |
| 60 | 24 |
| 65 | 8 |



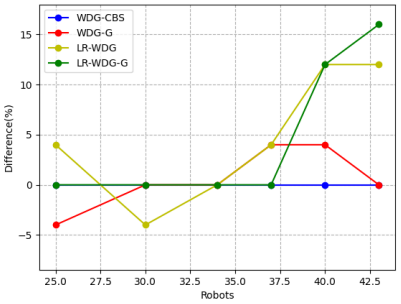

**(b)** 15% Obstacles

| Robots | Opt (%) |
|---|---|
| 30 | 96 |
| 35 | 72 |
| 40 | 60 |
| 45 | 56 |
| 50 | 24 |
| 55 | 8 |




**(c)** 20% Obstacles

| Robots | Opt (%) |
|---|---|
| 25 | 88 |
| 30 | 60 |
| 34 | 68 |
| 37 | 40 |
| 40 | 24 |
| 43 | 12 |




**(d)** 25% Obstacles

**Figure 3.7:** For each obstacle %, the table on the left indicates the % of problems solved to optimality by WDG-CBS. In each graph, for each algorithm, we plot the difference between the % of problems solved to optimality by the algorithm and the % of problems solved to optimality by WDG-CBS.

trends indicate that as the level of congestion increases, integrating LRC with CBS can help increase the number of problems that are solved to optimality.

In Figure 3.8, we provide a summary of the runtimes and the optimality gaps with each algorithm on the 20% obstacle problem instances, and we analyze them. To save space, the analogous plots for the other % obstacle instances are provided in Appendix A, see Figures A.3 - A.5. Each graph in Figure 3.8 is split into 2 parts, we will start with the left half. The horizontal axis indicates time, and the vertical axis indicates the number of problem instances. A point $(p, q)$ on the curve corresponding to some algorithm in the left half should be interpreted as: $q$ out of the 25 instances were solved to optimality within $p$ seconds by the algorithm. We observe that, generally, the curve corresponding to LR-WDG (shown in yellow) lies above the curve corresponding to WDG-CBS (shown in blue), which indicates LR-WDG performs better than WDG-CBS. A similar trend is true between LR-WDG-G and WDG-G. In the initial 0-150 seconds or so, we observe that WDG-CBS and WDG-G lies higher than LR-WDG and LR-WDG-G. This behavior is consistent with the fact that, on easy problems, both WDG-CBS and WDG-G are able to quickly find the optimal solution, while LR-WDG and LR-WDG-G initially spends time executing Algorithm 2. Since in most cases, we find that LR-WDG and LR-WDG-G curves dominate WDG-CBS and WDG-G curves after 150 seconds, we can infer that Algorithm 2 terminates quickly despite its conservative termination criterion.

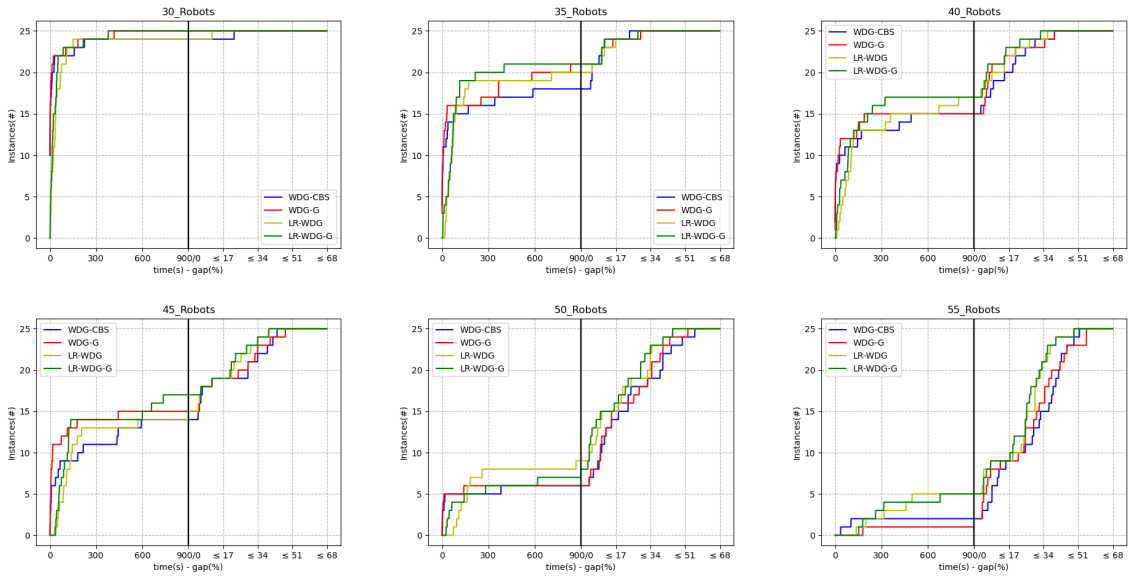



**Figure 3.8:** Runtime plots and gaps for 20% obstacle instances.

The right half of each plot in Figure 3.8, is meant to give a sense of the quality of the lower bound obtained with each algorithm. To measure the quality of the lower bound achieved by an algorithm for a given MAPF problem instance we designed the following measure. Let $lb$ be the lower bound to the optimal objective value obtained by the algorithm, and let $ub$ be the tightest known upper bound for the problem instance. The quality of the lower bound is computed as $\left(1 - \frac{lb - SP([\mathbf{N}])}{ub - SP([\mathbf{N}])}\right) \times 100$, where, recall from Section 3.3.6, $SP([\mathbf{N}])$ denotes the sum of the shortest start-end paths for all $\mathbf{N}$ robots in the problem. As a consequence of Eqn (3.41), notice that $lb \geq SP[\mathbf{N}]$, and so this measure is guaranteed to be non-negative for all four algorithms. A lower value of this measure indicates that the algorithm has made greater progress in closing the gap to the optimal solution, and so we refer to this measure as the gap. We came up with this non-standard measure called gap, because, in most of the experiments, $SP[\mathbf{N}]$ and the optimal objective value for a problem instance are very close, and so more standard measures such as the optimality-gap (computed as $100 \times \left(\frac{ub-lb}{lb}\right)$) turned out to be within $2-3\%$. If we use optimality-gap instead to compare the four algorithms, then the differences in the optimality-gap will be very small, owing to which, we may incorrectly conclude that the performance of all four algorithms are very similar. Hence we wanted a better measure to compare the 4 algorithms. When applying this measure on a given problem instance, we used the same $ub$ value in the computation of gap for all four algorithms. We obtained the $ub$ value by recording the best primal solution found for the problem instance across all 4 algorithms.

We next describe how to interpret the right hand side of the plots shown in Figure 3.8. A point $(p, q)$ on the curve corresponding to some algorithm in the right half of each plot should be interpreted as: on $q$ out of the 25 instances, the algorithm was able to achieve a gap to within $p\%$ at the time of termination. We see that on problems with fewer robots, all algorithms are able to close the gap to within 25%, and for the larger problems this number increases to 50%.

| Alg. Comp \ Robots | 30 | 35 | 40 | 45 | 50 | 55 |
|---|---|---|---|---|---|---|
| LR-WDG | 95.6, 18.4 | 100, 24.9 | 92.3, 19.8 | 100, 26.5 | 83.3, 29.1 | 100, 26.3 |
| LR-WDG-G | 95.8, 14.5 | 95.2, 24 | 100, 19.6 | 92.8, 14.6 | 100, 11.7 | 100, 99.8 |

**Table 3.1:** Comparison of number of search nodes expanded in the conflict tree before proving optimality on 20% obstacle instances. The middle row measures the performance of LR-WDG relative to WDG-CBS. We explain how to interpret the entries in the table with an example. Consider the entry in the middle row under column 40. The first entry, i.e., 92.3, means that among the 40 robot instances that are solved to optimality by both methods, LR-WDG expanded fewer search nodes than WDG-CBS on 92.3% of those instances. The second entry gives us a sense of the magnitude of the pruning capability of LR-WDG relative to WDG-CBS. Among those 40 robot instances solved to optimality by both methods, for instances where LR-WDG expanded fewer search tree nodes than WDG-CBS, we computed the number of search tree nodes expanded by LR-WDG as a percentage of the number of search tree nodes expanded by WDG-CBS, and the average of those percentages is shown as the second entry, i.e., 19.8. The bottom row compares the performance of LR-WDG-G relative to WDG-G.

Since the areas under the curves of LR-WDG and LR-WDG-G are larger than those under WDG-CBS and WDG-G, we can infer that LR-WDG and LR-WDG-G make greater progress than WDG-CBS and WDG-G lower bound wise.

We next compare the number of search tree nodes expanded by WDG-CBS and LR-WDG (resp. WDG-G and LR-WDG-G) on problem instances that are solved to optimality by both algorithms. The results for the 20% obstacle instances are summarized in Table 3.1. Refer to the caption of Figure 3.1 to interpret the entries in the table. We can generally observe from the entries in the table that on instances where both algorithms solve the problem to optimality, on a vast majority of those instances LR-WDG (resp. LR-WDG-G) expands far fewer number of search nodes than WDG-CBS (resp. WDG-G). We also observed that for many problem instances, LR-WDG and LR-WDG-G are both able to obtain the optimal solution in the root node of the search tree itself. This observation should not be surprising, since, in many cases, LR-WDG and LR-WDG-G are both able to close the optimality gap to 0 during the LRC phase itself (i.e., Algorithm 2). Analogous results comparing the number of search tree nodes for the other obstacle % instances are provided in Section A.5, see Tables A.1 - A.3. From those tables, we once again observe that LR-WDG (resp. LR-WDG-G) expands fewer search tree nodes than WDG-CBS (resp. WDG-G) for proving optimality.

### Results on Empty, Room, and Maze Instances

Analogous to Figure 3.7, Figures 3.9a - 3.9c provides a comparison of the 4 algorithms in terms of the number of problems solved to optimality. On the *Empty* layout, observe that the performance of WDG-CBS and LR-WDG begins to diverge only for problem instances with more than 70 robots. On the contrary, for the *Random* instances, we saw the performance of the two algorithms diverge on problems with fewer robots. When the start and goal location for a robot are chosen randomly on the layout, with high probability there will be many alternative routes with the same cost available for each robot on the layout. Consequently, very few opportunities arise where at least one robot may be forced to take a longer path or wait to

| Robots | Opt (%) |
|---|---|
| 50 | 100 |
| 60 | 99 |
| 70 | 88 |
| 75 | 80 |
| 80 | 68 |
| 90 | 44 |

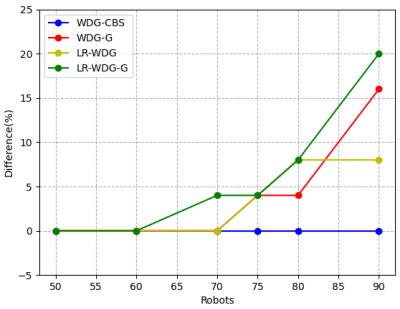



Figure 3.9(a) Empty

| Robots | Opt (%) |
|---|---|
| 15 | 100 |
| 20 | 100 |
| 25 | 76 |
| 30 | 32 |
| 35 | 12 |

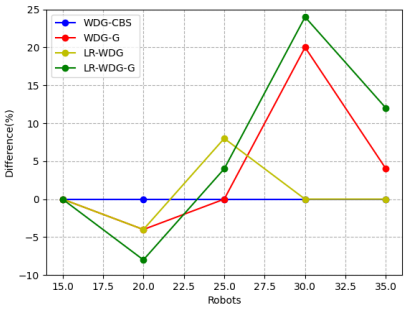



Figure 3.9(b) Room

| Robots | Opt (%) |
|---|---|
| 15 | 88 |
| 18 | 64 |
| 21 | 32 |
| 24 | 20 |

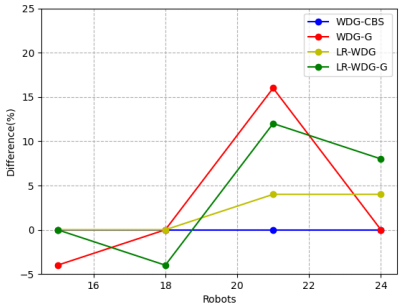



Figure 3.9(c) Maze

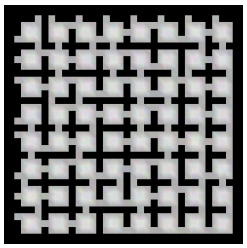


Figure 3.9(d) Room Layout

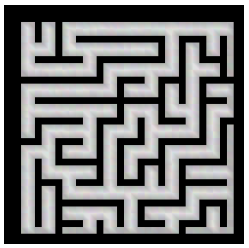


Figure 3.9(e) Maze Layout

avoid collisions. Only when the number of robots in the problem increases, the frequency of such opportunities occurring increases for our template based approach to be effective. This probably explains the results on the *Empty* layout.

On the *Room* and *Maze* layouts, observe that the performance of all 4 algorithms deteriorates very rapidly as the number of robots increases. On these layouts, low cost start-end paths for the robots tend to lie adjacent to the obstacles, see Figure 3.10a for an example of such paths on the *Maze* layout. Consequently, conflicts are expected to occur at locations besides obstacles. However, what makes these problems especially challenging is that, when planning a path for a robot using CBS, we may have to deal with multiple conflicts for a robot, and they may be spatio-temporally well separated from each other. For visualizing such conflicts, see the conflicts for robot 1 on the *Maze* layout in Figure 3.10a. Both the WDG node evaluation function, and our Template based Lagrangian node evaluation function are not very effective in these types of conflict scenarios. To understand why, for the example shown in Figure 3.10a, WDG treats the two conflicts as if they are independent of each other, while clearly the action of robot 1 around robot 2 determines when and where robot 1 will come in the vicinity of robot 3 later. Furthermore, since the two conflicts are spatio-temporally well separated, it is computationally prohibitive to build a single large template that captures both conflicts. Owing to this computational bottleneck, our template based approach generates cuts for each conflict separetely. Consequently, the algorithms considered in this work struggle to derive strong lower bounds for these types of scenarios. The runtime and gap plots are provided in in the Appendix, see Figures A.6 - A.8. The reader can observe that as the number of robots in the problem increase, the gap increases at a higher rate for the *Maze* and *Room* problem

instances, in comparison to the gap we reported for the other layout types. Analogous to Table 3.1, the tables for comparing the number of search tree nodes expanded for the *Empty*, *Maze* and *Room* instances are provided in Section A.5, see Tables A.4 - A.6. From those tables, we once again observe that LR-WDG (resp. LR-WDG-G) expands fewer search tree nodes than WDG-CBS (resp. WDG-G)

## 3.7 Future Work

In this section, we present three future directions for our work, centered around rst-neighborhoods.

### 3.7.1 Cuts from multiple rst-neighborhoods (Consensus cuts)

For a given rst-neighborhood $S$, observe that the size of the DD $\mathcal{D}(S)$ scales exponentially with $|\mathcal{R}(S)|$, which makes it computationally intractable to build DDs with a large $|\mathcal{R}(S)|$. Since the strength of our cut-generation method crucially relies on analyzing the paths of multiple robots within a rst-neighborhood, addressing this computational bottleneck in a tractable manner is beneficial. Instead of considering a large rst-neighborhood, one idea is to consider multiple smaller rst-neighborhoods, where each of those smaller rst-neighborhoods are parameterized with fewer robots, such that, the set of all robots considered in these smaller rst-neighborhoods together cover the robots in the larger rst-neighborhood. These smaller rst-neighborhoods can then be linked together using the familiar idea of consensus.

We illustrate the idea in the previous paragraph with the help of the problem shown in Figure 3.1. Instead of a single rst-neighborhood $S$ parameterized with all three robots as considered in Example 1, we can consider 2 smaller rst-neighborhoods $S_1$, $S_2$, such that, $S_1 \cup S_2 = S$, with $\mathcal{R}(S_1) = \{1, 2\}$, and $\mathcal{R}(S_2) = \{1, 3\}$. The other parameters $L(S_1), T(S_1)$ $L(S_2), T(S_2)$ can be chosen similar to how they were chosen in Example 1. Recall from Eqn (3.14), for generating valid inequalities for $\mathbf{P}$ using neighborhood $S$, we solved problems of the form shown in Eqn (3.14). Since we assumed that constructing the DD $\mathcal{D}(S)$ for $P(S)$ is expensive, we will instead maximize the objective of Eqn (3.14) over the constraints $x(S_1) \in P(S_1)$, and $x(S_2) \in P(S_2)$. The corresponding formulation is represented as the Primal consensus problem shown below, and we denote its optimal objective value by $opt\_val$.

$$
\begin{aligned}
& \boxed{\text{Primal consensus}} \\
opt\_val = \max_{x(S),y,z} \quad & d^{\mathsf{T}} x(S) \\
\text{s.t.} \quad & y \in P(S_1), \quad z \in P(S_2) \\
& y = x(S_1), \quad z = x(S_2) && (3.56) \\
& \sum_{a \in S \cap A_i(t)} x(a) \leq 1, \, \forall i \in \mathcal{R}(S_1) \cup \mathcal{R}(S_2), \forall t \in T(S_1) \cup T(S_2) && (3.57)
\end{aligned}
$$

The inequality that gets generated by solving the Primal consensus problem is given in Eqn (3.58). Equation (3.58) is valid for $\mathbf{P}$, since, $\text{Proj}_{x(S_1)}(\mathbf{P}) \subseteq P(S_1)$, $\text{Proj}_{x(S_2)}(\mathbf{P}) \subseteq P(S_2)$,

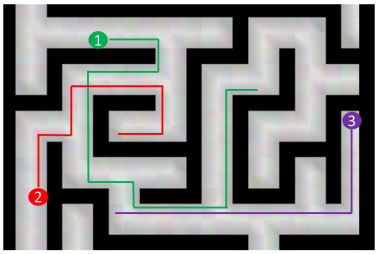



Figure 3.10(a)

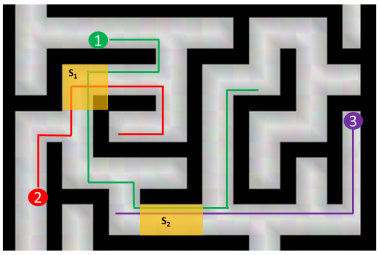



Figure 3.10(b)

In 3.10a, the shortest paths for the robots are indicated. The paths for robots 1 and 2 collide, and the paths for robots 1 and 3 collide. The yellow regions indicate the rst-neighborhoods $S_1$ and $S_2$.

**Figure 3.10**

and Eqn (3.57) is also valid for $\mathbf{P}$. We refer to the cuts of this type as consensus cuts.

$$d^{\mathsf{T}} x(S) \leq \lfloor opt\_val \rfloor \tag{3.58}$$

For computational ease, we can obtain $opt\_val$ in Eqn (3.58) by solving the dual to the Primal Consensus problem. We can obtain the dual by simply dualizing Eqns (3.56), (3.57). The resulting dual problem can be efficiently solved using projected first order gradient methods, we skip those details here. For the example from Figure 3.1, we performed an experiment where we applied Algorithm 2 and only included a single cut generated by solving the Primal consensus problem with two robot templates, i.e., $|\mathcal{R}(S_1)| = |\mathcal{R}(S_2)| = 2$. In that experiment, Algorithm 2 terminated with the optimal objective value, thus demonstrating a positive example for this approach.

### 3.7.2 Serial cuts

When the conflicts are spatio-temporally well separated, as shown in Figure 3.10a, we earlier mentioned in Section 3.6.2 that these situations are challenging to derive strong lower bounds. So we propose the following the idea. Consider the rst-neighborhoods $S_1, S_2$ for the conflicts in Figure 3.10b, where $S_1$ is parameterized by robots 1 and 2, and $S_2$ by robots 1 and 3. Instead of generating cuts for the 2 conflicts independently, i.e., by optimizing over the relaxation polytopes $P(S_1)$ and $P(S_2)$ separately, we can instead consider building a relaxation $P'$ to $\text{Proj}_{x(S_1 \cup S_2)}(\mathbf{P})$, and generate cuts by optimizing over $P'$. By doing so, we are jointly analyzing the paths of robots 1, 2 and 3, and this may help us generate better cuts.

For rst-neighborhoods $S_1, S_2$, assume that $T(S_1) = \{\mathbf{l_1}, \mathbf{l_1}+1, \ldots, \mathbf{u}_1\}$, and $T(S_2) = \{\mathbf{l_2}, \mathbf{l_2}+1, \ldots, \mathbf{u}_2\}$. For serial cuts, $S_1, S_2$ are chosen such that $\mathbf{u_1} < \mathbf{l_2}$. Define $S_1^{'} = S_1 \backslash \{\underset{i \in \mathcal{R}(S_1)}{\cup} A_i(\mathbf{u_1})\}$, and $S_2^{'} = S_2 \backslash \{\underset{i \in \mathcal{R}(S_2)}{\cup} A_i(\mathbf{l_2} - 1)\}$. We will construct $P'$ such that it is a relaxation for $\text{Proj}_{x(S_1^{'} \cup S_2^{'})}(\mathbf{P})$. Let $\mathcal{D}(S_1)$ (resp. $\mathcal{D}(S_2)$) be the DD representation of $P(S_1)$ (resp. $P(S_2)$). Instead of providing the polyhedral description for $P'$, we will directly explain the DD for $P'$, denote the DD by $\mathcal{D}_{P'}$. We will construct $\mathcal{D}_{P'}$ by connecting the states in the penultimate layer (i.e., corresponding to time $\mathbf{u_1}$) of $\mathcal{D}(S_1)$, to the states in the second layer (i.e., corresponding to $\mathbf{l_2}$) of $\mathcal{D}(S_2)$. Before we connect the states, we first remove the sink node and all arcs from the last layer of $\mathcal{D}(S_1)$. Likewise, we remove the source node and all

arcs from the first layer of $\mathcal{D}(S_2)$. A state $s_1$ in $\mathcal{D}(S_1)$ is connected to a state $s_2$ in $\mathcal{D}(S_2)$ by an arc with no labels, iff, $\forall i \in \mathcal{R}(S_1) \cap \mathcal{R}(S_2)$, one of the following condition holds:

- $s_1[i] \neq \mathbf{o}$, $s_2[i] \neq \mathbf{o}$, and there is a path from $s_1[i]$ to $s_2[i]$ in the flow graph $F_i(N_i, A_i)$.
- $s_1[i] \neq \mathbf{o}$, $s_2[i] = \mathbf{o}$, and $\exists w_i^{\mathbf{l_2}} \in N_i(\mathbf{l_2}) \backslash L_i^{\mathbf{l_2}}(S_2)$, such that, there is a path from $s_1[i]$ to $w_i^{\mathbf{l_2}}$ in $F_i(N_i, A_i)$.
- $s_1[i] = \mathbf{o}$, $s_2[i] \neq \mathbf{o}$, and $\exists w_i^{\mathbf{u_1}} \in N_i(\mathbf{u_1}) \backslash L_i^{\mathbf{u_1}}(S_1)$, such that, there is a path from $w_i^{\mathbf{u_1}}$ to $s_2[i]$ in $F_i(N_i, A_i)$.
- $s_1[i] = \mathbf{o}$, $s_2[i] = \mathbf{o}$, $\exists w_i^{\mathbf{u_1}} \in N_i(\mathbf{u_1}) \backslash L_i^{\mathbf{u_1}}(S_1)$, and $\exists v_i^{\mathbf{l_2}} \in N_i(\mathbf{l_2}) \backslash L_i^{\mathbf{l_2}}(S_2)$, such that, there is a path from $w_i^{\mathbf{u_1}}$ to $v_i^{\mathbf{l_2}}$ in $F_i(N_i, A_i)$.

It is not hard to see that every vertex of $\text{Proj}_{x(S_1' \cup S_2')}(\mathbf{P})$ is encoded by a $sr - sk$ path in $\mathcal{D}_{P'}$.

### 3.7.3 Branching on cuts

In previous sections, we used objective cuts and delay-and-long cuts entirely for the purpose of designing the node evaluation function via the Lagrangian. In this section, we show that the information present in each cut, can alternatively be used for branching on a conflict in CBS. For the vertex conflict shown in Figure 3.2, we derived Eqn (3.27). From this inequality, we can infer that in any feasible solution to the problem, the disjunction shown in Eqn (3.59) must be satisfied. So we can consider using the disjunction in Eqn (3.59) instead for branching on the conflict. The key feature that allows us to branch on objective cuts and delay-and-long cuts is that all the non-zero coefficients in the cut are 1, and all variables with non-zero coefficients in the cut correspond to the same time.

$$\begin{pmatrix} x(O_1^2, Q_1^3) = 0 & x(Z_1^2, Q_1^3) = 0 & x(Q_1^2, Q_1^3) = 0 \\ x(W_1^2, Q_1^3) = 0 & x(W_1^2, W_1^3) = 0 & x(Q_1^2, W_1^3) = 0 \\ x(W_1^2, X_1^3) = 0 & & \end{pmatrix} \vee \big(x(W_2^2, Q_2^3) = 0\big) \tag{3.59}$$

The observation made in the previous paragraph suggests a very simple modification to the branching rule of CBS. For branching on a conflict, we check whether an inequality that contains the conflict edges exists among $\hat{E}x \leq \hat{f}$ (i.e. cuts from Algorithm 2). If there are no such inequalities, we apply the usual branching rules that we used for the experiments in Section 3.6.2. Otherwise, amongst those inequalities containing the conflict edges, we pick the inequality which is violated the most by the conflict-containing shortest start-end paths stored in the search tree node, and then we branch on the conflict similar to how we did earlier for the example from Figure 3.2. In case of ties in picking the most violated inequality, we arbitrarily select an inequality that is not dominated by the other inequalities. To illustrate the benefit of branching on cuts, we tested this new branching rule for the highly congested set of scenarios represented in Figure 3.11. Averaged across the 16 instances, relative to WDG-CBS, we observed a 23% reduction in the number of search nodes expanded before finding the optimal solution with LR-WDG. In contrast, without our branching rule, LR-WDG expanded only 8% fewer nodes relative to WDG-CBS. These experimental results suggest that there may be potential with this approach, or some variant of it, for branching.

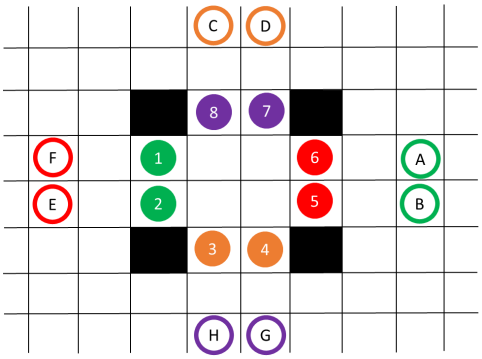



**Figure 3.11:** We show a partial view of a $10 \times 10$ 4-connected grid. The solid circles represent start locations for the 8 robots in the problem. Using these start locations we generated 16 problems by varying the destination locations as follows. Robots of the same color can only be assigned to a ring of the same color as its destination. So the destination of robots 1 and 2 can either be A or B, and so if A is assigned to robot 1, B must be assigned to robot 2. The makespan constraint for the problem, i.e, $\mathbf{T}$, was set to 16.

## 3.8 Related Work and Discussions

As our work derives valid inequalities from DDs, we begin this section by discussing how our work relates to prior methodologies on obtaining cuts from DDs. For solving 0-1 IPs using a cutting plane approach, the authors in Tjandraatmadja and van Hoeve [2019] construct a relaxed DD for the entire feasible region. For performing separation, they propose a method that returns a facet of the relaxation. In Davarnia and van Hoeve [2020], the authors proposed a computationally simpler procedure for obtaining separation inequalities using relaxed DDs, that are not necessarily facet-defining. Their procedure outputs the farthest separation inequality for a point lying outside the relaxation. In Mogali *et al.* [2020], we adopted the procedure from Davarnia and van Hoeve [2020] to generate cuts. The cuts proposed in this chapter, i.e., objective and delay-and-long inequalities, are fundamentally different from the cuts in [Tjandraatmadja and van Hoeve, 2019; Davarnia and van Hoeve, 2020]. Unlike Tjandraatmadja and van Hoeve [2019]; Davarnia and van Hoeve [2020], in our approach, we used the objective in the problem that is being optimized to specify the LHS of the inequality, and optimized over the DD only to compute the RHS of the inequality.

Using Lagrangians to obtain lower bounds is a well known and widely used idea in Mathematical programming [Geoffrion, 1974] and Constraint Programming [Benoist *et al.*, 2001], and the references are too many to list. Prior works [Bergman *et al.*, 2015; Castro *et al.*, 2020] have combined Lagrangians and DDs to generate lower bounds. Typically, they consider each combinatorial constraint in the problem separately, and for each constraint, they construct a relaxation to the feasible region defined by that constraint using a DD. The relaxations for the different constraints are then suitably combined. In contrast, we used DDs to construct relaxations to projections of the feasible region. We are not aware of any prior works that implements LRC, where the cuts are generated from DDs. Lagrangian decomposition has been applied for search based methods in the context of cost partitioning of abstraction heuristics [Pommerening *et al.*, 2019]. There is no concept of a cut or the construction of a relaxation to the projection of the feasible region, or the use of LRC in Pommerening *et al.* [2019]. The only commonality our paper shares with Pommerening *et al.* [2019] is in the use of Lagrangian decomposition.

We next move onto prior related woks on MAPF. In Gange *et al.* [2019], the authors present a Constraint Programming approach for MAPF with nogood learning. Their approach extracts nogoods during the search process, and uses those nogoods to avoid duplicated search

effort in the search tree. Each nogood can be viewed as specifying a set of path costs for some subset of robots in $\mathcal{R}$, that cannot occur in any feasible solution.

These nogoods are then utilized to guide the search in a core-guided strategy. Objective cuts can be viewed as being analogous to nogoods, since, recall from Example 2, these cuts are also able to provide similar certificates of infeasibility globally for the search space. However, the methodologies to obtain these inferences on the path costs are different in the two approaches.

In [Lam *et al.*, 2019; Lam and Le Bodic, 2020], the authors present a branch-cut-and-price (BCP) approach for solving the MAPF problem. As it is typical of BCP approaches, the IP formulation works in the space of start-end paths for the robots, and for every start-end path there is a decision variable to decide whether to include the path or not in the optimal solution. Many types of cuts were introduced, some of which cut off a fractional solution, and some of which are based on the conflicts present in the fractional solution. In Lam and Le Bodic [2020], six types of conflict classes were identified, and cuts specific to each class were proposed. These classes are based on characterizing the conflicting robot paths. For the sum of completion times objective, the current best results available for the problem instances from Stern *et al.* [2019] are due to Lam and Le Bodic [2020]. While the approach taken in our work is fundamentally different to this BCP approach, we speculate that the cuts presented in our work may be useful for the BCP approach, because our template based approach provides a generic way to analyze the paths of multiple robots concurrently for generating cuts.

CBS is amongst the state-of-the-art methods for MAPF. In Section 3.5.1, we described CBS and summarized the literature related to CBS. We next mention in passing some older approaches for solving the MAPF problem. One such group of methods are those that rely on solvers. Some representative publications are SAT [Surynek *et al.*, 2016], Answer Set Programming [Erdem *et al.*, 2013], and Integer Linear Programming [Yu and LaValle, 2013a]. These approaches have typically modeled the MAPF problem using simple time-expanded models, and rely entirely on the power of the solver. These approaches are only competitive for problems with small number of agents.

We next discuss the dependence of our cut generation procedure on the MAPF objective being minimized. To apply our cut generation scheme shown in Algorithm 3, our MAPF objective should allow us to do the following operation:

- Given a set of robots and a combined cost (objective) for the set of robots, we should be able to partition the combined cost individually between robots as shown in line 8 of Algorithm 3. Further, we should be able to translate the cost for each robot into arcs within a rst-neighborhood. What we mean by that is, in order to formulate the objective in the optimization problem (shown in Eqn (3.28)) for generating cuts, we should be able to identify all those arcs within the rst-neighborhood through which the shortest start-end path for the robot does not exceed the cost specified for the robot in the partition.

In Example 2, we provided a demonstration of the operation shown above with the MAPF objective described in Section 3.1. More generally, it is not hard to see that the operation described above can also be performed with objective functions where the cost of a solution can be decomposed into a sum of robot start-end path costs, where the cost of a start-end

path is given by the sum of cost of the arcs used in the path. Examples of other scheduling objectives that satisfy the above requirement include weighted sum of completion times and max tardiness. So our cut generation procedure can be adapted to other MAPF objectives.

In the remainder of this section, we will describe how our work may be useful to variants of the MAPF problem. In Atzmon *et al.* [2018], the authors introduced a new notion of robustness, called k-robustness, which we state verbatim: *A plan that is robust to k delays per agent during plan execution, i.e., each agent may be delayed up to k times during plan execution and the plan would still be safe (no collisions).* The authors demonstrate how existing algorithms for MAPF, such as A* and CBS, can be adapted to solve the k-robust variant of the MAPF problem. In Li *et al.* [2019b], the authors consider the MAPF variant where some agents need to occupy multiple locations on the grid simultaneously. They presented a generalized version of CBS, called Multi-Constraint CBS, which adds multiple constraints (instead of one constraint) in the constraint generation step, while generating the child nodes. We can adapt the LRC lower bounding scheme described in Algorithm 2 to both these variants of the MAPF problem. We simply have to add more constraints to the relaxation polytope $P(S)$ described in Section 3.3.4, to generate tighter inequalities. The details are beyond the scope of this thesis.

## 3.9 Summary

In this chapter, we presented a Lagrangian Relax-and-Cut scheme for generating tight lower bounds for the MAPF problem. Cuts were generated from a novel cut generation scheme. At a high level, tight polyhedral relaxations to projections of the feasible region are computed in our cut generation scheme. We developed a query procedure, that probes the existence of feasible solutions satisfying certain criterion. We attempt to answer those queries by only analyzing the relaxation polytopes, and whenever the answer to a query is found to be negative, we obtain a cut. Decision diagrams were used to compactly represent the relaxation, allowing us to efficiently obtain answers to our queries. We incorporated the lower bound from the LRC scheme into a state-of-the-art variant of conflict-based search as a node evaluation function. Our experimental results demonstrate that incorporating our lower bounding scheme improves the performance of CBS by reducing the optimality gap, and also increases the number of problems that are solved optimally.

## 3.10 Summary of contributions

- A new polyhedral cutting plane approach for the MAPF problem. Our approach is eclectic in the sense that, it combines LRC, projections and DDs. Cuts from relaxations to projections are used for tightening the linear description.
- The projections considered in this work are specified using *Templates*. The use of templates enabled us to capture the facial structure of the projection polytope, which allowed us to reuse this structural information for obtaining other projections. The idea

of using templates for specifying projections may be more broadly applicable to other combinatorial problems.

- The cuts from templates presented in this chapter are somewhat novel in nature. A query procedure that accounts for the objective being minimized is used to specify the LHS of the cut, and an optimization problem over a relaxation to the projection of the feasible region is solved to compute the RHS of the cut. The optimization problem is solved efficiently with the help of a DD.
- We demonstrated how the Lagrangian information can be used to strengthen the WDG node evaluation function.

# Part II

# Chapter 4

# Blocking Job Shop problem

## 4.1 Introduction

The classical job shop (JS) problem [Conway *et al.*, 2003] is a difficult combinatorial optimization problem that has been widely studied. A set of jobs have to be executed on a set of machines, where each job is defined as a sequence of operations, and each operation specifies the machine on which it has to be processed. A machine can process at most one job operation at a time, a processing time is defined for each operation and, once started, an operation cannot be interrupted (non-preemptive). The goal is to find a particular sequencing of operations on each of the machines and a schedule (i.e., start time for each operation) for the sequencing that minimizes an objective function. A number of possible objectives have been investigated for the classical job shop problem. A prominent one is makespan minimization, where the time it takes to completely service all job operations has to be minimized.

One of the assumptions in the classical JS problem is that as soon as an operation finishes executing on a machine, the machine becomes available. Basically, this means that there exists a buffer to store the job (product) between successive operations, and that this buffer has sufficient capacity to store all partially serviced jobs. In most real-world settings, this is not a realistic expectation as there may be physical limits on the capacity of the buffer. Different versions of the job shop problem have been defined in the literature, to include more realistic features in the problem formulation. In the blocking job shop (BJS) [Mascis and Pacciarelli, 2002], when a job operation finishes executing on a machine, it remains on this machine (blocks the machine) until the downstream machine required to continue processing the job becomes available. BJS is a useful model for flexible manufacturing systems with limited storage space between operations and limited capacity to move the materials from one machine to the next one [Hall and Sriskandarajah, 1996]. Complexity results published in Hall and Sriskandarajah [1996] show that makespan minimization of the BJS problem is difficult to solve; in particular, the BJS problem with only two machines is strongly NP-hard.

There are two versions of blocking researched in the literature: blocking with swap allowed and blocking without swap. These two versions differ in the way a “deadlock situation" is handled, where a deadlock situation is a state of the system where there exists a set of jobs such that each job in the set is waiting for a machine that is blocked by some other job from

the same set. When swaps are allowed, once every job in the deadlocked set is serviced by the machine they currently occupy, all jobs in the deadlocked set are simultaneously moved to their respective downstream machines. In the no swap allowed case, the simultaneous movement of jobs in the deadlocked set is not allowed, so any solution for which a deadlock state is present is infeasible. The blocking job shop problem has found many applications in areas such as train scheduling [D'ariano *et al.*, 2007; Lange and Werner, 2018; Liu and Kozan, 2009; Strotmann, 2008], production of steel, transportation and material handling in manufacturing [Lange and Werner, 2019; Mati and Xie, 2011; Poppenborg *et al.*, 2012], automated warehouse [Klinkert, 2001; Mati *et al.*, 2001a], a store-and-forward 1-network [Mascis and Pacciarelli, 2002], and cyclic scheduling [Brucker and Kampmeyer, 2008]. Blocking constraints have also appeared in applications such as truck scheduling at tank terminals [Van den Bossche *et al.*, 2020], some variants of railway scheduling [Meng and Zhou, 2014; Törnquist and Persson, 2007], and aircraft scheduling [Sama *et al.*, 2017].

Local search methods for JS problems typically employ a metaheuristic procedure to enhance the search performance. In particular, tabu search approaches are currently among the state of the art [Dabah *et al.*, 2017; Nowicki and Smutnicki, 2005] for both JS and BJS. In this work, we establish a number of theoretical results and use them to implement an efficient tabu search approach for the BJS problem. Typical neighborhoods used for solving job-shop problems are based on the critical path of the current solution; our approach employs the *N4* and *N5* neighborhoods [Błażewicz *et al.*, 1996] originally proposed for JS problems, adapted for BJS problems with some modifications. The neighbors generated for a solution are not always guaranteed to be feasible – this does not seem to critically affect the practical performance of local search methods for JS problems. However, for BJS problems this is not the case and so various procedures are routinely used to restore feasibility for infeasible neighbors, at an additional computation cost. In this work, we propose new theoretical results that can be used (1) to quickly determine which neighbors are infeasible, and (2) to provide computationally efficient procedures to restore feasibility. We also present empirical results that show how our work advances the current state of the art. Although this work focuses on the BJS problem, we believe our work may be of interest to a broader audience interested in local search procedures for problems with blocking constraints. Finally, parts of this chapter previously appeared in Mogali *et al.* [2021a].

## 4.2 Literature review

Mascis and Pacciarelli [2002] present a detailed study of the BJS problem and its variants; they introduce the Alternative Graph model (refer Section 2.2.2 in Chapter 2) which is an extension of the Disjunctive Graph representation widely used for the JS problem, and present complexity results for the BJS problem with no swap. They also show that popular heuristics for the JS problem, like the "Shifting Bottleneck Heuristic" [Adams *et al.*, 1988] and dispatch based rules, often fail to produce a feasible solution when applied to BJS problems, and describe four greedy constructive heuristics for obtaining a feasible solution in these situations. Meloni *et al.* [2004] present a constructive procedure that combines the heuristics proposed in Mascis and Pacciarelli [2002] with a rollout scheme. Pranzo and Pacciarelli [2016] present an

Iterative Greedy approach, embedded within a metaheuristic (Simulated Annealing or Random Walk). During the destruction phase, a randomly-chosen part of a solution is destroyed; in the constructive phase, procedures introduced in [Mascis and Pacciarelli, 2002] are used to complete the solution. Oddi *et al.* [2012] present an Iterative Flattening Search approach for BJS problem with swap, which similarly to Pranzo and Pacciarelli [2016], uses a destruction and construction phase at each iteration. The destruction phase selects a portion of the solution to destroy either randomly or based on temporal slack, while the constructive phase uses a precedence constraint posting procedure. Empirical results from Oddi *et al.* [2012] show that the Iterative Flattening Search outperforms the Self-Adapting Large Neighborhood Search implemented in IBM's ILOG CP Optimizer.

The constructive procedures described in Mascis and Pacciarelli [2002]; Meloni *et al.* [2004]; Pranzo and Pacciarelli [2016]; Oddi *et al.* [2012] do not guarantee that a feasible solution is generated at each iteration. Moreover, these constructive heuristics require a destruction phase, where the part of the solution that is destroyed is randomly selected. There is a very strong randomness component in these procedures, and the experimental results for this random exploration of the search space generally have not been competitive when compared with some later papers (discussed below) addressing the BJS problem. In addition, some of these methods require temporal propagation, which can be quite expensive and almost prohibitive for large problems instances.

Another type of constructive approach can be broadly classified as a job insertion based procedure. One of the earliest examples is Mati *et al.* [2001b], where a geometry based job insertion procedure embedded within tabu search is described. A method to solve the two job problem optimally, and a heuristic to sequentially expand a partial solution by introducing one job at a time are proposed. All jobs present in the partial solution are combined to form a single job, and the new job is inserted using the optimal two job insertion procedure.

A larger number of studies [Bürgy, 2017; Dabah *et al.*, 2017; Gröflin and Klinkert, 2009; Gröflin *et al.*, 2011], use Job Insertion methods somewhat differently to the approach taken in [Mati *et al.*, 2001b]; these are perhaps among the most successful methods so far for the BJS problem. All these studies use a JS local search neighborhood (such as *N1, NA, NB, N2, N4, N5* [Błażewicz *et al.*, 1996]), to generate neighbors of the current solution. Unfortunately, the neighbors produced by these neighborhood moves are not always feasible. A job insertion based feasibility recovery (JIFR) mechanism is used to convert an infeasible solution to a feasible one. Basically, JIFR heuristics operate by selecting one of the jobs in the infeasible solution and altering the position where its operations are executed on each machine.

The theory around job insertion based feasibility recovery has been a natural outgrowth from studies of the job insertion polytope. The job insertion polytope is a polyhedral characterization of the space of feasible insertions of a job, given a feasible solution for the remaining jobs. The job insertion polytope was first published in the context of JS problems by Kis and Hertz [Kis and Hertz, 2003], showing that the job insertion polytope is an integral polytope. On the problem of optimally reinserting a job (with makespan as the objective to minimize), they present a novel lower bounding procedure. In Gröflin and Klinkert [2007], using a different parameterization of the unknowns in the problem, an alternative characterization of the job insertion polytope is provided; this procedure is applicable to many variants of the JS problem.

Building on the theoretical work presented in [Gröflin and Klinkert, 2007], Gröflin and

Klinkert [2009] present a JIFR procedure referred to as “closure". A salient feature of closure is that it performs the fewest number of pairwise swaps of operations on machines to restore feasibility, where every swap action involves an operation belonging to the job that is being inserted. Closure has subsequently been applied to problems with objectives other than makespan for several variants of JS problems, for example, closure embedded in a tabu search framework has been used in [Bürgy, 2017; Gröflin *et al.*, 2011].

Dabah *et al.* [2017] also uses a JIFR procedure, where AMCC (Avoid Most Critical Completion time) ranking is the criterion guiding the job insertion procedure. AMCC [Mascis and Pacciarelli, 2002] is a greedy heuristic that avoids a sequencing decision that causes the largest increase in makespan in the partial solution. The empirical results in Dabah *et al.* [2017] suggest that job insertion with AMCC leads to solutions with lower makespan than those obtained using closure. Dabah et al. extend their algorithm to be suitable for parallelism [Dabah *et al.*, 2019]; this approach has produced many of the best known results for problem instances in the BJS problem datasets.

### 4.2.1 Current challenges

Our approach, similar to most successful algorithms published for the BJS problem, is a local search heuristic that uses a JS neighborhood with a JIFR procedure, embedded in a tabu search metaheuristic. The practical performance of such an algorithm is affected by two factors: (i) The properties of the chosen neighborhood, and (ii) the computational efficiency of the procedures used to enumerate and evaluate all neighbors at each step of the search. For the first factor, ideally one would prefer to have a neighborhood that specifies only few neighbors, and yet possesses the opt-connectivity property. A neighborhood is opt-connected if given any arbitrary start solution, there exists a sequence of solutions terminating with an optimal solution, such that any solution in the sequence is a neighbor of its predecessor in the sequence. Currently, very little is known about the properties of commonly used BJS neighborhoods in general, and their opt-connectivity in particular. For the second factor, the propensity for generating infeasible neighbors when expanding a BJS neighborhood and the consequent need to recover feasible solutions adds considerable computational expense to the basic search cycles. This work primarily addresses the computational challenges associated to (ii).

### 4.2.2 Our work and contributions

In this work, we adapt the *N4* neighborhood for the BJS problem, by combining the *N4* JS neighborhood Błażewicz *et al.* [1996] with JIFR. Briefly, the *N4* neighborhood partitions the critical path into contiguous blocks of operations that require the same machine, and generates each neighbor by relocating an operation in the critical block to either the beginning or the end of the block, thereby reordering operations on the machine associated with the critical block to generate each neighbor. The precise definition of the *N4* neighborhood adapted for the BJS problem is provided in Section 4.4. We chose to adapt *N4* because it is a fairly large neighborhood, subsuming most popular JS neighborhoods. Neighbors within the *N4* neighborhood of a solution can be classified into two types, feasible and infeasible. For infeasible neighbors, JIFR procedures are applied to convert them into feasible solutions. The techniques proposed

in this work assume the BJS no-swap version. Some of these techniques can be modified to make them applicable to the other BJS variants, see discussions in Section 4.11.

The main contributions of this work include the following:

1. We propose an algorithm that enumerates all the feasible *N4* neighbors in complexity that, in practice, scales linearly with the number of operations, i.e., $\mathbf{MJ}$, where $\mathrm{M}, \mathbf{J}$ are the number of machines and jobs respectively.

2. We describe an efficient algorithm for computing the makespan of all feasible *N4* neighbors.

3. We provide an efficient computational procedure for obtaining the polyhedral description of the Job Insertion Polytope **JIP** (in the context of BJS). Many papers, Bürgy [2017]; Gröflin and Klinkert [2009]; Gröflin *et al.* [2011], have previously exploited the structure of the (**JIP**) for developing Job insertion algorithms (see, e.g., the concept of closure [Gröflin and Klinkert, 2009]) for complex variants of JS problems. However, the computational aspect of efficiently implementing such schemes for feasibility recovery has not received any attention to the best of our knowledge, and nor has the complexity of existing procedures been analyzed. We describe the **JIP** with at most $\mathrm{M}^2\mathbf{J} + 2\mathbf{MJ}$ linear constraints by exploiting structure in the BJS problem, and provide a method for generating those constraints in $\mathcal{O}(\mathrm{M}^2\mathbf{J})$ complexity. Our contribution helps in implementing job insertion procedures efficiently by making use of the procedure for obtaining the **JIP**.

4. We introduce a new feasibility recovery procedure that can be used to expand the set of neighbors that are generated, in the hope that the expanded neighborhood leads the search to regions in the search space that are currently unreachable with existing JIFR methods. The size of the expanded neighborhood is controllable for computational tractability.

5. In Section 4.9, we report the performance of local search with *N4* neighborhood and tabu search as our metaheuristic for BJS no-swap instances that incorporates all the above-mentioned procedures. An experimental analysis was conducted using instances from the Lawrence [Lawrence, 1984], and Taillard [Taillard, 1993] benchmarks. For the purposes of comparison, previously published results only exist for the Lawrence instances. On those instances, we are able to obtain new best results on 28 out of the 40 instances. The average best solution we obtained across multiple runs after just 10 minutes of simulation time is better than the previous best known results on 21 out of the 40 instances. We also present results on BJS problem instances as large as $100 \times 20$ (number of jobs $\times$ the number of machines) in Taillard's benchmark dataset for the first time in BJS literature. Prior to the work reported here, the largest $\mathbf{J} \times \mathrm{M}$ BJS instances for which results have been published was $30 \times 10$ and $15 \times 15$.

6. For cases where $\mathbf{J} > \mathrm{M}$, we present a structural result that is satisfied by any feasible schedule for a BJS instance. The new result reveals an important structural difference in schedules for pure blocking and non-blocking problems. As real world problems typically tend to have $\mathbf{J} \geq \mathrm{M}$, discussing these structural characterizations is insightful. We discuss the utility of such results in improving the implementation of job insertion based local search procedures.

Throughout this chapter we will assume that all jobs require service from every machine exactly once, a common assumption made in literature. Note however, that all the local search procedures presented in this work can be easily adapted to cases where jobs require services from different subsets of machines or jobs may require a specific machine more than once.

## 4.3 Blocking Job Shop problem

In this section, we introduce the notation used in the rest of the chapter and formally describe the BJS problem. Our notation is based on the notation introduced in Balas and Vazacopoulos [1998] for the JS problem. For a BJS problem with $\mathbf{M}$ machines and $\mathbf{J}$ jobs, let $O = \{o_i | i = 1, \ldots, \mathbf{J} \cdot \mathbf{M}\}$, represent the set of all operations. For any operation $o \in O$, let $\alpha(o)$ denote the job predecessor of $o$ and $\gamma(o)$ denote the job successor of $o$. The order in which machines are accessed by each job is sometimes referred to as the technological order of the job in literature. If $o$ is the last operation in the technological order of the job, then $\gamma(o)$ should be interpreted as a dummy operation. Let $\bar{O}$ denote the set of all operations, including the dummy operations at the end of each job. The processing duration of $o$, is denoted by $p(o)$. Let $M : O \to [1, \mathbf{M}]$ output the machine required by operation $o$, and let $J : \bar{O} \to [1, \mathbf{J}]$ output the job to which $o$ belongs to. For $o \in \bar{O}$, if we let $t_o$ denote the start time of $o$, then the makespan minimization formulation for the BJS problem is shown below.

$$\begin{aligned} \min_{T, t_o \in \mathbb{R}_+ \, \forall o \in \bar{O}} \quad & T \\ \text{subject to} \quad & t_{\gamma(o)} \geq t_o + p(o) \;\forall o \in O, \quad t_o \leq T \;\forall o \in \bar{O}, \\ & \big(t_u \geq t_{\gamma(v)}\big) \vee \big(t_v \geq t_{\gamma(u)}\big) \;\forall u, v \in O \text{ such that } M(u) = M(v) \text{ and } u \neq v \end{aligned}$$

A schedule generated for the BJS problem using the disjunctive formulation shown above may contain swaps, refer to Section 4.1 for the definition of a swap. Unfortunately, a schedule with swaps is infeasible for the BJS no-swap problem. To best model the no-swap constraint we will instead work with a graphical representation of the BJS problem called the alternative graph representation [Mascis and Pacciarelli, 2002].

The alternative graph $G = (\bar{O}, F, \mathcal{A})$ for the BJS problem is a triple, where $\bar{O}$ is the set of nodes in $G$. $F$ is the set of fixed directed arcs, i.e., $F = \{(o, \gamma(o)) | o \in O\}$. The weight of each arc $(o, \gamma(o)) \in F$ is set to $p(o)$. $\mathcal{A}$ is a set of directed alternative arc pairs. There is a 1:1 correspondence between pairs of operations in $O$ that require the same machine and the pairs of alternative arcs in $\mathcal{A}$. For example, suppose $o_i, o_j \in O$ with $M(o_i) = M(o_j)$, then $\mathcal{A}$ contains an arc pair $((\gamma(o_i), o_j), (\gamma(o_j), o_i))$, modeling the fact that either $o_i$ is before $o_j$, or $o_j$ is before $o_i$. Alternative arcs represent the blocking constraints and have 0 weight. Figure 4.1 shows a simple BJS problem expressed as an alternative graph.

A selection $S$ is a subset of the alternative arcs in $\mathcal{A}$, containing at most one arc from each arc pair in $\mathcal{A}$. Given a selection $S$ and alternative arc pair $e = (a_1, a_2) \in \mathcal{A}$, if $S$ does not contain either of the arcs in $e$, then following the notation in Mascis and Pacciarelli [2002] we say that $e$ is *unselected* in $S$. If $a_1$ is present in $S$, then we say $a_1$ is *selected* in $S$. Likewise, we say that $a_1$ is *forbidden* in $S$ if $a_2$ is present in $S$, i.e., $a_2$ is *selected* in $S$. A selection

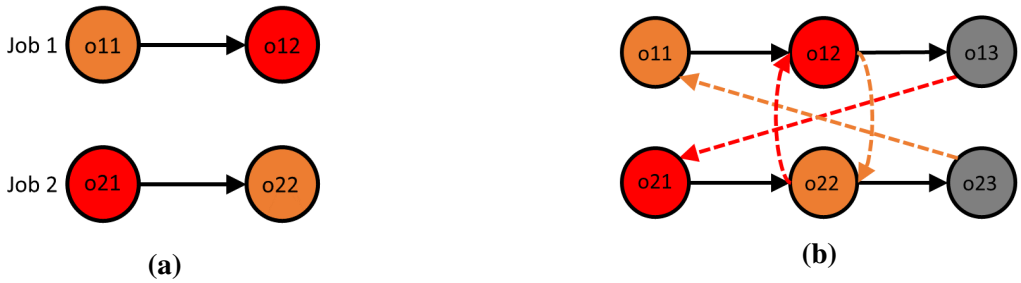



**Figure 4.1:** In Figure 4.1a, we show a problem with 2 jobs and 2 machines. The operations corresponding to Job-1 (resp. Job-2) are $o11, o12$ (resp. $o21, o22$) and so $o12 = \gamma(o11)$ (resp. $o22 = \gamma(o21)$). In this example $M(o11) = M(o22)$ and $M(o12) = M(o21)$. In Figure 4.1b we show the alternative graph representation of the problem. Nodes $o13, o23$ are dummy nodes. The unbroken arcs in Figure 4.1b (shown in black) are the arcs in $F$. Since $M(o11) = M(o22)$, corresponding to the pair $o11, o22$, we have alternative arcs $(o12, o22)$ and $(o23, o11)$ which are shown as dashed orange arcs. Analogously, since $M(o12) = M(o21)$, we have alternative arcs $(o13, o21)$ and $(o22, o12)$, both shown in red. So in this example $\mathcal{A} = \{((o12, o22), (o23, o11)), ((o13, o21), (o22, o12))\}$. The selection corresponding to the solution where $o11$ is serviced earlier than $o22$ and $o12$ is serviced earlier than $o21$ is given by the set $\{(o12, o22), (o13, o21)\}$. Alternatively, if $o22$ is chosen to be the machine successor of $o11$, and $o12$ as the machine successor of $o21$, then such a selection creates a 0 length cycle involving nodes $o12, o22$. After $o11$ and $o21$ are serviced, the only way to continue servicing the jobs is to swap the two jobs on the machines. For the BJS no-swap problem, such a resolution is not permitted and represents a deadlock [Mascis and Pacciarelli, 2002].

$S$ is *complete* if exactly one arc from each alternative arc pair is *selected* in $S$, and *partial* otherwise.

Given a selection $S$, $G(\bar{O}, F \cup S)$ is the graph containing only arcs $F \cup S$. Given $G(\bar{O}, F \cup S)$, temporal relations between nodes can be inferred. If $(u, v) \in F \cup S$ and $l(u, v)$ is the length of the arc $(u, v)$, then the relation $t_v \geq t_u + l(u, v)$ is true, and hence one can compute a complete (partial) schedule iff $G(\bar{O}, F \cup S)$ does not contain positive length cycles. A selection $S$ is said to be consistent for the BJS no-swap problem iff $G(\bar{O}, F \cup S)$ does not contain any directed cycles, including 0 length cycles. 0 length cycles are prohibited to prevent deadlock situations (refer Figure 4.1 for an example). We refer to a complete consistent selection as a *feasible* selection.

Given a feasible selection $S$, a well known technique to derive a schedule using $S$ makes use of the longest path computations on the graph $G(\bar{O}, F \cup S)$. Following standard practice to facilitate this computation, we introduce dummy nodes $\theta, \Lambda$, and introduce 0 length arcs from $\theta$ to the first operation (node) in the technological order of each job, and 0 length arcs from the dummy node (operation) of each job to $\Lambda$. Denote the graph with new nodes and arcs included by $G(\tilde{O}, F \cup S)$, where $\tilde{O} = \bar{O} \cup \theta \cup \Lambda$. The notation $L_S(o_i, o_j)$ is used to denote the cost of the longest path from node $o_i$ to node $o_j$ in $G(\tilde{O}, F \cup S)$. If a path does not exist from $o_i$ to $o_j$ in $G(\tilde{O}, F \cup S)$, then we will assume $L_S(o_i, o_j) = -\infty$. A schedule from selection $S$ is obtained by setting the start time of each node $o$ to $L_S(\theta, o)$. The makespan of the schedule associated to selection $S$ is given by $L_S(\theta, \Lambda)$. For brevity, in the rest of the chapter we will abbreviate $G(\tilde{O}, F \cup S)$ as $G(F \cup S)$.

Given a feasible selection $S$ and operation $o$, let $\beta_S(o)$ denote the machine predecessor of $o$ in $S$, i.e., the operation preceding $o$ on $M(o)$. Let $\delta_S(o)$ denote the machine successor of $o$ in $S$. We define a minimal representation of a complete consistent selection $S$ as the set of directed arcs $\{(\gamma(o), \delta_S(o)) \,|o \in O\}$. Denoting the minimal representation of $S$ by $S_{min}$, observe that although $S_{min}$ is a selection containing fewer arcs than $S$, the longest paths between any pair of nodes in $G(F \cup S)$ is preserved in $G(F \cup S_{min})$ also. Often in this chapter, we present algorithms that work by directly manipulating a minimal representation instead of its complete selection counterpart.

For a consistent selection $S$, if $A$ is a subset of $S$, then the longest path cost from $o_i$ to $o_j$ in $G(F \cup S)$ that does not use any arc in $A$ is denoted by $L_{S\setminus A}(o_i, o_j)$. We denote $R_S(o) \subseteq \tilde{O}$ as the set of nodes reachable by a directed path originating from $o \in \tilde{O}$ in $G(F \cup S)$. Likewise, $\bar{R}_S(o) \subseteq \tilde{O}$ denotes the set of nodes from which $o$ is reachable by some directed path in $G(F \cup S)$. $P_S(o)$ is the set of all directed paths originating from $o \in \tilde{O}$ in $G(F \cup S)$, and $\bar{P}_S(o)$ is the set of all directed paths in $G(F \cup S)$ that terminate at $o$.

Techniques developed in this work extensively use the concept of topological ordering of nodes in a graph. For a consistent selection $S$, a topological ordering of nodes in $G(F \cup S)$ is a linear ordering of the nodes, such that, if $(u, v)$ is an arc in $G(F \cup S)$, then $u$ is before $v$ in the ordering. If $T_S$ is some valid topological order for nodes in $G(F \cup S)$, $T_S[i]$ denotes the node at position $i$ in $T_S$, and $T_S^{-1}(o)$ denotes the position of node $o$ in $T_S$.

## 4.4 Critical blocks and *N4* neighborhood

Given a feasible selection $S$ for the BJS instance, a critical path $cp(S)$ refers to any one of the longest paths from $\theta$ to $\Lambda$ in $G(F \cup S)$. Critical blocks, first introduced in the context of JS problems, can be intuitively thought of as a maximal sequence of operations on $cp(S)$ that are executed on the same machine [Nowicki and Smutnicki, 1996]. Critical blocks play a crucial role in the definition of local search neighborhoods for JS problems and its variants. In the context of the BJS problem, the simple, intuitive definition for critical blocks needs to be amended to handle complexities introduced due to blocking.

In BJS, any two consecutive operations $u_i$, $u_j$ on the same machine are connected indirectly in $G(F \cup S)$, via a job arc from $u_i$ to its successor $\gamma(u_i)$, and then an alternative arc from $\gamma(u_i)$ to $u_j$, i.e., the job successor of $u_i$ to the machine successor of $u_i$. Thus, for any consecutive sequence of operations $(u_1, \ldots, u_n)$ $(n \geq 2)$ on the same machine $m$ that lie on $cp(S)$, there will be a sub-path in $cp(S)$ of the form $u_1, \gamma(u_1), u_2, \gamma(u_2), \cdots\cdots u_n$. Analogous to the JS definition of critical block, we define each such maximal sequence $(u_1, \ldots, u_n)$ as a type-1 critical block.

After extracting all type-1 critical blocks from $cp(S)$, there can still be critical blocks in $cp(S)$ that have not been accounted for. In particular, there may still be additional alternative arcs on $cp(S)$, corresponding to situations where there are two consecutive operations on the same machine but only the second operation of the pair is on $cp(S)$. For every such additional alternative arc $(\gamma(u_1), u_2)$ on $cp(S)$, we define $(u_1, u_2)$ as a type-2 critical block.

**Example 7.** *Consider the example in Figure 4.2. Looking at the critical path in Figure 4.2b,*

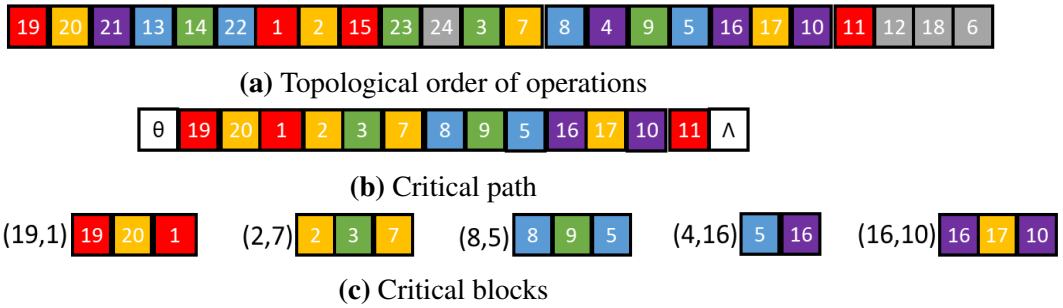
*observe that (19, 1) is a maximal sequence of operations for the red machine, and the corresponding sub-path on the critical path is 19, 20, 1. The corresponding type-1 critical block is 19 and 1, denoted by (19, 1) in Figure 4.2c. Similarly, (2,7), (8,5) and (16,10) are also type-1 critical blocks. The alternative arc (5,16) is not included in any of the sub-paths corresponding to the type-1 critical blocks, so it defines the type-2 critical block (4,16). Note that all the other arcs not included in type-1 critical blocks are job arcs (from operation 1 to operation 2, from 7 to 8, and from 10 to 11). Also, a type-1 critical block must contain at least two operations, so 11 by itself is not considered as a critical block.*

Using these critical block definitions, we can now describe the *N4* neighborhood for BJS. The moves that define the *N4* neighborhood involve selecting a single operation in a critical block $cB$, and moving the operation either before or after $cB$ (referred to respectively as either a forward and backward move). More precisely, assume we are given a critical block $cB = (u_1, \ldots, u_n)$. Then:

- A forward *N4* move selects an operation $w \in \{u_1, \ldots, u_{n-1}\}$ ($n \geq 2$) and makes $w$ the machine successor of $u_n$ (i.e., $w$ is shifted to the end of $cB$). A forward move is applicable to both the types of critical blocks.
- A backward move *N4*, alternatively, is applicable only to a type-1 critical block. There are two cases:
  - If the incoming arc to $u_1$ on $cp(S)$ is a job arc (i.e., arc in $F$), then an operation $w \in \{u_2, \ldots, u_n\}$ ($n \geq 2$) is selected and made the machine predecessor of $u_1$ (i.e., $w$ is shifted to the beginning of $cB$, as is the case with backward moves in the JS problem).
  - If the incoming arc to $u_1$ on $cp(S)$ is an alternative arc (i.e., arc in $S$), then an operation $w \in \{u_2, \ldots, u_n\}$ ($n \geq 2$) is made the machine predecessor of $\beta_S(u_1)$ This is because moving an operation to the block's beginning in this case will not result in a neighbor with lower makespan than that of $S$. The blocking time on the machine cannot be reduced by such a move, since the blocking time is independent of which operation is the first one in the block. This fact was also recognized in Heitmann [2007] (see Theorem 6.2).

The set of all solutions that can be generated by applying these forward and backward moves to all critical blocks of $S$ is defined as the *N4* neighborhood of $S$. The general motivating principle, as just alluded to, is to generate neighbors that move one or more operations on $cp(S)$, since only changes to the critical path can result in a solution with a smaller makespan than $S$. In Section 4.5 we provide an efficient method for determining which *N4* neighbors are feasible. In Section 4.6, we discuss JIFR procedures for transforming infeasible *N4* moves into feasible solutions.

**Example 8.** *To illustrate N4 neighborhood generation, recall the example shown in Figure 4.2. For the type-1 critical block* $(19, 1)$*, both the forward and the backward move procedures create the same neighbor where 19 becomes the machine successor of 1. The next two critical blocks,* $(2, 7)$ *and* $(8, 5)$*, are also type-1 critical blocks and yield analogous single*




**(a)** Topological order of operations

**(b)** Critical path

**(c)** Critical blocks

**Figure 4.2:** Consider a $4 \times 5$ BJS problem where operations belonging to job $i$ are $\{6(i-1)+k\}_{k=1}^{6}$, and operation $6i$ is the dummy operation of job $i$. The technological order for each job can be inferred from Figure 4.2a. Operations marked with the same color require the same machine, dummy operations are marked gray. Figure 4.2a also represents a solution to the problem instance. In Figure 4.2a, the operations are arranged according to a topological ordering of the solution and so the sequence of operations on each machine can be inferred. For e.g. on the yellow machine, the order is 20, 2, 7, 17. In Figure 4.2b, we show the critical path for our solution. In Figure 4.2c, each pair provides information for a critical block. In each pair, the item on the left is the critical block and the figure to its right is the sub-path on the critical path from which the critical block was derived.

*neighbor results. The block* $(4, 16)$ *is a type-2 critical block. In this case, the forward move makes 4 the machine successor of 16. Finally, the block* $(16, 10)$ *is a type-1 critical block so the forward move makes 16 the successor of 10. However, in this case the incoming arc into 16 on the critical path is the alternative arc* $(5, 16)$*, and consequently the backward move for the block* $(16, 10)$ *makes 10 the predecessor of 4.*

## 4.5 Enumerating all *N4* neighbors

In this section, we describe an efficient procedure for determining the feasibility of solutions generated by *N4* moves. Let $S$ be a feasible selection, and $cB = (u_1, u_2, \ldots, u_n, l)$ be a critical block in the critical path of $G(F \cup S)$. Recall from the definition of a critical block that $u_{i+1} = \delta_S(u_i), i = 1, \ldots, n-1$ and $\delta_S(u_n) = l$. The forward *N4* move corresponding to $u_i$ makes $u_i$ the machine successor of $l$. We present theoretical results showing that if the forward move results in a cycle in the graph (meaning the move is infeasible), then the cycle must contain the arc from the job successor of $l$ to $u_i$ (Lemma 2). We use this result (in Theorem 2) to prove that the feasibility of a forward move can be determined by checking for the existence of a path in the initial $G(F \cup S)$, from $u_i$ to the job successor of $l$, that does not contain the arc from the job successor of $u_i$ to the machine successor of $u_i$ (which is deleted by the forward move). In Lemma 3 and Theorem 3, analogous results are shown for the backward move.

Let $S_i$ be the selection obtained by making $u_i$ the machine successor of $l$. Based on the theoretical results, we present an algorithm (Algorithm 1) that checks the feasibility of a forward move by considering the outgoing arcs from $u_i$ and looking for a path in $G(F \cup S)$ to the job successor of $l$ containing these arcs (and not containing the arc from the job successor

of $u_i$ to the machine successor of $u_i$). The path's existence is a certificate for the inconsistency of $S_i$ (as stated in Theorem 2). The algorithm uses a topological ordering of the nodes. For nodes in the topological order between the first one in the critical block and ending with the job successor of $l$, starting with the node before the job successor of $l$, and moving to the left, the algorithm computes the longest path to the job successor of $l$. The longest path quantities computed are then used to determine the subset of forward *N4* moves associated with the critical block that result in a feasible neighbor.

Next, we separately consider consistency checking for the forward and backward move. We emphasize that the algorithms presented for consistency checking below work with the minimal representations of feasible selections.

### 4.5.1 Consistency checking for the forward move

Given the **minimal representation** $S$ of some feasible selection, consider the forward move in which operation $u$ is made the machine successor of operation $l$, and let $S'$ be the resulting selection. To generate $S'$, the arcs $(\gamma(\beta_S(u)), u)$, $(\gamma(u), \delta_S(u))$ and $(\gamma(l), \delta_S(l))$ must be removed from $S$, and arcs $(\gamma(\beta_S(u)), \delta_S(u))$, $(\gamma(l), u)$ and $(\gamma(u), \delta_S(l))$ must be added. We will recall this fact repeatedly in the proofs presented in this section. Portions of the graphs $G(F \cup S), G(F \cup S')$ that are affected by the move are shown in Figures 4.3a, 4.3b.

We begin with a lemma. If $T_S$ is a topological ordering of the nodes in $G(F \cup S)$, Lemma 1 (below) identifies paths in graphs $G(F \cup S)$ and $G(F \cup S')$ that are identical. First, a forward move will not change either the reachable nodes or any of the possible paths originating in a node that appears in $T_S$ after the position of the job successor of $l$, i.e., $\gamma(l)$. Second, for any of the nodes appearing before the position of $u$ in $T_S$, the forward move of $u$ will not change either the set of paths terminating in that node, or the set of nodes from which that node can be reached.

**Lemma 1.** *1. For $o \in \tilde{O}$ such that $T_S^{-1}(o) > T_S^{-1}(\gamma(l))$, we have $P_S(o) = P_{S'}(o)$ and $R_S(o) = R_{S'}(o)$.*
*2. For $o \in \tilde{O}$ such that $T_S^{-1}(o) < T_S^{-1}(u)$, we have $\bar{P}_S(o) = \bar{P}_{S'}(o)$ and $\bar{R}_S(o) = \bar{R}_{S'}(o)$.*

*Proof.* We prove 1; the proof of 2 is analogous. Let $X = \{o \,|\, o \in \tilde{O}, T_S^{-1}(o) > T_S^{-1}(\gamma(l))\}$. For every $o \in X$, the definition of $T_S$ implies that $R_S(o) \subseteq X$. While manipulating $S$ to get $S'$, observe that no arc whose tail is a node present in $X$ is added or removed. Since $R_S(o) \subseteq X \ \forall o \in X$, any path in $G(F \cup S)$ originating from a node in $X$ can only contain nodes in $X$, by the previous observation it follows then that such a path must also be present in $G(F \cup S')$. Conversely, suppose $G(F \cup S')$ contains a path originating from a node in $X$ but absent in $G(F \cup S)$, then it must be the case that $S'$ contains an arc $(v, w)$, where $v \in X$ and $w \notin X$. Since $v \in X$, by our previous observation it implies $(v, w)$ is also present in $S$. However this contradicts the earlier established fact $R_S(v) \subseteq X$ since $w \in R_S(v)$ but $w \notin X$. So no such $(v, w)$ arc could have been present in $S'$ to begin with. □

Lemma 2 (below) states that if the forward move of $u$ results in an inconsistent selection, then the cycle in the graph is caused by the new (alternative) arc $(\gamma(l), u)$ added to get $S'$.

**Lemma 2.** *If $S'$ is inconsistent, then every cycle in $G(F \cup S')$ contains the arc $(\gamma(l), u)$ .*

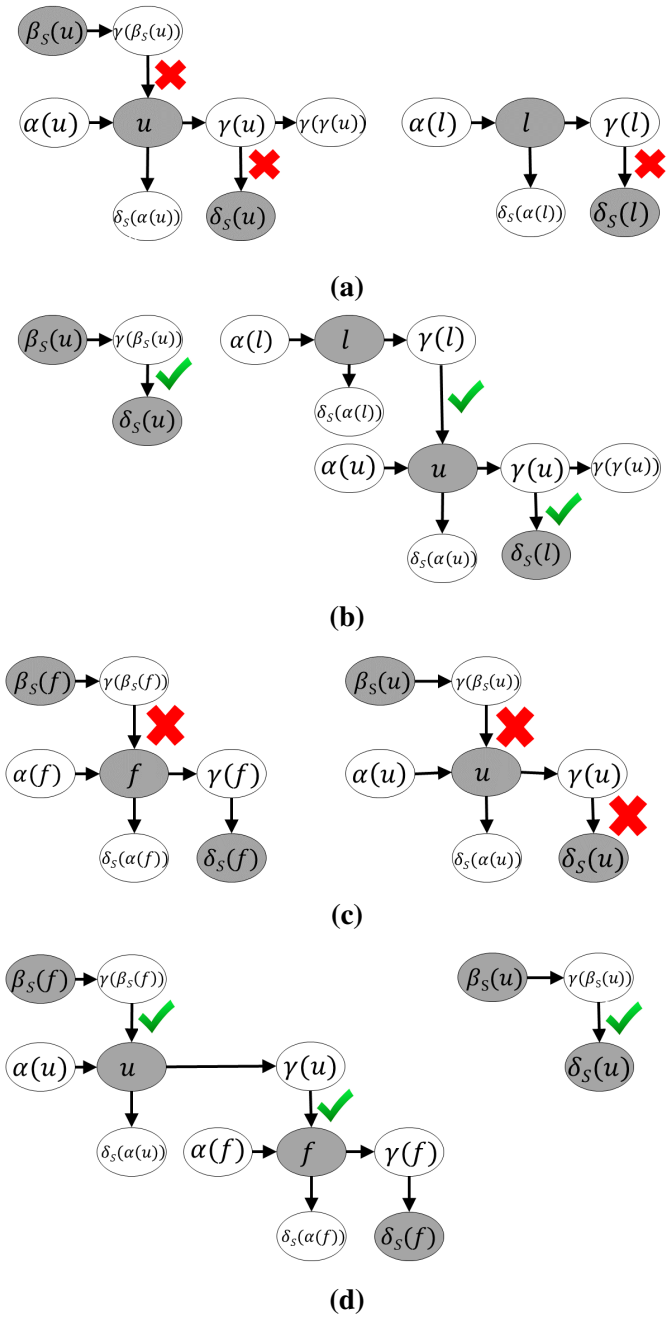



**Figure 4.3:** Figures (4.3a) (resp. (4.3c)) show a partial portion of $G(F \cup S)$. Red crosses indicate that those arcs will be removed by the forward move (resp. backward move). Figures (4.3b) (resp. (4.3d)) show a partial portion of $G(F \cup S')$. Green arrows indicate that those arcs were added by the forward move (resp. backward move). In figures (4.3a), (4.3c), (4.3b) and (4.3d), only arcs that are referenced in the makespan computation of $G(F \cup S')$ are shown.

*Proof.* If $S'$ is inconsistent, then $G(F \cup S')$ contains a cycle. Suppose the cycle in $G(F \cup S')$ does not contain any of the following arcs: $(\gamma(l), u)$, $(\gamma(\beta_S(u)), \delta_S(u))$, $(\gamma(u), \delta_S(l))$, then observe that such a cycle necessarily passes through arcs present in $G(F \cup S)$ also, but since we know that $S$ is consistent this case cannot happen. So the cycle must necessarily contain at least one of those 3 arcs mentioned earlier. Suppose the cycle contains the arc $(\gamma(\beta_S(u)), \delta_S(u))$, then it must have been the case that $\delta_S(u) \in \bar{R}_{S'}(\gamma(\beta_S(u)))$. Clearly $\delta_S(u) \notin \bar{R}_S(\gamma(\beta_S(u)))$, since otherwise $S$ could not have been consistent. So by extension $\delta_S(u) \notin \bar{R}_{S'}(\gamma(\beta_S(u)))$, since $\bar{R}_{S'}(\gamma(\beta_S(u)) = \bar{R}_S(\gamma(\beta_S(u))$ by Lemma 1. Hence, we have reached a contradiction. Similarly using Lemma 1, we can show that the cycle cannot contain the arc $(\gamma(u), \delta_S(l))$ either. So every cycle in $G(F \cup S')$ contains the arc $(\gamma(l), u)$. □

Lemma 2 implies that if $S'$ is inconsistent, then there must exist a path from $u$ to $\gamma(l)$ in $G(F \cup S')$. Theorem 2 provides a procedure to check consistency of $S'$ by examining the graph $G(F \cup S)$, and looking for a path from $u$ to $\gamma(l)$ that does not contain the arc $(\gamma(u), \delta_S(u))$ (this arc is deleted when moving $u$ forward, i.e., $(\gamma(u), \delta_S(u)) \notin S'$).

**Theorem 2.** *If $S'$ is inconsistent, then any path $P$ from $u$ to $\gamma(l)$ in $G(F \cup S')$ is also present in $G(F \cup S)$. Conversely, any path $P'$ from $u$ to $\gamma(l)$ in $G(F \cup S)$ that does not pass through the arc $(\gamma(u), \delta_S(u))$ can be found in $G(F \cup S')$.*

*Proof.* In the proof of Lemma 2, we have implicitly shown that any path $P$ from $u$ to $\gamma(l)$ in $G(F \cup S')$ cannot pass through the arcs $(\gamma(\beta_S(u)), \delta_S(u))$, $(\gamma(u), \delta_S(l))$. Further, $P$ does not pass through the arc $(\gamma(l), u)$ since $P$ is a path from $u$ to $\gamma(l)$ in $G(F \cup S')$. Hence, $P$ only passes through arcs that are also present in $G(F \cup S)$, so $P$ can be found in $G(F \cup S)$ also. Note that $P$ cannot pass through the arc $(\gamma(u), \delta_S(u))$, since it is absent in $G(F \cup S')$.

To prove the second statement, observe that given some topological ordering $T_S$ for the nodes in $G(F \cup S)$, the portion of $P'$ to $\gamma(l)$ in $G(F \cup S)$ can only contain nodes from the set $W = \{o | o \in \tilde{O}, T_S^{-1}(u) \leq T_S^{-1}(o) < T_S^{-1}(\gamma(l))\}$. Notice that $\gamma(u)$ is the only node in $W$ with an outgoing alternative arc that is different in $S$ and $S'$, i.e., $(\gamma(u), \delta_S(u)) \in S$ but $(\gamma(u), \delta_S(u)) \notin S'$. The outgoing arcs of the remaining nodes in $W$ are identical to those in $G(F \cup S')$. So we conclude that $P'$ only passes through arcs which are also present in $G(F \cup S')$. □

**Example 9.** *Let's apply Theorem 2 to determine the feasibility of the neighbors generated by the forward move for the case shown in Example 8. Consider the neighbor generated by applying the forward move to the critical block $(16, 10)$ in which 16 (i.e., $u$) becomes the machine successor of 10 (i.e., $l$). Since 11 is the machine successor of 15, $G(F \cup S)$ (where $S$ corresponds to the solution shown in Figure 4.2) contains the arc $(16, 11)$ (and by extension a path from $u$ to $\gamma(l)$), and so Theorem 2 would predict that the forward move will generate an infeasible neighbor. Let us verify this claim. First note that both in the old solution and the neighbor, 15 is the machine predecessor of 11, and the forward move makes 16 the machine successor of 10 in the neighbor. Clearly the neighbor generated contains a deadlock situation involving operations 10,11,15 and 16 (substitute them for $o11, o12, o21, o22$ in Figure 4.1 for visualizing the deadlock), and so the neighbor is indeed infeasible. We leave it as an exercise for the reader to verify that the only feasible neighbors in Example 8 generated by applying forward moves are those obtained from critical blocks $(8, 5)$ and $(4, 16)$.*

Analogous results relating feasibility of a neighbor and the existence of certain paths in $G(F \cup S)$ (such as path $P$ defined in Theorem 2) have been published in Balas and Vazacopoulos [1998] for the JS problem. In this work we extend these results for the BJS problem. Further, we exploit these results to propose an efficient algorithm for enumerating all feasible neighbors that are generated by applying *N4* moves to a critical block.

**Algorithm to determine feasibility of forward moves**

Algorithm 5 determines the subset of forward *N4* moves associated with a critical block that result in a feasible neighbor. Consider the critical block $cB = (u_1, \ldots, u_n, l)$ $(n \geq 1)$. To determine whether any given forward move for operation $u_i$ is feasible, the algorithm checks for the existence of a path from $u_i$ to $\gamma(l)$ in $G(F \cup S)$ that does not go through the arc $(\gamma(u_i), \delta_S(u_i))$. If such a path exists, then by Theorem 2 we know that $S_i$ is inconsistent. The only outgoing arcs from $u_i$ in $G(F \cup S)$ are $(u_i, \gamma(u_i))$ and $(u_i, \delta_S(\alpha(u_i)))$. The only outgoing arcs from $\gamma(u_i)$ are to $\gamma(\gamma(u_i))$ and $\delta_S(u_i)$, and also recall that $(\gamma(u_i), \delta_S(u_i))$ is removed by the forward move (see Figure 4.3a). Consequently, determination of forward move feasibility checking for $u_i$ reduces to checking two conditions:

1. Whether there is a path from $\gamma(\gamma(u_i))$ to $\gamma(l)$ in $G(F \cup S)$. If a path exists, i.e., if $L_S(\gamma(\gamma(u_i)), \gamma(l)) > -\infty$, then the forward move is infeasible.

2. Whether there is a path from $\delta_S(\alpha(u_i))$ to $\gamma(l)$ in $G(F \cup S)$ that does not pass through the arc $(\gamma(u_i), \delta_S(u_i))$, in which case the forward move is infeasible. Contrary to the previous case, this check is not equivalent to determining if $L_S(\delta_S(\alpha(u_i)), \gamma(l)) > -\infty$, since the longest path (if any) may have passed through the arc $(\gamma(u_i), \delta_S(u_i))$ (which is removed by the forward move of $u_i$).

If neither of these checks yield a path, then the forward move of $u_i$ is feasible.

To perform these checks, Algorithm 5 uses two arrays $A, B$ of length $|\tilde{O}|$ to store longest path costs:[1]

- For $i \in \left[T_S^{-1}(u_1), T_S^{-1}(\gamma(l)\right]$, we compute $L_S(T_S[i], \gamma(l))$ and store it in $A[i]$.
- For $i \in [1, n]$ and $j \in \left(T_S^{-1}(u_i), T_S^{-1}(\gamma(u_i))\right]$, we compute the longest path cost from $T_S[j]$ to $\gamma(l)$ that does not pass through the arc $(\gamma(u_i), \delta_S(u_i))$ and store it in $B[j]$, i.e., $B[j] = L_{S \setminus (\gamma(u_i), \delta_S(u_i))}(T_S[j], \gamma(l))$.

To facilitate computation of these checks, we define an operator $\overset{T_S}{\succeq}$ (shown in Eqn (4.1)) that compares nodes in $\tilde{O}$ based on their position in $T_S$. Similarly, $\overset{T_S}{\succ}$ is the operator obtained by replacing $\geq$ in Eqn (4.1) with $>$.

$$o_1 \overset{T_S}{\succeq} o_2 := \begin{cases} \text{true, if } T_S^{-1}(o_1) \geq T_S^{-1}(o_2) \\ \text{false, otherwise} \end{cases} \tag{4.1}$$

[1]Note that while our algorithm computes longest path costs to determine the existence of paths, the same results could be obtained by simply computing reachability. The computational cost is the same either way.

**Algorithm 5** Feasibility Check($T_S$, $cB = (u_1, \ldots, u_n, l)$)

1: **for** $i \in \left[T_S^{-1}(u_1), T_S^{-1}(\gamma(l))\right)$ **do**
2: $A[i] \leftarrow -\infty$, $B[i] \leftarrow -\infty$
3: $i \leftarrow T_S^{-1}(\gamma(l))$, $k \leftarrow n$, $A[i] \leftarrow 0$
4: **while** $i > T_S^{-1}(u_1)$ **do**
5: $i \leftarrow i - 1$, $o \leftarrow T_s[i]$
6: **if** $\gamma(l) \overset{T_S}{\succeq} \gamma(o)$ and $A[T_S^{-1}(\gamma(o))] \neq -\infty$ **then**
7: $A[i] \leftarrow A[T_S^{-1}(\gamma(o))] + p(o)$
8: **if** $\gamma(l) \overset{T_S}{\succeq} \delta_S(\alpha(o))$ **then**
9: $A[i] \leftarrow \max(A[i], A[T_S^{-1}(\delta_S(\alpha(o)))])$
10: **if** $\gamma(u_n) \overset{T_S}{\succeq} o$ **then**
11: **if** $o = \gamma(u_k)$ and $\gamma(l) \overset{T_S}{\succeq} \gamma(\gamma(u_k))$ **then**
12: **if** $A[T_S^{-1}(\gamma(\gamma(u_k)))] \neq -\infty$ **then**
13: $B[i] \leftarrow p(o) + A[T_S^{-1}(\gamma(\gamma(u_k)))]$
14: **if** $\gamma(u_k) \overset{T_S}{\succ} o$ and $o \neq u_k$ **then**
15: **if** $\gamma(u_k) \overset{T_S}{\succ} \gamma(o)$ and $B[T_S^{-1}(\gamma(o))] \neq -\infty$ **then**
16: $B[i] \leftarrow p(o) + B[T_S^{-1}(\gamma(o))]$
17: **else if** $\gamma(l) \overset{T_S}{\succeq} \gamma(o) \overset{T_S}{\succ} \gamma(u_k)$ and $A[T_S^{-1}(\gamma(o))] \neq -\infty$ **then**
18: $B[i] \leftarrow p(o) + A[T_S^{-1}(\gamma(o))]$
19: **if** $\gamma(u_k) \overset{T_S}{\succeq} \delta_S(\alpha(o))$ **then**
20: $B[i] \leftarrow max(B[i], B[\delta_S(\alpha(o))])$
21: **else if** $\gamma(l) \overset{T_S}{\succeq} \delta_S(\alpha(o)) \overset{T_S}{\succ} \gamma(u_k)$ **then**
22: $B[i] \leftarrow max(B[i], A[T_S^{-1}(\delta_S(\alpha(o))])$
23: **if** $o = u_k$ **then**
24: $k \leftarrow k - 1$

For critical block $cB$, the steps to compute $A$ and $B$ are specified in Algorithm 5. The algorithm proceeds by iterating over the nodes in reverse topological order from $\gamma(l)$ to $u_1$. For each node $o$ encountered, lines 6 through 9 compute $L_S(o, \gamma(l))$ as the longest path between $o$ and $\gamma(l)$ through the outgoing arcs from $o$, i.e., $(o, \gamma(o))$ and $(o, \delta_S(\alpha(o)))$ and store it in buffer $A$. By traversing $T_S$ in the reverse topological order, both $L_S(\gamma(o), \gamma(l))$ and $L_S(\delta_S(\alpha(o)), \gamma(l))$ have been previously computed and stored in $A$; and these values are used to compute $L_S(o, \gamma(l))$.

Buffer $B$ is populated in lines 11 through 22. Since $u_n$ is the last operation in $cB$ for which feasibility of the forward move needs to be determined, $T_S^{-1}(\gamma(u_n))$ is the last index of $B$ needed (line 10). Recall from the definition of buffer $B$, for $u \in \{u_1, \ldots, u_n\}$ and $j \in \left(T_S^{-1}(u), T_S^{-1}(\gamma(u))\right]$, $B[j]$ needs to be set to $L_{S \setminus (\gamma(u), \delta_S(u))}(T_S[j], \gamma(l))$. In lines 11 through 13, we compute the longest such path from $\gamma(u)$ to $\gamma(l)$ passing through the arc $(\gamma(u), \gamma(\gamma(u)))$. In lines 14 through 22, nodes occurring between $T_S^{-1}(u)$ and $T_S^{-1}(\gamma(u))$

are encountered in reverse topological order. The two outgoing neighbors for each node are considered separately. If the outgoing node occurs later than $\gamma(u)$ in $T_S$, then a path to $\gamma(l)$ is unaffected by the arc $(\gamma(u), \delta_S(u))$, and the corresponding path cost can be computed using values in $A$, as shown in lines 17-18 and 21-22. If the outgoing node occurs no later than $\gamma(u)$ in $T_S$, then the longest path cost from that outgoing node to $\gamma(l)$ not passing through the arc $(\gamma(u), \delta_S(u))$ was already computed and stored in buffer $B$, and can be used as shown in lines 15-16 and 19-20.

Once buffers $A, B$ are populated using Algorithm 5, Proposition 2 identifies all the feasible forward moves for critical block $cB$. The complexity of executing Algorithm 5 is $\mathcal{O}(TL(cB))$, where $TL(cB) = T_S^{-1}(\gamma(l)) - T_S^{-1}(u_1)$.

**Proposition 2.** *The forward move that makes $u_i$ the machine successor of $l$ results in a feasible selection iff none of these conditions are true:*

*(i)* $B[T_S^{-1}(\gamma_S(u_i))] \neq -\infty$

*(ii)* $(\gamma(u_i) \overset{T_S}{\succ} \delta_S(\alpha(o)))$ *and* $B[T_S^{-1}(\delta_S(\alpha(o)))] \neq -\infty$

*(iii)* $\gamma(l) \overset{T_S}{\succeq} \delta_S(\alpha(o)) \overset{T_S}{\succ} \gamma(u_i)$ *and* $A[T_S^{-1}(\delta_S(\alpha(o)))] \neq -\infty$

*Proof.* Condition (i) checks the existence of a path from $u_i$ to $\gamma(l)$ through the arc $(\gamma(u_i), \gamma(\gamma(u_i)))$. Conditions (ii) and (iii) check the existence of a path from $u_i$ to $\gamma(l)$ through arc $(u_i, \delta_S(\alpha(u_i)))$, that does not pass through $(\gamma(u_i), \delta_S(u_i))$. □

Given buffers $A, B$, the complexity of checking the conditions mentioned in Proposition 2 for each forward move is $\mathcal{O}(1)$. So the overall complexity of checking feasibility of the outputs of all forward moves in the critical block $cB$ is $\mathcal{O}(TL(cB))$.

### 4.5.2 Consistency checking for backward moves

Let $S$ be the minimal representation of a feasible selection. Consider the backward move in which $u$ becomes the machine predecessor of operation $f$, and denote the output of the move by $S'$. To generate $S'$, the arcs $(\gamma(\beta_S(f)), f)$, $(\gamma(\beta_S(u)), u)$ and $(\gamma(u), \delta_S(u))$ must be removed from $S$, and arcs $(\gamma(\beta_S(f)), u)$, $(\gamma(u), f)$ and $(\gamma(\beta_S(u)), \delta_S(u))$ must be added.

We state a Lemma analogous to Lemma 1 and a theorem analogous to Theorem 2.

**Lemma 3.** *1. For $o \in \tilde{O}$ such that $T_S^{-1}(o) < T_S^{-1}(f)$, we have $\bar{P}_S(o) = \bar{P}_{S'}(o)$ and $\bar{R}_S(o) = \bar{R}_{S'}(o)$.*

*2. For $o \in \tilde{O}$ such that $T_S^{-1}(o) > T_S^{-1}(\gamma(u))$, we have $P_S(o) = P_{S'}(o)$ and $R_S(o) = R_{S'}(o)$.*

Lemma 3 states that a backward move will not change the paths to any node that appears in the topological order $T_S$ before $f$, and it will also not change the paths originating from a node that appears in $T_S$ after the job successor of the operation $u$ that is moved.

**Theorem 3.** *If $S'$ is inconsistent, then there exists a path from $f$ to $\gamma(u)$ in $G(F \cup S')$. Any such path $P$ from $f$ to $\gamma(u)$ in $G(F \cup S')$ is also present in $G(F \cup S)$. Conversely, any path $P'$ from $f$ to $\gamma(u)$ in $G(F \cup S)$ which does not pass through the arc $(\gamma(\beta_S(u)), u)$ is also found in $G(F \cup S')$.*

The proof of Theorem 3 is similar to the proof for Theorem 2. Based on Theorem 3, an algorithm analogous to Algorithm 5 and Proposition 2 can be formulated; these are not shown.

We have also developed a theoretical basis for efficiently computing the makespan of the feasible neighbors and these results are also incorporated into the tabu search procedure described later in Section 4.8. However, given the high percentage of infeasible solutions that are typically produced using the *N4* neighborhood in the BJS context, the key determiner of solving efficiency is the procedure for recovering feasibility. So we consider feasibility recovery in Section 4.6. For completeness, the makespan computation for feasible *N4* neighbors is included in the Appendix, see Section B.

## 4.6 Job insertion feasibility recovery (JIFR)

For any generated move that is determined to be inconsistent, the next step is to transform it into a feasible neighbor solution. In this section, we describe two JIFR algorithms that can be applied to accomplish this. Before describing the algorithms, we briefly introduce some terminology. An alternative arc $a$ is said to be associated to a job $\mathcal{J}$ if either $J(tail(a))$ or $J(head(a))$ is $\mathcal{J}$, where $head(a)$ (resp. $tail(a)$) refers to the head (resp. tail) of arc $a$. An alternative arc pair is said to be associated with a job $\mathcal{J}$ if the arcs in the pair are associated to $\mathcal{J}$. At a high level, a JIFR algorithm takes as input an inconsistent selection generated by a *N4* move, and then chooses a job whose operations can be rearranged to produce a feasible selection. All the alternative arcs associated to the chosen job are removed from the inconsistent selection and the resulting consistent partial selection is then extended to a complete consistent (feasible) selection by reinserting the job. The two JIFR algorithms that will be presented in this section differ in the way that the retracted job is chosen; both use the same job insertion algorithm presented in Section 4.7. The job insertion procedure takes as inputs a job $\mathcal{J}$, a consistent selection $S_{\mathcal{J}}$ with all alternative arc pairs associated with $\mathcal{J}$ *unselected* in $S_{\mathcal{J}}$, and an alternative arc $a$ associated with $\mathcal{J}$, and the procedure outputs a feasible selection $\bar{S}$ with $\bar{S}$ containing $a$.

### 4.6.1 JIFR-1

JIFR-1 (Algorithm 6) is similar to approaches from earlier work [Bürgy, 2017; Dabah *et al.*, 2017]. By removing all alternative arcs associated with job $\mathcal{J}$ in line 4, any cycle that was present in $S'$ earlier can no longer remain after the arcs have been removed, a fact that is deducible from Lemma 2. In line 5, setting $a = (\gamma(l), u)$ means that we are searching for a feasible selection $\bar{S}$ where $u$ is processed after $l$, but $u$ is not constrained to be the machine successor of $l$. A feasible selection $\bar{S}$ can always be recovered in line 5. An example of such a selection is to construct $S'$ by adding arcs from the *unselected* arc pairs, such that, the last

**Algorithm 6** JIFR-1 for the *N4* forward move

1: **Given:** Inconsistent selection $S'$, generated from solution $S$ by making $u$ the machine successor of $l$.
2: **Output:** A feasible selection $\bar{S}$.
3: Select job $\mathcal{J}$ from the set $\{J(u), J(l)\}$.
4: Remove all arcs associated with $\mathcal{J}$ from $S'$
5: Apply Job Insertion (i.e., Algorithm 8) with $\mathcal{J}$, $S'$ and $a = (\gamma(l), u)$ as input to obtain $\bar{S}$.
6: **return** $\bar{S}$

operation on every machine belongs to job $\mathcal{J}$. The complexity of Algorithm 6 is dominated by the complexity of the Job Insertion procedure.

Algorithm 6 assumes that the infeasible selection was generated by a *N4* forward move; adapting it to the backward move is easy. Suppose the backward move makes operation $u$ the machine predecessor of $f$, then the changes to make to Algorithm 6 are: in line 3, $\mathcal{J}$ is selected from the set $\{J(u), J(f)\}$, and arc $a$ in line 5 is set to $(\gamma(u), f)$.

### 4.6.2 JIFR-2

JIFR-2 provides an alternative way to convert the inconsistent selection $S'$ into a consistent selection. At a high level for the output of a forward move, JIFR-2 finds a job $\mathcal{J}$ other than $J(l), J(u)$, and removes all arcs associated with $\mathcal{J}$ from $S'$. A job insertion procedure is then invoked to reinsert $\mathcal{J}$ and extend $S'$ to a feasible selection. When performing the *N4* forward move on the minimal selection, the new selection obtained by omitting a job $\mathcal{J}$ can be constructed by following three steps shown below. We present theoretical results for determining whether job $\mathcal{J}$ (different from $J(l)$ or $J(u)$) disconnects $\gamma(l)$ and $u$, meaning that the new selection is consistent; the proofs are based on the minimal representation of the feasible selection $S$.

Let $S_{min}$ denote the minimal representation of the feasible selection $S$, and let $S'_{min}$ denote the selection obtained by performing the *N4* forward move on $S_{min}$. Since $S'$ is inconsistent, by Lemma 2 we know that all cycles in $G(F \cup S'_{min})$ pass through the arc $(\gamma(l), u)$. JIFR-2 identifies a job such that the removal of this job breaks all paths from $u$ to $\gamma(l)$, disconnecting all cycles. For any job $\mathcal{J}$ other than $J(l), J(u)$, let $S'_{\mathcal{J}}$ denote the selection obtained by omitting job $\mathcal{J}$ from $S'_{min}$. $S'_{\mathcal{J}}$ can be constructed as follows:

1. Copy all arcs in $S'_{min}$ to $S'_{\mathcal{J}}$.
2. To skip job $\mathcal{J}$ we add all arcs from set $H_{\mathcal{J}}$ to $S'_{\mathcal{J}}$, where $H_{\mathcal{J}} = \{(\gamma(\beta_{S'}(o)), \delta_{S'}(o)) \,|o \in O, J(o) = \mathcal{J}\}$.
3. Remove all arcs associated with job $\mathcal{J}$ from $S'_{\mathcal{J}}$.

If $G(F \cup S'_{\mathcal{J}})$ is consistent, then $\mathcal{J}$ is a job that *disconnects* $\gamma(l)$ and $u$. If $\mathcal{J}$ disconnects $\gamma(l)$ and $u$, we can use the Job Insertion procedure to reinsert $\mathcal{J}$.

**Example 10.** *Note that the forward move applied to the critical block* $(2, 7)$ *in Example 8 generates an infeasible neighbor.* $G(F \cup S)$ *(where* $S$ *corresponds to the solution shown in the top row of Figure 4.2) contains the path* $2, 15, 23, 8$ *which by Theorem 2 is a certificate of infeasibility for the forward move. We will apply JIFR-2 to restore feasibility. Since by Theorem 2 one of the cycles that is present in the infeasible neighbor (i.e.,* $G(F \cup S')$*) is* $2, 15, 23, 8, 2$*, one possibility to disconnect the cycle is to remove operations belonging to either job* $J(15)$ *(i.e., 3) or* $J(23)$ *(i.e., 4) (note JIFR-1 would have selected* $J(2) = 1$ *or* $J(7) = 2$ *instead). We will leave at as an exercise for the reader to verify that* $G(F \cup S'_{\mathcal{J}})$ *is indeed consistent if* $\mathcal{J}$ *is 3 or 4.*

## Determining whether $\mathcal{J}$ *disconnects* $\gamma(l)$ and $u$

Checking whether $\mathcal{J}$ *disconnects* $\gamma(l)$ and $u$ is equivalent to determining whether $G(F \cup S'_{\mathcal{J}})$ does not contain any cycles. By exploiting the fact if $S'_{\mathcal{J}}$ is inconsistent then every cycle in $G(F \cup S'_{\mathcal{J}})$ contains the arc $(\gamma(l), u)$, Theorem 4 provides a simple procedure to determine consistency of $S'_{\mathcal{J}}$.

**Lemma 4.** *If* $S'_{\mathcal{J}}$ *is inconsistent, then every cycle in* $G(F \cup S'_{\mathcal{J}})$ *contains the arc* $(\gamma(l), u)$.

*Proof.* Suppose $o_1, o_2$ are a pair of nodes which are not connected by a path in $G(F \cup S'_{min})$, then observe that $o_1$ and $o_2$ could not have been connected by a path in $G(F \cup S'_{\mathcal{J}})$ either. The reader can verify this fact from the definition of $H_{\mathcal{J}}$, since every arc in $H_{\mathcal{J}}$ is between a pair of nodes which is connected by a path in $G(F \cup S'_{min})$. So by extension, any cycle in $G(F \cup S'_{\mathcal{J}})$ is at best a shorter version of some cycle that already exists in $G(F \cup S'_{min})$.

By Lemma 2, every cycle in $G(F \cup S'_{min})$ contains the arc $(\gamma(l), u)$. From Theorem 2, we know that every path from $u$ to $\gamma(l)$ in $G(F \cup S'_{min})$ is present in $G(F \cup S_{min})$. Hence, all nodes involved in any cycle present in $G(F \cup S'_{min})$ must belong to the set $\{o | o \in \tilde{O}, T_S^{-1}(u) \leq T_S^{-1}(o) \leq T_S^{-1}(\gamma(l))\}$. Based on the above arguments, for a (short) cycle by-passing the arc $(\gamma(l), u)$ to exist in $G(F \cup S'_{\mathcal{J}})$, it must be the case that $S'_{\mathcal{J}}$ contains an arc $(x, y)$ (other than $(\gamma(l), u)$) with $T_S^{-1}(u) \leq T_S^{-1}(y) < T_S^{-1}(x) \leq T_S^{-1}(\gamma(l))$. Clearly $G(F \cup S_{min})$ contains no such arc $(x, y)$, since otherwise $T_S$ is not a valid topological order for $G(F \cup S)$. Further, observe that no such arc $(x, y)$ was included while constructing $S'_{min}$ from $S_{min}$, and $S'_{\mathcal{J}}$ from $S'_{min}$. Hence no such arc $(x, y)$ is present in $G(F \cup S'_{\mathcal{J}})$. Hence, all cycles in $G(F \cup S'_{\mathcal{J}})$ contains the arc $(\gamma(l), u)$. □

**Theorem 4.** *If* $S'_{\mathcal{J}}$ *is inconsistent, then every path from* $u$ *to* $\gamma(l)$ *in* $G(F \cup S'_{\mathcal{J}})$ *can be found by searching for a path from* $u$ *to* $\gamma(l)$ *in* $G(F \cup S_{min} \cup H_{\mathcal{J}})$ *such that the path does not contain the arc* $(\gamma(u), \delta_S(u))$ *nor any node that belongs to job* $\mathcal{J}$*. Conversely every path from* $u$ *to* $\gamma(l)$ *in* $G(F \cup S_{min} \cup H_{\mathcal{J}})$ *that does not contain the arc* $(\gamma(u), \delta_S(u))$ *nor any node that belongs to job* $\mathcal{J}$ *can also be found in* $G(F \cup S'_{\mathcal{J}})$.

*Proof.* By Lemma 4, consistency checking of $G(F \cup S'_{\mathcal{J}})$ is equivalent to checking for the existence of a path from $u$ to $\gamma(l)$ in $G(F \cup S'_{\mathcal{J}})$. Checking for a path from $u$ to $\gamma(l)$ in $G(F \cup S'_{\mathcal{J}})$ is the same as checking for a path from $u$ to $\gamma(l)$ in $G(F \cup S'_{min} \cup H_{\mathcal{J}})$ which does not contain any node belonging to job $\mathcal{J}$. Nodes belonging to job $\mathcal{J}$ need to be skipped because arcs associated with $\mathcal{J}$ were removed in step 3 while constructing $S'_{\mathcal{J}}$ from $S'_{min}$.

Observe that both in the case of $G(F \cup S'_{min})$ and $G(F \cup S_{min})$, all arcs in $H_{\mathcal{J}}$ are between nodes that are previously connected by a path in those graphs. So adding arcs from $H_{\mathcal{J}}$ to $G(F \cup S'_{min})$ or $G(F \cup S_{min})$ cannot lead to the creation of a path between a pair of nodes if the pair of nodes was not connected by a path prior to the addition of arcs from $H_{\mathcal{J}}$. We know from Theorem 2 that the set of all paths from $u$ to $\gamma(l)$ in $G(F \cup S'_{min})$ is identical to the set of all paths from $u$ to $\gamma(l)$ in $G(F \cup S_{min})$ where every path in the latter set must not contain the arc $(\gamma(u), \delta_S(u))$. So we can deduce that the set of all paths from $u$ to $\gamma(l)$ in $G(F \cup S'_{min} \cup H_{\mathcal{J}})$ is identical to the set of all paths from $u$ to $\gamma(l)$ in $G(F \cup S_{min} \cup H_{\mathcal{J}})$ where every path in the latter set must not contain the arc $(\gamma(u), \delta_S(u))$. So consistency checking of $G(F \cup S'_{\mathcal{J}})$ is the same as checking the existence of a path from $u$ to $\gamma(l)$ in $G(F \cup S_{min} \cup H_{\mathcal{J}})$ that does not contain the arc $(\gamma(u), \delta_S(u))$ nor any node that belongs to job $\mathcal{J}$. □

Theorem 4 tells us that consistency checking of $S'_{\mathcal{J}}$ can be performed using $S_{min}$ in complexity $\mathcal{O}(T_S^{-1}(\gamma(l)) - T_S^{-1}(u))$, simply by determining whether there exists a path from $u$ to $\gamma(l)$ that does not contain the arc $(\gamma(u), \delta_S(u))$, nor any node that belongs to job $\mathcal{J}$.

### 4.6.3 Algorithm to perform JIFR-2

**Algorithm 7** JIFR-2 for the *N4* forward move

1: **Given:** Inconsistent selection $S'$, generated from solution $S$ by making $u$ the machine successor of $l$.
2: **Output:** A feasible selection $\bar{S}$.
3: Select a job $\mathcal{J}$ other than $J(u), J(l)$ that disconnects $\gamma(l)$ and $u$. If no such job can be found, then **return**.
4: Select any arc $a$ associated with $\mathcal{J}$ in $G(F \cup S)$ such that $a$ occurs on a path from $u$ to $\gamma(l)$ in $G(F \cup S)$.
5: Remove all arcs associated with $\mathcal{J}$ from $S'$.
6: Apply Algorithm 8 with $\mathcal{J}$, $S'$ and $\bar{a}$ as input to obtain $\bar{S}$, where $\bar{a}$ is the paired arc of $a$ that was *forbidden* in $S$.
7: **return** $\bar{S}$.

Algorithm 7 describes JIFR-2. Note that there can be more than one eligible candidate for job $\mathcal{J}$ in line 3, and more than one candidate arc $a$ in line 4 for the chosen job $\mathcal{J}$. While Algorithm 7 shows how to generate a single neighbor, we can generate more neighbors by considering all combinations of job and arc pairs satisfying the conditions in lines 3 and 4. The size of the neighborhood can be controlled by the user by omitting some of those combinations. The overall complexity of executing Algorithm 7 is dominated by the Job Insertion procedure.

The rationale to select $\bar{a}$ (i.e., the paired alternative arc of $a$) for inclusion in $\bar{S}$ in line 6 is: if we blame arc $a$ as the reason why $S'$ is inconsistent, then to make $S'$ consistent we need to disconnect the path from $u$ to $\gamma(l)$ in $G(F \cup S')$. One way to disconnect the path would be to remove $a$ from $S'$. In other words, we are interested in a feasible selection $\bar{S}$ containing $\bar{a}$. Finally, unlike JIFR-1, $\mathcal{J}$ selected in line 7 is different from $J(u)$, $J(l)$. However, there is

**Algorithm 8** Job Insertion Algorithm

1: **Given:** Job $\mathcal{J}$ to be inserted, consistent selection $S_{\mathcal{J}}$ with all arc pairs in $\mathcal{A}_{\mathcal{J}}$ *unselected*. Arc $a$ to be included in $\bar{S}$.
2: **Output:** A complete consistent selection $\bar{S}$.
3: $\bar{S} \leftarrow S_{\mathcal{J}}$.
4: $q \leftarrow a$.
5: **repeat**
6: $\bar{S} \leftarrow \bar{S} \cup q$.
7: $B_q \leftarrow \emptyset$
8: Gather inferred arcs in set $B_q$.
9: **for** each arc $g \in B_q$ **do**
10: $\bar{S} \leftarrow \bar{S} \cup g$
11: **if** No arc pair in $\mathcal{A}_{\mathcal{J}}$ is *unselected* in $\bar{S}$ **then**
12: **return** $\bar{S}$
13: **else**
14: Apply SMCP to get the next arc $q$ to include in $\bar{S}$ from the *unselected* arc pairs in $\mathcal{A}_{\mathcal{J}}$.
15: **until** No arc pair from $\mathcal{A}_{\mathcal{J}}$ is *unselected* in $\bar{S}$.

no guarantee there is a job $\mathcal{J}$ that disconnects $\gamma(l)$ and $u$ in line 3. While with JIFR-1 we are always able to recover a feasible solution, there is no such guarantee with JIFR-2.

## 4.7 Job insertion

In this section, we describe the Job Insertion procedure. The inputs to the procedure are: a job $\mathcal{J}$ to be inserted, an alternative arc $a$ associated to $\mathcal{J}$, and a consistent selection $S_{\mathcal{J}}$ containing selections for all alternative arc pairs in $\mathcal{A}$ other than the arc pairs associated with $\mathcal{J}$. The set of arc pairs in $\mathcal{A}$ associated to $\mathcal{J}$ is denoted by $\mathcal{A}_{\mathcal{J}}$; note all arc pairs in $\mathcal{A}_{\mathcal{J}}$ are *unselected* in $S_{\mathcal{J}}$. Algorithm 8 extends $S_{\mathcal{J}}$ to a complete consistent selection $\bar{S}$ which includes $a$.

Algorithm 8 is a simple temporal propagation type procedure. In line 8, as a consequence of including arc $q$ in line 6, additional arcs that need to be included in $\bar{S}$ from the *unselected* arc pairs in $\mathcal{A}_{\mathcal{J}}$ can be inferred. The inferred arcs are included in $\bar{S}$ in line 10. In line 11, if no more *unselected* arc pairs remain in $\mathcal{A}_{\mathcal{J}}$, then the algorithm terminates, and we simply return the feasible selection $\bar{S}$ as the output. Else, we use a heuristic to select the next arc from the remaining *unselected* arc pairs in $\mathcal{A}_{\mathcal{J}}$ to include in $\bar{S}$, and go back to line 6. The heuristic we use to choose the arc to include is the Select Most Critical Pair heuristic (SMCP) [Mascis and Pacciarelli, 2002].

For each *unselected* alternative arc pair $e = \langle g_1, g_2 \rangle$ in $\mathcal{A}_{\mathcal{J}}$ remaining in line 14, SMCP first computes the value $Sh_e = \min(L_{\bar{S} \cup g_1}(\theta, \Lambda), L_{\bar{S} \cup g_2}(\theta, \Lambda))$, and selects the arc pair with the maximum $Sh_e$ value. Say the selected arc pair is $\bar{e} = \langle \bar{g_1}, \bar{g_2} \rangle$, SMCP assigns arc $q$ in line 14 to $\underset{\bar{g_1}, \bar{g_2}}{\operatorname{argmin}}(L_{\bar{S} \cup \bar{g_1}}(\theta, \Lambda), L_{\bar{S} \cup \bar{g_2}}(\theta, \Lambda))$.

Identifying the inferred arcs is one of the computationally expensive parts of the Job In-

sertion procedure. While generic ways to perform inference exist in scheduling literature (see Oddi *et al.* [2012]), computationally efficient methods specialized for BJS problems have not received sufficient attention. To perform the "inference" step efficiently, we revisit the Job Insertion polytope introduced in [Gröflin and Klinkert, 2007] and employ the polytope's structural properties to develop an efficient algorithm. These structural properties allow us to build a data structure in terms of a conflict graph, on which we can perform inference efficiently. We also point out that this insight has been previously used (see Gröflin and Klinkert [2009]; Bürgy [2017]), however we make some additional contributions of our own to improve the efficiency and these are summarized in Section 4.7.5.

### 4.7.1 Job insertion polytope

We are given as input $\mathcal{J}, S_{\mathcal{J}}, \mathcal{A}_{\mathcal{J}}$ as defined in the beginning of Section 4.7. A feasible selection $\bar{S}$ to the job insertion problem is a complete consistent selection s.t. $S_{\mathcal{J}} \subset \bar{S}$. The Job insertion polytope (**JIP**) provides a polyhedral characterization of the space of feasible selections to the job insertion problem, we review the **JIP** in this section.

Notice that any feasible selection $\bar{S}$ to the job insertion problem can be completely described by specifying which alternative arc from each alternative pair in $\mathcal{A}_{\mathcal{J}}$ is *selected* in $\bar{S}$. For each alternative arc pair $\langle a_1, a_2 \rangle \in \mathcal{A}_{\mathcal{J}}$, binary variables $x_{a_1}, x_{a_2} \in \{0, 1\}$ are introduced to indicate whether arcs $a_1, a_2$ are present in $\bar{S}$. Using these binary variables, a set of linear constraints that any feasible selection to the Job Insertion problem must satisfy is presented.

To specify the constraints, we adopt the following notations. For any directed arc $g$, recall the head of $g$ is denoted by $head(g)$ and the tail of $g$ by $tail(g)$. Given an alternative arc pair from $\mathcal{A}_{\mathcal{J}}$, it will be important to distinguish the alternative arc from the pair whose head belongs to $\mathcal{J}$ from the arc in the pair whose tail belongs to $\mathcal{J}$. Without loss of generality, for any alternative arc pair $\langle a_1, a_2 \rangle \in \mathcal{A}_{\mathcal{J}}$, we assume that $head(a_1)$ is a node belonging to $\mathcal{J}$, consequently $tail(a_2)$ also belongs to $\mathcal{J}$. Finally, we introduce a function $pos : \bar{O} \to \mathbb{N}$, which takes as input a node (say $o$) from the graph $G(F \cup S_{\mathcal{J}})$, and returns the position of $o$ within sequence of nodes belonging to job $J(o)$.

Suppose $\langle g_1, g_2 \rangle, \langle h_1, h_2 \rangle \in \mathcal{A}_{\mathcal{J}}$ are distinct arc pairs satisfying the following conditions:

(I) $pos(head(g_1)) \leq pos(tail(h_2))$,
(II) A path exists from $head(h_2)$ to $tail(g_1)$ in $G(F \cup S_{\mathcal{J}})$.

then, notice that $\bar{S}$ cannot contain both $g_1$ and $h_2$ simultaneously, since $G(F \cup \bar{S})$ will contain a non-negative length cycle. So the constraints that are introduced to ensure consistency of $\bar{S}$ (i.e., the cycle elimination constraints) are:

$$x_{g_1} + x_{h_2} \leq 1,\ \forall \langle g_1, g_2 \rangle, \langle h_1, h_2 \rangle \in \mathcal{A}_{\mathcal{J}} \text{ s.t.} \langle g_1, g_2 \rangle, \langle h_1, h_2 \rangle \text{ are distinct pairs and (I), (II) are satisfied} \tag{4.2}$$

$$x_{a_1} + x_{a_2} = 1\ \forall \langle a_1, a_2 \rangle \in \mathcal{A}_{\mathcal{J}} \tag{4.3}$$

Theorem 2 from Gröflin and Klinkert [2009] implies that constraints in Eqn (4.2) are sufficient to eliminate all cycles and thus ensure consistency of $\bar{S}$. Eqn (4.3) is the assignment constraint,

which ensures that exactly one arc is chosen from any arc pair in $\mathcal{A}_{\mathcal{J}}$. Putting them together, the Job insertion polytope is:

$$\textbf{(JIP): conv}\left(x \in \{0,1\}^{2|\mathcal{A}_{\mathcal{J}}|} | x \text{ satisfies Eqns (4.2) and (4.3)}\right), \text{ where conv denotes convex hull.} \tag{4.4}$$

According to Gröflin and Klinkert [2007], there is a 1:1 correspondence between the vertices of **JIP** and feasible selections to the Job Insertion problem. The **JIP** in Eqn (4.4) is described by $\mathbf{M}(\mathbf{J}-1)$ assignment constraints in (4.3), and at most $\mathcal{O}(\mathbf{M}^2\mathbf{J}^2)$ cycle elimination constraints corresponding to Eqn (4.2).

### 4.7.2 Conflict Bipartite graph representation of the JIP

As Gröflin and Klinkert [2007] have shown, the **JIP** can be conveniently represented as a bipartite graph. Following standard convention, let $G_c^{\mathcal{J}} = (U_1, U_2, E)$ denote a bipartite graph, where $U_1, U_2$ are vertex sets and $E$ is an edge set. Each edge in $E$ connects a vertex in $U_1$ to a vertex in $U_2$. Slightly abusing notation, we define $U_1 = \{v_{a_1} | \langle a_1, a_2\rangle \in \mathcal{A}_{\mathcal{J}}\}$, and $U_2 = \{v_{a_2} | \langle a_1, a_2\rangle \in \mathcal{A}_{\mathcal{J}}\}$. So $U_1$ contains vertices corresponding to all alternative arcs whose head lies in $\mathcal{J}$, while $U_2$ contains vertices corresponding to all alternative arcs whose tail lies in $\mathcal{J}$. The edges in $E$ represent all pair-wise conflicts between alternative arcs. By pair-wise conflicts, we mean that $E$ contains the edge $(v_{g_1}, v_{h_2})$, if including both alternative arcs $g_1$ and $h_2$ in $\bar{S}$ will result in $\bar{S}$ being inconsistent. Notice that Eqns (4.2), (4.3) precisely prohibits all pair-wise edge conflicts from occurring in $\bar{S}$. So, corresponding to each constraint in Eqns (4.2) and (4.3), $E$ contains an edge between the appropriate pair of vertices. Formally:

$$\begin{aligned} E = &\{(v_{a_1}, v_{a_2}) | \langle a_1, a_2\rangle \in \mathcal{A}_{\mathcal{J}}\} \\ &\cup \{(v_{g_1}, v_{h_2}) | \langle g_1, g_2\rangle, \langle h_1, h_2\rangle \in \mathcal{A}_{\mathcal{J}} \text{ s.t.} \langle g_1, g_2\rangle, \langle h_1, h_2\rangle \text{ are pair-wise distinct and (I), (II)} \\ &\qquad\qquad \text{are satisfied.}\} \end{aligned}$$

It was shown in Gröflin and Klinkert [2007] that there is a 1:1 correspondence between maximum stable sets of $G_c^{\mathcal{J}}$ and vertices of **JIP**. Any maximum stable set of $G_c^{\mathcal{J}}$ has cardinality $|\mathcal{A}_{\mathcal{J}}| = \mathbf{M}(\mathbf{J}-1)$. Since $G_c^{\mathcal{J}}$ was constructed using selection $S_{\mathcal{J}}$, for convenience, we also refer to $G_c^{\mathcal{J}}$ as the conflict graph for $S_{\mathcal{J}}$.

### 4.7.3 Performing Algorithm 8 using the conflict graph

Given the above background, we can make the Job Insertion algorithm provided in Algorithm 8 more precise by modifying it to employ the conflict graph. In addition to the inputs of Algorithm 8, we assume that the conflict graph $G_c^{\mathcal{J}}(U_1, U_2, E)$ for $S_{\mathcal{J}}$ is also provided as input; the output (as before) will be a feasible selection $\bar{S}$ s.t. $S_{\mathcal{J}} \cup \{a\} \subset \bar{S}$. The resulting algorithm is a temporal propagation procedure coupled with SMCP, where the propagation portion is reminiscent of the closure procedure from Bürgy [2017].

We use the following notations for describing Algorithm 9. For vertex $v \in U_1 \cup U_2$, let $\Delta_{G_c^{\mathcal{J}}}(v)$ denote the set of all vertices in $G_c^{\mathcal{J}}$ connected to $v$ by an edge in $E$. Define a function $V(\cdot)$ which takes as input an alternative arc belonging to some alternative arc pair in

**Algorithm 9** Job Insertion With Conflict Graph

1: **Given:** Job $\mathcal{J}$ to be inserted, consistent selection $S_{\mathcal{J}}$ with all arc pairs in $\mathcal{A}_{\mathcal{J}}$ *unselected*, and conflict graph $G_c^{\mathcal{J}}$. Arc $a$ to be included.
2: **Output:** A complete consistent selection $\bar{S}$
3: **Initialize:** Stack $K \leftarrow \{V(a)\}$, sets $Q, \bar{Q} \leftarrow \emptyset$, selection $\bar{S} \leftarrow S_{\mathcal{J}}$.
4: **repeat**
5: **repeat**
6: $w \leftarrow K.pop()$
7: **if** $w \in Q$ **then**
8: **continue**
9: $Q.insert(w)$
10: $\bar{S} \leftarrow \bar{S} \cup V^{-1}(w)$
11: **for all** $u \in \Delta_{G_c^{\mathcal{J}}}(w)$ **do**
12: **if** $u \notin \bar{Q}$ **then**
13: $\bar{Q}.insert(u)$ { $u$ excluded from the stable set since an edge connecting $u$ and $w$ exists in $G_c^{\mathcal{J}}$.}
14: $\bar{u} \leftarrow V(Pa(V^{-1}(u)))$
15: **if** $\bar{u} \notin Q$ **then**
16: $K.push(\bar{u})$ {Since $V^{-1}(u)$ will be excluded from $\bar{S}$, $V^{-1}(\bar{u})$ must be included in $\bar{S}$ due to Eqn (4.3).}
17: **until** $K$ is empty
18: **if** $|Q| = |\mathcal{A}_{\mathcal{J}}|$ **then**
19: **return** $\bar{S}$
20: **else**
21: Apply SMCP to get the next arc $q$ to include in $\bar{S}$ from *unselected* arc pairs in $\mathcal{A}_{\mathcal{J}}$
22: $K.push(V(q))$
23: **until** No arc pair from $\mathcal{A}_{\mathcal{J}}$ is *unselected* in $\bar{S}$.

$\mathcal{A}_{\mathcal{J}}$, and returns the vertex in $G_c^{\mathcal{J}}$ corresponding to the input alternative arc. We define the inverse function $V^{-1}(\cdot)$ which takes as input a vertex from $U_1 \cup U_2$, and returns the alternative arc corresponding to the vertex. Finally, we introduce a function $Pa(\cdot)$, which takes as input an alternative arc, and returns its paired arc. So if $\langle a_1, a_2 \rangle \in \mathcal{A}_{\mathcal{J}}$, then $Pa(a_1) = a_2$ (resp. $Pa(a_2) = a_1$).

Algorithm 9 constructs the feasible selection $\bar{S}$ by computing a maximum stable set $Q$ on $G_c^{\mathcal{J}}$. The alternative arcs corresponding to vertices in $Q$ are included in the selection $\bar{S}$. Algorithm 9 consists of 2 nested loops, where the inner loop (lines 5 - 17) essentially performs the inference step mentioned earlier in Algorithm 8. In this inference step, the effect of adding a vertex $w$ into the stable set is propagated to infer other vertices that must also be included in (or excluded from) the stable set as a consequence. SMCP is applied to obtain the next arc $q$ to include in $\bar{S}$ on each iteration of the outer loop, and the overall process terminates when a maximum stable set of $G_c^{\mathcal{J}}$ is found (line 18).

### 4.7.4 Complexity of executing Algorithm 9

We can decompose the execution of Algorithm 9 into an inference by propagation step (inner loop) and the SMCP step (line 21). We first analyze the complexity of the propagation step. In this step, the vertices that need to be included (or excluded) in the stable set are discovered by examining edges in $E$ through lines 11 - 16. Algorithm 9 examines each edge in $E$ at most twice, and so the complexity of the propagation step for the entire execution of Algorithm 9 is bounded by $\mathcal{O}(|E|)$. A single execution of the SMCP step in line 21 requires $\mathcal{O}(\mathbf{MJ})$ computations, and so if the outer loop in Algorithm 9 executes $r$ times, the overall run time of Algorithm 9 is bounded by $\mathcal{O}(|E|) + r\,\mathcal{O}(\mathbf{MJ})$. While $\mathbf{M}(\mathbf{J}-1)$ is a trivial upper bound for $r$, empirically we observe that $r$ is often significantly smaller than $\mathbf{M}(\mathbf{J}-1)$ for SMCP.

Algorithm 9 requires the conflict graph as an input to the algorithm. Obtaining all the edges in the conflict graph is expensive, since for every pair of alternative pairs $\langle g_1, g_2\rangle, \langle h_1, h_2\rangle \in \mathcal{A}_{\mathcal{J}}$ satisfying condition (I), we need to check if condition (II) is also true. Note that we need to verify this condition for $\mathcal{O}(\mathbf{M}^2\mathbf{J}^2)$ pairs. Checking condition (II) for each pair can be done using simple path-finding algorithms like depth-first search, or through the use of the Floyd-Warshall algorithm. The complexity of constructing the conflict graph with such procedures is $\mathcal{O}(\mathbf{M}^3\mathbf{J}^3)$.

### 4.7.5 Improving the complexity of job insertion

From Section 4.7.4 we can generally conclude that the complexity of performing job insertion using an expensive procedure to construct a dense conflict graph is detrimental to the performance of local search, since these conflict graphs need to be constructed at every iteration of the search, and in the worst case up to $\mathbf{J}$ conflict graphs (one for each job) at any given iteration may need to be constructed. In the following section, we show that using at most $\mathcal{O}(\mathbf{M}^2\mathbf{J})$ cycle elimination constraints along with the constraints in Eqn (4.3) suffices for representing the **JIP**. Note that by reducing the number of cycle elimination constraints needed to represent **JIP**, we can prune the edges in the conflict graph. In sections 4.7.6 and 4.7.7 we enumerate the pruned set of cycle elimination constraints and also provide a method to generate the pruned set of constraints in $\mathcal{O}(\mathbf{M}^2\mathbf{J})$ complexity. With these improvements, the complexity of performing Job Insertion with Algorithm 9 on the sparser conflict graph is upper bounded by $\mathcal{O}(\mathbf{M}^2\mathbf{J}) + r\,\mathcal{O}(\mathbf{MJ})$, where $r$ recall is the number of times the outer loop in Algorithm 9 is executed.

Before moving on to the proposed improvements, we briefly explain how our work advances the current state of the art. In Gröflin and Klinkert [2007], the authors implicitly recognized that some constraints in Eqn (4.2) were redundant, so they only included the irredundant cycle elimination constraints in their description of the **JIP**. However, they do not specify the irredundant set from the constraints shown in Eqn (4.2), nor do they provide an efficient method for generating these constraints. Our work proposes an efficient procedure for obtaining the **JIP**.

Algorithm 9 can also be used to obtain the output of the closure procedure [Gröflin and Klinkert, 2009]. We restrict Algorithm 9 to one complete execution of the inner loop. For all *unselected* arc pairs remaining after the previous step, we retain the same selection that

is present in $S$ for each of those arc pairs. This scheme has a complexity of $\mathcal{O}(\mathbf{M}^2\mathbf{J})$, since $r = 1$. Bürgy [2017] conceptually describes closure on a conflict graph, but a method for constructing the conflict graph is not mentioned. Our work provides an efficient implementation for closure in the context of BJS.

Finally, the current state-of-the-art results for the BJS problem are due to Dabah *et al.* [2017, 2019], and their approach for feasibility recovery is similar to JIFR-1. A partial selection is iteratively expanded, and every time a new selection is made, it is followed by a check to determine if there exists an *unselected* arc pair for which a selection cannot be made without leading to inconsistency. A complexity analysis for their procedure is not provided, but they remark that it is expensive. In comparison, the empirical results in Section 4.9.3 suggests that our procedure is cheaper.

### 4.7.6 Pruning redundant cycle elimination constraints

Let $\bar{O}_{\mathcal{J}}$ denote the set of nodes in $\bar{O}$ other than those that belong to job $\mathcal{J}$. Given $S_{\mathcal{J}}$, some topological order $T_{S_{\mathcal{J}}}$ for nodes in $G(F \cup S_{\mathcal{J}})$, we define $f_o^m$ for each $o \in \bar{O}_{\mathcal{J}}$ and each machine $m$ as shown below in Eqn (4.5).

$$f_o^m = \begin{cases} \theta, & \text{if no path to } o \text{ from any node in } \bar{O}_{\mathcal{J}} \text{ requiring machine } m \text{ exists in } G(F \cup S_{\mathcal{J}}) \\ \left\{ \arg\max_u \; T_{S_{\mathcal{J}}}^{-1}(u) \; {}^{\text{s.t. } u \in \bar{O}_{\mathcal{J}} \cap O,\, M(u)=m \text{ and a path}}_{\text{from } u \text{ to } o \text{ exists in } G(F \cup S_{\mathcal{J}})} \right\}, & \text{otherwise} \end{cases} \tag{4.5}$$

**Definition 1.** *For $o \in \bar{O}_{\mathcal{J}}$, $H_o = \{f_o^m | m \in \{1, 2, \ldots, \mathbf{M}\}\}$ denotes the predecessor map for $o$ in $S_{\mathcal{J}}$.*

The predecessor map of $o$ provides a compact representation of the set of other nodes whose start times can be no later than the start time of $o$ in any feasible schedule obeying the same relative ordering of operations on machines as also implied in $S_{\mathcal{J}}$.

**Definition 2.** *$\{H_o\}_{o \in \bar{O}_{\mathcal{J}}}$ will be referred to as the predecessor buffer for $S_{\mathcal{J}}$, where $H_o$ is as defined in Definition 1.*

Assuming we have computed $f_o^m$ for all nodes in $\bar{O}_{\mathcal{J}}$, we can specify the pruned set of cycle elimination constraints by making use of conditions (I) and (II) from Section 4.7.1. The first set of constraints in the pruned set are:

$$\begin{aligned} x_{g_1} + x_{h_2} \leq 1, \forall \text{ pairwise distinct } \langle g_1, g_2 \rangle, \langle h_1, h_2 \rangle \in \mathcal{A}_{\mathcal{J}} \text{ s.t. condition (I) is satisfied} \\ \text{and } f_{tail(g_1)}^{M(head(h_2))} = head(h_2) \end{aligned} \tag{4.6}$$

Note that if $f_{tail(g_1)}^{M(head(h_2))} = head(h_2)$, then condition (II) is satisfied by $\langle g_1, g_2 \rangle, \langle h_1, h_2 \rangle$, which automatically implies validity of Eqn (4.6) as a cycle elimination constraint. There are no more than $(\mathbf{J}-1)\frac{\mathbf{M}(\mathbf{M}+3)}{2}$ constraints corresponding to Eqn (4.6). To see why, note there are exactly $\mathbf{M}(\mathbf{J}-1)$ candidates for $g_1$. For any given $g_1$, there are at most $\mathbf{M} + 1 - pos(head(g_1))$ candidates for $h_2$ satisfying the conditions in Eqn (4.6).

We move on to the second set of constraints in the pruned set. For any $o \in O \cap \bar{O}_{\mathcal{J}}$ (i.e., all dummy nodes and job $\mathcal{J}'s$ nodes are excluded), let $u(o) \in O$ denote the node belonging to job $\mathcal{J}$ which requires the same machine as $o$, i.e., $J(u(o)) = \mathcal{J}$ and $M(u(o)) = M(o)$. Consider the arcs $g(o) = (\gamma(u(o)), o)$, and $h(o) = \big(\gamma(u(o)), \delta_{S_{\mathcal{J}}}(o)\big)$. $g(o)$ is an alternative arc to node $o$, and $h(o)$ is an alternative arc to the machine successor of $o$ according to $S_{\mathcal{J}}$. $\delta_{S_{\mathcal{J}}}(o)$ in the strictest sense is an abuse of notation since $S_{\mathcal{J}}$ is not a complete selection, nonetheless $S_{\mathcal{J}}$ defines a complete consistent selection for the system of jobs other than $\mathcal{J}$, hence $\delta_{S_{\mathcal{J}}}(o)$ can be computed relative to those jobs from $S_{\mathcal{J}}$. The other constraints in the pruned set are:

$$x_{g(o)} \leq x_{h(o)} \; \forall o \in O \cap \bar{O}_{\mathcal{J}} \tag{4.7}$$

Equation (4.7) is equivalent to the logical implication $(x_{g(o)} \implies x_{h(o)})$. The validity of Eqn (4.7) follows from the fact that any complete consistent selection containing $S_{\mathcal{J}}$ and arc $g(o)$ must also contain $h(o)$, since if $u(o)$ is serviced before $o$, it also implies that $u(o)$ is serviced before $\delta_{S_{\mathcal{J}}}(o)$. There are $\mathbf{M(J-2)}$ constraints corresponding to Eqn (4.7).

**Proposition 3.** *Every constraint in Eqn (4.2) is implied by the set of constraints in Eqns (4.6) and (4.7).*

*Proof.* Let $\langle g_1, g_2\rangle, \langle h_1, h_2\rangle \in \mathcal{A}_{\mathcal{J}}$ denote a pair of distinct alternative arc pairs corresponding to which there is a cycle elimination constraint in Eqn (4.2), and let $w = f_{tail(g_1)}^{M(head(h_2))}$. If $head(h_2) = w$, then there is already an identical constraint in the system of inequalities specified in Eqn (4.6), and so there is nothing to prove in this case. Recall that $M(head(h_2)) = M(w)$, so if $head(h_2) \overset{T_{S_{\mathcal{J}}}}{\succ} w$, then the existence of a path from $head(h_2)$ to $tail(g_1)$ in $G(F \cup S_{\mathcal{J}})$ trivially contradicts the definition of $w$. So we can assume that $w \overset{T_{S_{\mathcal{J}}}}{\succ} head(h_2)$. We know that the constraint $x_{g_1} + x_{(tail(h_2),w)} \leq 1$ is specified in the system of inequalities corresponding to Eqn (4.6). Since $w \overset{T_{S_{\mathcal{J}}}}{\succ} head(h_2)$ and $M(w) = M(head(h_2))$, by transitivity on the system of inequalities in (4.7), we know that the constraint $x_{h_2} \leq x_{(tail(h_2),w)}$ is implied. Adding the previous two constraints gives us the cycle elimination constraint for the pair $\langle g_1, g_2\rangle, \langle h_1, h_2\rangle$. □

As a consequence of Proposition 3, an alternate representation for **JIP** is shown in Eqn (4.8). For constructing the conflict graph for $S_{\mathcal{J}}$ based on the **JIP** representation in Eqn (4.8), we can reuse the procedure from Section 4.7.2. In order to do so, we need to convert Eqn (4.7) into an inequality of the form as shown in Eqn (4.6). Notice that we could have alternatively written Eqn (4.7) as $x_{g(o)} + x_{Pa(h(o))} \leq 1$, where $Pa(\cdot)$ returns the paired alternative arc as defined in Section 4.7.3. Finally, observe that the conflict graph $G_c^{\mathcal{J}}$ corresponding to the **JIP** in Eqn (4.8) is sparser than the version described in Section 4.7.2, since the **JIP** in Eqn (4.8) is described by roughly a factor of **J** fewer constraints.

$$\mathbf{conv}(x \in \{0,1\}^{2|\mathcal{A}_{\mathcal{J}}|} | x \text{ satisfies Eqns (4.3), (4.6) and (4.7)}) \tag{4.8}$$

### 4.7.7 Computation of $f_o^m$

Algorithm 10 computes $f_o^m$ for all $o \in \bar{O}_{\mathcal{J}}$, based on the following idea. Let the selection $S_{min}$ denote the minimal representation of $S_{\mathcal{J}}$. Then, observe that we could have replaced $S_{\mathcal{J}}$ in Eqn (4.5) by $S_{min}$, since if there exists a path between some pair of nodes in $G(F \cup S_{\mathcal{J}})$, then the same pair of nodes are connected by a path in $G(F \cup S_{min})$ as well. Given any node $v \in \bar{O}_{\mathcal{J}}$, notice that any path to $v$ in $G(F \cup S_{min})$ passes through either $\alpha(v)$ or $\gamma(\beta_{S_{\mathcal{J}}}(v))$. So suppose we have $f^m_{\alpha(v)}$ and $f^m_{\gamma(\beta_{S_{\mathcal{J}}}(v))}$, then obtaining $f_v^m$ can be done as shown in lines 7 - 10. Algorithm 10 traverses the nodes in $\bar{O}_{\mathcal{J}}$ in some topological order $T_{S_{\mathcal{J}}}$ (say) of $G(F \cup S_{\mathcal{J}})$, and computes the desired $f_{\cdot}^{\cdot}$ for each node by comparing with the $f_{\cdot}^{\cdot}$ values of the node's predecessors. The complexity of computing $T_{S_{\mathcal{J}}}$ is $\mathcal{O}(\mathbf{MJ})$, and the complexity of executing Algorithm 10 is $\mathcal{O}(\mathbf{M^2J})$, since for each node in $\bar{O}_{\mathcal{J}}$ we make $2\mathrm{M}$ comparisons (lines 7 - 10).

**Algorithm 10** Computing $f_o^m$

1: **Input:** $G(F \cup S_{\mathcal{J}}), T_{S_{\mathcal{J}}}$
2: $f_o^m = \begin{cases} o & \text{if } o \in O \text{ and } m = M(o) \\ \theta & \text{otherwise} \end{cases}, \forall o \in \bar{O}_{\mathcal{J}}$
3: **for** $i \leftarrow 1 : |T_{S_{\mathcal{J}}}|$ **do**
4: $\quad v \leftarrow T_{S_{\mathcal{J}}}[i]$ {$v$ is the node in position $i$ of $T_{S_{\mathcal{J}}}$}
5: $\quad$ **if** $J(v) = \mathcal{J}$ **then**
6: $\quad\quad$ **continue**
7: $\quad$ **for all** $w \in \{\alpha(v), \gamma(\beta_{S_{\mathcal{J}}}(v))\}$ **do**
8: $\quad\quad$ **for all** $m \in \{1, 2, \ldots, \mathrm{M}\}$ **do**
9: $\quad\quad\quad$ **if** $f_w^m \overset{T_{S_{\mathcal{J}}}}{\succ} f_v^m$ **then**
10: $\quad\quad\quad\quad f_v^m \leftarrow f_w^m$

## 4.8 Tabu search metaheuristic implementation

We implemented our local search procedure using tabu search, a popular metaheuristic choice for JS and BJS problems. The tabu search implementation is described in Algorithm 11. A tabu list of fixed length $L$ stores attributes of solutions that occurred in the previous $L$ iterations. The search discourages revisiting solutions by checking if they contain attributes that are stored in the tabu list. The algorithm chooses the neighbor with the lowest cost in the neighborhood as the solution for the next iteration. Following Bürgy [2017], the cost of a neighbor is computed as the sum of the neighbor's makespan plus a correction term. The correction term is set to 0 if the neighbor does not contain an attribute present in the list, else it is set to $k\,B$, where $k$ is the earliest position in the tabu list of an attribute present in the neighbor, and $B$ is a large constant to discourage tabu solutions being selected over non-tabu neighbors. The progress of the search is monitored, and if for some user defined $maxNonImprov$ iterations no new best solution is found, the search is restarted with a new seed.

**Algorithm 11** Tabu Search Framework

```
1: Given:
2: Initial feasible solution with selection S.
3: Initial tabu list TL.
4: best iteration ← 0, iteration ← 0,
5: best solution ← makespan(S).
6: repeat
7:    Set S' to be the best neighbor of S
8:    Update tabu list with attributes of S'
9:    if makespan(S') < best solution then
10:      best solution ← makespan(S')
11:      best iteration ← iteration
12:   else
13:      if iteration - best iteration > maxNonImprov then
14:         return best solution
15:      else if objectives cycling then
16:         return best solution
17:   S ← S'
18:   iteration ← iteration + 1
19: until Termination Criterion
20: return best solution
```

With respect to maintaining and updating the tabu list (line 8), the basic idea is to treat a move as a sequence of machine swap operations and add an entry into the tabu list corresponding to each swap. This idea was adopted from Dell'Amico and Trubian [1993], and we explain our implementation with the help of an example. Let $S$ be the selection corresponding to the solution of the current iteration, and let operations $o_1, o_2, o_3$ form a critical block on $cp(S)$, where $o_2 = \delta_S(o_1)$ and $o_3 = \delta_S(o_2)$. Assume the best solution in the neighborhood is obtained by the *N4* forward move for $S$, which makes operation $o_1$ the machine successor of operation $o_3$. The tabu list is updated by deleting the oldest entry from the list, and inserting the set $H = \{(o_1, o_2), (o_1, o_3)\}$ at the end of the list. Symmetrically, to check if a neighbor contains a tabu attribute, we check whether a swap from the sequence of swaps associated to the *N4* move that generated the neighbor is present in the tabu list. If present, such a neighbor is considered to be tabu by the list.

Tabu search suffers from the drawback of cycling, i.e., repeatedly encountering the same sequence of solutions during search. To detect cycles, we employ the long term memory structures previously used in Nowicki and Smutnicki [1996]. A necessary condition to detect cycling is the following. Denote $Vec$ to be a vector of nonnegative reals, where $Vec[i]$ is set equal to the makespan value of the solution at iteration $i$. Assume $Vec[i] = Vec[i-a] = Vec[i-2a] = \ldots = Vec[i-ka]$, where $a, k$ are some positive numbers. We characterize this sequence of solutions as cycling with period $a$ and having completed $k$ cycles. If $k$ exceeds a certain pre-defined constant $maxCycle$, tabu search is restarted from a new seed/elite solution (see next paragraph).

Another common practice to enhance tabu search performance is to restart the search process by keeping track of elite solutions [Nowicki and Smutnicki, 1996]. Elite solutions are previously encountered promising makespan neighbors that were skipped over for some other solution in the neighborhood for the subsequent iteration. When a restart is performed, the most promising elite solution recorded is chosen as the starting solution, and the state of the search (i.e., tabu list) is restored to what it was when the elite solution was first encountered. Our implementation of elite solutions was adopted from Bürgy [2017].

## 4.9 Evaluation

### 4.9.1 Neighborhoods considered

Note that we can replace *N4* with *N5*, and still apply all the techniques presented in Sections 4.5 - 4.7, since *N4* subsumes *N5*. Briefly, the moves in *N5* are a restricted version of the *N4* moves. In our description of the forward move in Section 4.4, *N5* restricts $w$ to just the machine predecessor of $u_n$ if the neighbor is being generated from a type-1 critical block. Similarly, for the backward move, *N5* restricts $w$ to just the machine successor of $u_1$. For the solution shown in Example 8, the *N4* and *N5* neighbors coincide.

In Section 4.9.3, we report results with both *N5* and *N4* neighborhoods. Although *N4* subsumes *N5*, there are several important pragmatic considerations when choosing a neighborhood for practical applications. Given the fact that constructing a single neighbor with Job Insertion is very expensive, one may wonder if choosing a neighborhood that is smaller would perform better within practical computational time budgets. With a smaller neighborhood, the search can obviously transition between many more solutions within a given time limit. Also, since search using smaller neighborhoods are more likely to get stuck earlier in local minima, the search will be restarted more often. Restarts can help diversify the search, thus improving the overall performance.

### 4.9.2 Experimental setup

All experiments in this section were carried out on an Intel i7-4790 3.6 GHz processor. Our code is implemented in C++ and runs on one CPU thread. For all experiments we set $maxCycle = 3$, $B = 100$, and $maxNonImprov = 200$. The tabu length parameter was set based on instance size; details are given in Section 4.9.3.

For every infeasible solution generated by a *N4* move, 2 feasible solutions can be recovered with JIFR-1 since there are 2 choices for $\mathcal{J}$ in line 3 of Algorithm 6. However, recalling the discussion in Section 4.6.3, more than 2 feasible solutions can be recovered from a given infeasible solution when using JIFR-2. Including both feasibility recovery procedures often results in a very large neighborhood, and our experimental design was influenced by this observation. For all experiments reported in this chapter, JIFR-1 was applied to recover 2 feasible solutions for every infeasible solution generated during the search. JIFR-2, in contrast, was used more selectively. JIFR-2 was only used in a subset of the experiments performed and for those that included it, JIFR-2 was applied in addition to JIFR-1 only when a new best

solution has not been found for $0.7 * maxNonImprov$ consecutive search iterations. In this case, we continued applying JIFR-2 until either a new best solution was found or a restart occurred. Intuitively, one can think of our implementation as expanding the search space dynamically to avoid getting stuck in a sub-optimal region.

Finally, as mentioned in Section 4.8, whenever a restart occurs during tabu search, the starting solution is either an elite solution previously discovered during search, or a seed solution is randomly generated. Restart from a randomly generated seed solution occurs iff no unexplored elite solution remains at that stage. A seed solution is randomly generated in two steps. First a random permutation of jobs is generated. Then the solution (selection) is obtained from the permutation order as follows: Suppose job $\mathcal{J}$ occurs earlier than $\bar{\mathcal{J}}$ in the permutation order, then for every machine $m \in \{1, 2, \ldots, \mathrm{M}\}$, the operation in $\mathcal{J}$ that requires $m$ is serviced earlier by machine $m$ than the operation in $\bar{\mathcal{J}}$ requiring $m$. The seed solutions generated typically are of very poor quality in terms of makespan.

### 4.9.3 Experimental Results

Currently, best known solutions for the BJS problem with no swap are only available for instances from the Lawrence benchmark. In Table 4.1 we report the results (makespan) we obtained on these benchmark instances, and compare them with previous best known results. For the experiments with *N5* neighborhood reported under Table 4.1, only JIFR-1 was used for feasibility recovery, while for experiments using *N4* both JIFR-1 and JIFR-2 were used, where JIFR-2 was applied selectively as described in Section 4.9.2. Note that the two chosen configurations represent extremes in terms of neighborhood sizes, where the former is the smallest and the latter is the largest among the set of possibilities considered in this work. Following common practice for reporting results in BJS literature (see [Bürgy, 2017], [Gröflin *et al.*, 2011]), we ran our tabu search approach 10 times for each instance in the dataset, and recorded the best solution for each of those 10 runs after 1, 5, 10, 20 and 30 minutes. We report the best solution obtained among those 10 runs under the Best columns in Table 4.1. We also report the average of the best solutions across the 10 runs under the Avg columns. We report results after 5 and 20 minutes in Table 6 (see Appendix B.5). The tabu length for experiments reported under Table 4.1 was set to 10. The solution with the best makespan across the 10 runs for each instance can be accessed from `https://github.com/jmogali/Solution_Data/tree/master/Results_Data`.

Most of the previous best known results on the Lawrence instances are due to Dabah *et al.* [2019]. Out of the 40 instances, we report new best results on 28 instances. On 11 instances, our best result matches with the previous best known result, and on only one instance, i.e., LA14, the previous best solution is better than the one we obtained with either neighborhood. Note that we have obtained better solutions on most of the larger instances, i.e., instances having more than 10 jobs or machines. Relative to the previous best known results, the average improvement we have obtained is 2.95% for the $15 \times 10$ instances, 5.37% for the $20 \times 10$ instances, 4% for the $30 \times 10$ instances, and 2.35% for the $15 \times 15$ instances. The setting under which the experiments were carried out in Dabah *et al.* [2019] is different to ours. Some previous best results due to Dabah *et al.* [2019] in Table 4.1 were produced using a parallel tabu search with 240 concurrent CPU threads on a supercomputer. Unfortunately, the average

**Table 4.1:** Performance comparison against previous best known on Lawrence instances using *N5*, and *N4* neighborhoods. Numbers in bold font indicate the makespan of the best solution for that instance. Underlined instances are those for which a new best solution is reported by an experiment conducted in this work.

| Inst. | Size | Previous best known | N5 with JIFR-1 | | | | | | N4 with JIFR-1 & 2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 60 sec | | 600 sec | | 1800 sec | | 60 sec | | 600 sec | | 1800 sec | |
| | | | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg |
| LA01 | 10 × 5 | **881**[a,b,c] | **881** | 881 | 881 | 881 | 881 | 881 | **881** | 881 | 881 | 881 | 881 | 881 |
| LA02 | 10 × 5 | **900**[a,b,c] | **900** | 900 | 900 | 900 | 900 | 900 | **900** | 900 | 900 | 900 | 900 | 900 |
| LA03 | 10 × 5 | **808**[a,b,c] | **808** | 808 | 808 | 808 | 808 | 808 | **808** | 808 | 808 | 808 | 808 | 808 |
| LA04 | 10 × 5 | **859**[a,b,c] | **859** | 859 | 859 | 859 | 859 | 859 | **859** | 859 | 859 | 859 | 859 | 859 |
| LA05 | 10 × 5 | **732**[a,b,c] | **732** | 732 | 732 | 732 | 732 | 732 | **732** | 732 | 732 | 732 | 732 | 732 |
| LA06 | 15 × 5 | 1199[c] | 1195 | 1203.2 | **1194** | 1195.2 | 1194 | 1194.4 | **1194** | 1205.1 | 1194 | 1195 | 1194 | 1194.2 |
| LA07 | 15 × 5 | 1130[c] | **1127** | 1132.1 | 1127 | 1127.2 | 1127 | 1127 | **1127** | 1132.4 | 1127 | 1127 | 1127 | 1127 |
| LA08 | 15 × 5 | **1173**[c] | **1173** | 1190.2 | 1173 | 1175.3 | 1173 | 1173 | **1173** | 1189.4 | 1173 | 1173.8 | 1173 | 1173 |
| LA09 | 15 × 5 | 1311[b,c] | **1305** | 1312.3 | 1305 | 1305 | 1305 | 1305 | **1305** | 1316 | 1305 | 1305.5 | 1305 | 1305.1 |
| LA10 | 15 × 5 | 1222[c] | 1222 | 1226.9 | **1218** | 1221.2 | 1218 | 1220 | **1218** | 1228.3 | 1218 | 1220.8 | 1218 | 1220 |
| LA11 | 20 × 5 | 1591[c] | 1563 | 1588.2 | 1522 | 1557.3 | **1501** | 1538.7 | 1558 | 1596 | 1522 | 1557.3 | 1522 | 1543.6 |
| LA12 | 20 × 5 | 1399[c] | 1388 | 1414.1 | **1353** | 1382.1 | 1353 | 1375.9 | 1397 | 1417.6 | **1353** | 1391.1 | 1353 | 1383.7 |
| LA13 | 20 × 5 | 1538[c] | 1523 | 1545.6 | **1508** | 1531.6 | 1508 | 1518.7 | 1554 | 1569.5 | **1508** | 1527.7 | 1508 | 1523.4 |
| LA14 | 20 × 5 | **1544**[c] | 1582 | 1602.5 | 1568 | 1586 | 1557 | 1571.6 | 1575 | 1615.2 | 1548 | 1574 | 1548 | 1564.9 |
| LA15 | 20 × 5 | 1616[c] | **1565** | 1621.8 | 1565 | 1591.5 | 1565 | 1586.4 | 1605 | 1626.8 | 1587 | 1596.4 | 1578 | 1591.2 |
| LA16 | 10 × 10 | **1148**[a,b,c] | **1148** | 1148.6 | 1148 | 1148 | 1148 | 1148 | **1148** | 1148.6 | 1148 | 1148 | 1148 | 1148 |
| LA17 | 10 × 10 | **968**[a,b,c] | **968** | 968 | 968 | 968 | 968 | 968 | **968** | 969.9 | 968 | 968 | 968 | 968 |
| LA18 | 10 × 10 | **1077**[a,b,c] | **1077** | 1082.4 | 1077 | 1077 | 1077 | 1077 | **1077** | 1090.9 | 1077 | 1077 | 1077 | 1077 |
| LA19 | 10 × 10 | **1102**[a,b,c] | **1102** | 1110.5 | 1102 | 1102 | 1102 | 1102 | **1102** | 1114.1 | 1102 | 1102.4 | 1102 | 1102 |
| LA20 | 10 × 10 | **1118**[a,b,c] | **1118** | 1118 | 1118 | 1118 | 1118 | 1118 | **1118** | 1123.6 | 1118 | 1118 | 1118 | 1118 |
| LA21 | 15 × 10 | 1501[c] | 1518 | 1556.6 | **1483** | 1514.2 | 1483 | 1499 | 1508 | 1547.8 | **1483** | 1516.7 | 1483 | 1505 |
| LA22 | 15 × 10 | 1368[c] | **1328** | 1376.2 | 1328 | 1355.1 | 1328 | 1341 | **1328** | 1399.3 | 1328 | 1363.2 | 1328 | 1342.9 |
| LA23 | 15 × 10 | 1534[c] | **1475** | 1525.2 | 1475 | 1505.6 | 1475 | 1483.9 | 1524 | 1564.9 | **1475** | 1513.1 | 1475 | 1497.4 |
| LA24 | 15 × 10 | 1447[c] | 1439 | 1482.8 | **1402** | 1441.2 | 1402 | 1408.1 | **1402** | 1466.2 | 1402 | 1432.5 | 1402 | 1419.4 |
| LA25 | 15 × 10 | 1453[c] | 1445 | 1466.7 | 1409 | 1420.1 | 1409 | 1418.3 | 1422 | 1465.9 | **1406** | 1428.9 | 1406 | 1420.9 |
| LA26 | 20 × 10 | 1968[c] | 1928 | 1980.8 | 1906 | 1957 | 1885 | 1928.2 | 1920 | 2002.1 | **1870** | 1929.6 | 1870 | 1920.3 |
| LA27 | 20 × 10 | 2047[c] | 2006 | 2064.8 | 1957 | 2025 | **1933** | 2007.4 | 2041 | 2092.5 | 1992 | 2045.2 | 1987 | 2028 |
| LA28 | 20 × 10 | 2046[c] | 1993 | 2016.7 | 1973 | 1997.2 | **1937** | 1972.8 | 1972 | 2051.1 | 1972 | 2016.5 | 1971 | 1989.8 |
| LA29 | 20 × 10 | 1857[c] | 1844 | 1898.3 | 1819 | 1862.8 | **1764** | 1825.9 | 1819 | 1898 | 1808 | 1869.6 | 1808 | 1850.3 |
| LA30 | 20 × 10 | 2033[c] | 1944 | 2024.6 | 1944 | 1991 | **1939** | 1965.3 | 2005 | 2050.6 | 1953 | 2008.6 | 1953 | 1987.8 |
| LA31 | 30 × 10 | 2866[c] | 2760 | 2842.8 | 2746 | 2784.5 | 2746 | 2780.8 | 2815 | 2893.5 | 2760 | 2824.7 | **2714** | 2811.6 |
| LA32 | 30 × 10 | 3040[c] | 2994 | 3106.6 | 2982 | 3038.2 | 2982 | 3029.2 | 3004 | 3124.5 | 2962 | 3010.9 | **2928** | 2995.2 |
| LA33 | 30 × 10 | 2803[c] | 2738 | 2843.1 | **2717** | 2783.2 | 2717 | 2770.6 | 2779 | 2866.2 | 2732 | 2768.7 | 2732 | 2765.7 |
| LA34 | 30 × 10 | 2862[c] | 2843 | 2905.7 | **2769** | 2831.9 | 2769 | 2831.9 | 2838 | 2937.7 | 2812 | 2858.6 | 2812 | 2855.6 |
| LA35 | 30 × 10 | 2871[c] | 2880 | 2938.4 | 2762 | 2850.4 | 2762 | 2849.8 | 2889 | 2953.8 | 2806 | 2886.8 | **2757** | 2855.2 |
| LA36 | 15 × 15 | 1757[c] | 1733 | 1804.3 | 1697 | 1763.5 | **1683** | 1731.3 | 1733 | 1804.5 | 1701 | 1762.3 | 1700 | 1742.8 |
| LA37 | 15 × 15 | 1871[c] | 1887 | 1929.4 | 1862 | 1903.8 | **1856** | 1879.7 | 1881 | 1943.5 | 1881 | 1912.8 | 1867 | 1907.9 |
| LA38 | 15 × 15 | 1716[c] | 1675 | 1734.5 | **1665** | 1725.2 | 1665 | 1697.4 | 1726 | 1761.9 | 1692 | 1726.8 | 1670 | 1707.6 |
| LA39 | 15 × 15 | 1750[c] | 1750 | 1792.2 | 1728 | 1758.5 | 1728 | 1745.2 | 1725 | 1782.7 | **1720** | 1749.8 | 1720 | 1741.1 |
| LA40 | 15 × 15 | 1742[c] | 1737 | 1785.4 | **1712** | 1755.5 | 1712 | 1743.3 | 1766 | 1798.4 | 1747 | 1769.7 | 1735 | 1760.9 |

a = Mati et al. [Mati and Xie, 2011], b = Pranzo and Pacciarelli [Pranzo and Pacciarelli, 2016], c = Dabah et al. [Dabah *et al.*, 2019]

| Instance Size | # Iterations for N5 with JIFR-1 | # Iterations for N4 with JIFR -1 & 2 |
|---|---|---|
| 10×5 | 2186983 | 1831003 |
| 15×5 | 898528.5 | 738437.6 |
| 20×5 | 495716.7 | 375890.1 |
| 10×10 | 716766.8 | 385476.4 |
| 15×10 | 265798.2 | 138097.8 |
| 20×10 | 150163.7 | 71599.6 |
| 30×10 | 63237.1 | 28826.6 |
| 15×15 | 142547.4 | 58196.1 |

**Table 4.2:** Average # of tabu search iterations within 30 minutes for experiments conducted corresponding to Table 4.1.

| Instance size | # Iterations |
|---|---|
| 15 × 10 | 277012.1 |
| 20 × 10 | 179871.5 |
| 30 × 10 | 62709.3 |
| 15 × 15 | 145515.9 |

**Table 4.3:** The number of tabu search iterations on LA instances in 30 minutes averaged across 10 runs with *N4* neighborhood and JIFR-1.

time it takes to obtain solutions is not provided in Dabah *et al.* [2019]. We computed the average of the best solution makespan obtained using our approach after 10 minutes across the 10 runs, and found that this value was strictly better than the previous best known solution on 21 out of the 40 instances for both *N5* and *N4* neighborhoods.

The best solution found using *N4* was better than the best solution using *N5* on 7 instances, whereas the best solution found using *N5* was better than the best solution using *N4* on 12 instances. In terms of the best solution makespan averages, the results were better with *N5* on 21 instances, while the results were better using *N4* on only 6 instances. The number of tabu search iterations within 30 minutes averaged across the 10 runs for each neighborhood is provided in Table 4.2. On at least 4 out of the 5 instances for each of the problem sizes: $20 \times 5$, $15 \times 10$, $20 \times 10$ and $15 \times 5$, we observed that the best solution makespan average after 30 minutes with *N5* is better than with *N4*. For those problems sizes, we can observe that we are able to roughly perform twice the number of tabu search iterations with *N5* as compared to *N4*, which perhaps explains the strong performance with *N5*.

### Gauging the computational speedup

The main contributing factors to the strong performance of our approach over prior approaches are the efficiency improvements we proposed, namely for feasibility detection of *N4*/*N5* neighbors, makespan computations and the Job Insertion procedure. We observed that generally between 40-45% of the *N4* neighbors generated were feasible, thus allowing us to skip invoking any feasibility recovery procedure for those neighbors. None of the papers using Job Insertion for feasibility recovery include details on the implementation of the recovery procedure, so making a quantitative statement about improvement in complexity is not possible. For a sense of the speed-up achieved with our approach, we compare the number of tabu search iterations completed within a given time limit using other approaches.

Among the papers that have reported results for BJS with no swap, only Dabah *et al.* [2017, 2019] use Job Insertion for feasibility recovery. All experiments in Dabah *et al.* [2017] were performed with the *N1* neighborhood, and the procedure they applied for feasibility recovery is described in Section 4.7.5. Note that their feasibility recovery procedure is very similar in

spirit to JIFR-1. They performed 100,000 Tabu search iterations on the $10 \times 5$, $15 \times 5$, $20 \times 5$, $10 \times 10$ Lawrence (LA) instances and 50,000 iterations for the larger LA instances (on a 2GHz CPU). While the average time to complete those iterations was not provided in Dabah *et al.* [2017], the authors remark that it takes roughly 5 minutes for small instances and between 2-3 hours for the larger instances. In Table 2 of Dabah *et al.* [2019], the same group of authors as Dabah *et al.* [2017] provide the average time it takes to complete 1000 iterations of Tabu search for the $30 \times 10$ LA instances with *N1* neighborhood. The average timings they report are no less than 1000 seconds for all instances.

To understand how much speed up has been gained with our efficiency improvements, we performed the following experiment. We recorded the average number of tabu search iterations we were able to perform within 30 minutes on the LA instances using *N4* neighborhood and JIFR-1. We choose *N4* for comparison because the number of neighbors in the *N4* neighborhood is (roughly) twice the size of *N1* neighborhood, so the number of times the Job Insertion procedure is invoked during search is roughly proportional for the two approaches. The average number of tabu search iterations within 30 minutes across 10 runs for the larger LA instances is reported in Table. 4.3.

**Impact of JIFR-2 on search performance**

The inclusion of JIFR-2 helps to increase the neighborhood size, but it may lead to fewer restarts thus limiting search diversity, or the additional computational cost may limit the overall exploration of the search space. We conducted an experiment on the larger Lawrence instances LA21- LA40, where we studied the impact of including JIFR-2 with *N4* and *N5*. We conducted 5 independent runs of 20 minutes on each of those LA instances for both neighborhoods with and without JIFR-2, and recorded the makespan of the best solution obtained in each run. For the experiments involving the *N5* (resp. *N4*) neighborhood with and without JIFR-2, we ensured that the randomly generated seeds were identical, to remove any bias caused by these seed solutions. We summarize the results below.

We observed that with the *N5* neighborhood, the solution with the lowest makespan across the 5 runs was obtained with JIFR-2 included on 8 out of the 20 instances, and without JIFR-2 on 7 out of the 20 instances. However, the value of the best solution makespan after 20 minutes averaged across the 5 runs was better without JIFR-2 on 15 out of the 20 instances. For the *N4* neighborhood, similar trends were observed. Best results with JIFR-2 included were produced on 8 out of the 20 instances, and without JIFR-2 on 5 out of the 20 instances. The best result averages across the 5 runs was better without JIFR-2 on 11 out of the 20 instances.

While the results in the previous paragraph demonstrate the utility of expanding the neighborhood with JIFR-2 in achieving better best results, the average best results tell a different story. It is not all that frequent that JIFR-2 is able to guide the search to more promising regions during search. The problem can be attributed to the fact that including JIFR-2 results in a very large neighborhood. This observation suggests that alternative schemes that enumerate fewer neighbors generated with JIFR-2 may be beneficial in balancing the trade-off between solution quality and an enlarged neighborhood. Such schemes are an interesting direction for future research.

**Table 4.4:** Best and average result after 60 seconds with *N4* and JIFR-1 & 2 compared against results reported in a = Mati and Xie [2011], b = Pranzo and Pacciarelli [2016]

| Instance | Previous best result from a, b | Best (60 sec) | Avg (60 sec) |
|---|---|---|---|
| LA01 | 881[a, b] | 881 | 881 |
| LA02 | 900[a, b] | 900 | 900 |
| LA03 | 808[a, b] | 808 | 808 |
| LA04 | 859[b] | 859 | 859 |
| LA05 | 732[b] | 732 | 732 |
| LA06 | 1243[a] | 1194 | 1205.1 |
| LA07 | 1143[b] | 1127 | 1132.4 |
| LA08 | 1213[b] | 1173 | 1189.4 |
| LA09 | 1311[b] | 1305 | 1316 |
| LA10 | 1237[b] | 1218 | 1228.3 |
| LA11 | 1683[b] | 1558 | 1596 |
| LA12 | 1479[a] | 1397 | 1417.6 |
| LA13 | 1642[b] | 1554 | 1569.5 |
| LA14 | 1686[b] | 1575 | 1615.2 |
| LA15 | 1682[b] | 1605 | 1626.8 |
| LA16 | 1148[b] | 1148 | 1148.6 |
| LA17 | 968[b] | 968 | 969.9 |
| LA18 | 1077[b] | 1077 | 1090.9 |
| LA19 | 1124[b] | 1102 | 1114.1 |
| LA20 | 1164[b] | 1118 | 1123.6 |
| LA21 | 1627[b] | 1508 | 1547.8 |
| LA22 | 1435[b] | 1328 | 1399.3 |
| LA23 | 1574[b] | 1524 | 1564.9 |
| LA24 | 1530[b] | 1402 | 1466.2 |
| LA25 | 1558[b] | 1422 | 1465.9 |
| LA26 | 2159[b] | 1920 | 2002.1 |
| LA27 | 2191[b] | 2041 | 2092.5 |
| LA28 | 2319[b] | 1972 | 2051.1 |
| LA29 | 2054[b] | 1819 | 1898 |
| LA30 | 2263[b] | 2005 | 2050.6 |
| LA31 | 3403[a] | 2815 | 2893.5 |
| LA32 | 3576[b] | 3004 | 3124.5 |
| LA33 | 3255[b] | 2779 | 2866.2 |
| LA34 | 3306[b] | 2838 | 2937.7 |
| LA35 | 3373[a] | 2889 | 2953.8 |
| LA36 | 1835[b] | 1733 | 1804.5 |
| LA37 | 1931[b] | 1881 | 1943.5 |
| LA38 | 1813[b] | 1726 | 1761.9 |
| LA39 | 1811[b] | 1725 | 1782.7 |
| LA40 | 1815[b] | 1766 | 1798.4 |

### Search performance under low computation time regimes

Many papers describe BJS applications where the computation time available for applying the BJS heuristic is short. In such situations, it may be preferable to design the BJS heuristic as a form of VLNS (very large neighborhood search) approach as opposed to a tabu search approach, since tabu search approaches may spend an unreasonable proportion of computational time in evaluating all the neighbors of each solution. Mati and Xie [2011], Pranzo and Pacciarelli [2016] have pursued constructive VLNS approaches and reported results with short computation times.

We compare the results under the Best and Avg column under 60 seconds reported in Table 4.1 for the *N4* neighborhood with JIFR - 1 & 2 against the results provided in Mati and Xie [2011], Pranzo and Pacciarelli [2016]. Such a comparison helps determine if our approach is competitive against existing VLNS approaches for low computation time regimes. The comparison is provided in Table 4.4. The results in Mati and Xie [2011] were obtained after performing 10 independent runs of 200 seconds on each instance, while results corresponding to Pranzo and Pacciarelli [2016] were obtained from 10 independent runs of 60 seconds each. On 30 out of the 40 LA instances, the value under the Avg column is better than the value under the previous best column. To get a sense of the relative improvement with our approach, for each row in Table 4.4 we computed the quantity $(\text{Previous best result - Best (60 sec)}) * 100/\text{Previous best result}$. For the $20 \times 10$ instances the relative improvement is $11.14\%$, for the $30 \times 10$ instances it is $15.28\%$, and for the $15 \times 15$ instances it is $4.07\%$.

### First results for Taillard instances

Based on the observations drawn from experiments in Section 4.9.3, we conclude that the best performing configuration in terms of obtaining best results on the Lawrence instances was the *N5* neighborhood along with JIFR - 1 & 2. Using the same configuration, we ran our heuristic 5 times on all Taillard instances. The results on the largest Taillard instances are shown in Table 4.5. The reader can find results for the remaining Taillard instances in Table

**Table 4.5:** Results on the largest Taillard instances

| Instance | Size | 60 sec | | 300 sec | | 600 sec | | 1200 sec | | 1800 sec | | Avg # Iter |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg | (1800 sec) |
| TA71 | $100 \times 20$ | 16560 | 17426.6 | 13372 | 13485.4 | 12744 | 12882.6 | 12231 | 12568.4 | 12153 | 12369.4 | 2275.6 |
| TA72 | $100 \times 20$ | 15445 | 16225.8 | 12901 | 13076.4 | 12182 | 12387.2 | 11688 | 11907.4 | 11444 | 11745.6 | 2437.8 |
| TA73 | $100 \times 20$ | 16444 | 17370.4 | 13375 | 13533.4 | 12602 | 12792.8 | 11991 | 12269.4 | 11766 | 12078.6 | 2391.6 |
| TA74 | $100 \times 20$ | 16266 | 16963.6 | 13354 | 13526.4 | 12702 | 12791.6 | 12038 | 12285.2 | 11897 | 12044.8 | 2569.8 |
| TA75 | $100 \times 20$ | 16189 | 17127.6 | 13288 | 13497.6 | 12451 | 12729.8 | 12002 | 12181.8 | 11476 | 11911.4 | 2287.6 |
| TA76 | $100 \times 20$ | 15532 | 16578 | 13081 | 13447.8 | 12543 | 12865.8 | 12067 | 12388.2 | 12067 | 12223.8 | 2352 |
| TA77 | $100 \times 20$ | 16369 | 17674.8 | 13436 | 13640.4 | 12813 | 13037.6 | 12457 | 12543.8 | 12265 | 12412.2 | 2383.4 |
| TA78 | $100 \times 20$ | 16649 | 17007.8 | 13198 | 13506.4 | 12596 | 12756 | 11793 | 12074.6 | 11697 | 11898.6 | 2329.8 |
| TA79 | $100 \times 20$ | 15834 | 17145.8 | 13243 | 13439 | 12508 | 12820.6 | 11999 | 12281 | 11870 | 12118.4 | 2236.6 |
| TA80 | $100 \times 20$ | 15227 | 16186.4 | 12640 | 12943.6 | 12042 | 12310.2 | 11654 | 11869.8 | 11405 | 11729 | 2253.4 |

B.10 , see Appendix B.5. The tabu length parameter for the smaller Taillard instances TA01 - TA40 was set to 10, and 15 for the remaining instances. The problem dimensions in the Taillard dataset are significantly larger than those in the Lawrence dataset, and to the best of our knowledge the results shown in Table B.10 are the first results for the Taillard instances. The solution corresponding to the best makespan for each instance can be accessed from `https://github.com/jmogali/Solution_Data/tree/master/Results_Data`. In the last column of Tables 4.5 and B.10 we report the number of tabu search iterations within 30 minutes averaged across the 5 runs on each instance.

## 4.10 Structural characterization of feasible schedules and implications for local search

Despite the impressive results in the previous section, we believe there remains room for further improvement. Specifically, there are structural characteristics unique to the BJS problem that, to our knowledge, have not heretofore been acknowledged that offer additional opportunities for improving BJS solving efficiency. In this section, we identify these structural properties and suggest how they can be exploited. We believe this to be one interesting direction for future research.

We begin with a simple observation: in any feasible schedule for the BJS problem, at any instant in time, job $\mathcal{J}$ can be in exactly one of the following 3 states:

**STATE 1:** The first operation of $\mathcal{J}$ has not yet been serviced.

**STATE 2:** All the operations in $\mathcal{J}$ have been serviced.

**STATE 3:** $\mathcal{J}$ is currently occupying (blocking) some machine.

By contrast, for the JS problem, in addition to the three states mentioned above there is a fourth state, where only some operations of $\mathcal{J}$ have been serviced and job $\mathcal{J}$ is currently waiting in a buffer area, i.e., job $\mathcal{J}$ is in-progress but not blocking any machine from being accessed by the other jobs.

This distinction has interesting consequences with respect to the structure of feasible solutions. Consider a BJS problem instance where $\mathbf{J} \gg \mathbf{M}$, and let $Sch$ be a feasible schedule for the problem instance. Let $St_{\mathcal{J}}(Sch)$ denote the start time of the first operation of job $\mathcal{J}$ in $Sch$, and let $Co_{\mathcal{J}}(Sch)$ denote the completion time of the last operation of job $\mathcal{J}$ in $Sch$. Then $[St_{\mathcal{J}}(Sch), Co_{\mathcal{J}}(Sch)]$ represents the interval in which job $\mathcal{J}$ occupies some machine (i.e., state 3 in the description above), which we refer to as job $\mathcal{J}$'s service interval in schedule $Sch$. Theorem 5 shows that the service interval of a given job $\mathcal{J}$'s in $Sch$ will overlap with the service intervals of a bounded subset of other jobs in $Sch$, where two job service intervals are considered to be overlapping iff the length of the overlap is greater than zero. To prove Theorem 5, we make use of Lemma 5, and both proofs are provided in Appendix B.4. The bound stated in Lemma 5 (by extension the bound established in Theorem 5) can be shown to be tight, an example revealing tightness is shown in Appendix B.4. Note that in the absence of blocking (i.e., JS problem), the service interval of a job can overlap with the service interval of all other jobs, so for the JS problem the corresponding bound in Theorem 5 is $\mathbf{J} - 1$, where the service interval of a job is the time interval in which the job is in either state 3 or 4 in this case.

**Lemma 5.** *Given job $\mathcal{J}$ and a feasible schedule $Sch$ to the BJS problem instance, let $Q$ denote the set of all jobs that have a service interval strictly contained within the service interval of $\mathcal{J}$, then $|Q| \leq (\mathbf{M} - 1)^2$.*

**Theorem 5.** *A job's service interval can overlap with the service interval of at most $\mathbf{M}^2 - 1$ other jobs.*

Let us denote the set of all jobs in our problem by $Jo$. For feasible selection $S$, the following corollary reveals some structural insight into the graph $G(F \cup S)$.

**Corollary 1.** *For a job $\mathcal{J}$, define a set $W_{\mathcal{J}}^S = \{\bar{\mathcal{J}} | \bar{\mathcal{J}} \in Jo \backslash \mathcal{J}$ s.t. there exist alternative arcs $(u_1, v_1), (u_2, v_2)$ in $S$ where $u_1, v_2$ belong to $\mathcal{J}$ and $v_1, u_2$ belong to $\bar{\mathcal{J}}\}$. Intuitively, $W_{\mathcal{J}}^S$ is the set of jobs in which, for each $\bar{\mathcal{J}} \in W_{\mathcal{J}}^S$ we can identify a pair of machines $m_1, m_2$ s.t. $\mathcal{J}$ is serviced earlier than $\bar{\mathcal{J}}$ by $m_1$, and $\bar{\mathcal{J}}$ is serviced earlier than $\mathcal{J}$ by $m_2$. In any feasible schedule obeying the sequence of operations on machines implied by $S$, the service interval of a job in $W_{\mathcal{J}}^S$ overlaps with the service interval of $\mathcal{J}$, so it follows from Theorem 5 that $|W_{\mathcal{J}}^S| \leq \mathbf{M}^2 - 1$.*

### 4.10.1 Streamlining job insertion

The above structural properties can be exploited in at least a couple of ways to further improve the efficiency of BJS solving procedures. First, they provide constraints for reducing the cost of job insertion for problems with large job to machine ratios. Notice that any job $\bar{\mathcal{J}} \in Jo \backslash \{W_{\mathcal{J}}^S \cup \mathcal{J}\}$ (i.e., the complement of $W_{\mathcal{J}}^S$), falls under one of the following categories:

1. For all $m \in \{1, 2, \ldots, \mathbf{M}\}$, the operation belonging to $\bar{\mathcal{J}}$ that requires $m$ is serviced earlier than the operation belonging to $\mathcal{J}$ which also requires $m$. Denote the set of jobs under this category by $W_{\mathcal{J}}^{S-}$.

2. For all $m \in \{1, 2, \ldots, \mathbf{M}\}$, the operation belonging to $\bar{\mathcal{J}}$ that requires $m$ is serviced later than the operation belonging to $\mathcal{J}$ which also requires $m$. Denote the set of jobs under this category by $W_{\mathcal{J}}^{S+}$.

One way to interpret the definitions of $W_{\mathcal{J}}^{S}, W_{\mathcal{J}}^{S-}, W_{\mathcal{J}}^{S+}$ is that the service interval of job $\mathcal{J}$ in selection $S$ is sandwiched between the service intervals of jobs in the sets $W_{\mathcal{J}}^{S-}$ and $W_{\mathcal{J}}^{S+}$. Suppose the sets $W_{\mathcal{J}}^{S-}$, $W_{\mathcal{J}}^{S+}$ were known for a given input job $\mathcal{J}$ before applying Algorithm 9 to obtain a feasible selection $S$. Then Algorithm 9 could be restricted to only consider insertion options involving jobs in $W_{\mathcal{J}}^{S}$. The modification to Algorithm 9 is to simply pre-initialize the sets $Q, \bar{Q}$ (refer Algorithm 9 for definition) to account for the information provided in $W_{\mathcal{J}}^{S-}, W_{\mathcal{J}}^{S+}$. With this modification, the dependence on $\mathbf{J}$ in the complexity of Algorithm 9 gets replaced by the size of the set $W_{\mathcal{J}}^{S}$, and recall that $|W_{\mathcal{J}}^{S}| \leq \mathbf{M}^2 - 1$ (by Corollary 1).

Of course, we do not know $W_{\mathcal{J}}^{S-}$ or $W_{\mathcal{J}}^{S+}$ in advance, but instead we can approximate the set $W_{\mathcal{J}}^{S}$ in advance and modify Algorithm 9 to use this approximation in an analogous manner. In any feasible schedule obeying the ordering of operations on machines implied by $S$, observe that the jobs in the set $W_{\mathcal{J}}^{S}$ will be temporally close to each other since their respective service intervals overlap with the service interval of job $\mathcal{J}$. Given this, a natural idea for choosing $W_{\mathcal{J}}^{S}$ is to select at most $\mathbf{M}^2 - 1$ jobs that are temporally close to each other in any schedule computed using $G(F \cup S_{\mathcal{J}})$ for the jobs in the set $Jo \backslash \mathcal{J}$, where $S_{\mathcal{J}}$ is the selection obtained by removing all arc pairs associated to job $\mathcal{J}$ from $S$.

### 4.10.2 Incrementally computing the conflict graph when using JIFR-1 during local search

The insights gained from Theorem 5 can also be used to reduce the costs associated with the construction of a conflict graph during local search. Recall that the job insertion procedure from Section 4.7 is a 2-step procedure. The first step involves computing the predecessor buffer for populating edges in the conflict graph, and the second is to execute Algorithm 9 with the resulting conflict graph. Although the second step cost may be larger than the first step by a factor no larger than $\mathbf{J}$, the complexity of both steps is the same if we configure Algorithm 9 to execute "closure" by omitting the SMCP component as described earlier in Section 4.7.5. When $\mathbf{M}, \mathbf{J}$ are very large, it may be more attractive to omit SMCP and perform local search with just closure, since it is inherently cheaper. In such situations, optimizing the first step becomes an important consideration.

To this end, assume $S, \bar{S}$ are feasible selections where $\bar{S}$ is obtained by applying JIFR-1, and so $\bar{S}$ is the output of Algorithm 9 with $S_{\mathcal{J}}$ and job $\mathcal{J}$ as inputs. For enumerating neighbors of $\bar{S}$ using JIFR-1, depending on the jobs associated with the operations in the *N4* (resp. *N5*) moves, we may need to compute several conflict graphs, one for each job. For some job $\bar{\mathcal{J}}$, we use an implication of Theorem 5 to develop an incremental procedure for constructing the conflict graph for $\bar{S}_{\bar{\mathcal{J}}}$ (i.e., selection obtained by removing all arc pairs associated to job $\bar{\mathcal{J}}$ from $\bar{S}$) by maximally reusing edges from the conflict graph computed for $S_{\bar{\mathcal{J}}}$, if that conflict graph was computed earlier.

For convenience, let us denote the edges in the conflict graph for $S_{\bar{\mathcal{J}}}$ (resp. $\bar{S}_{\bar{\mathcal{J}}}$) by $E$ (resp. $\bar{E}$). If $\mathcal{J} = \bar{\mathcal{J}}$, note that $E$ and $\bar{E}$ are identical since $\bar{S}$ was obtained by inserting $\mathcal{J}$, and so we will assume that $\mathcal{J} \neq \bar{\mathcal{J}}$. Recall from Section 4.7.2 that any edge in a conflict graph can be mapped to one of the Eqns (4.3), (4.6), or (4.7). Most of the differences in the edges between $E$ and $\bar{E}$ are those that correspond to Eqn (4.6). Let $Ol$ denote the predecessor buffer for $S_{\bar{\mathcal{J}}}$, and let $Ne$ denote the predecessor buffer for $\bar{S}_{\bar{\mathcal{J}}}$. Observe the direct correspondence between the entries in the predecessor buffer and edges corresponding to Eqn (4.6). Consequently, by analyzing changes between $Ol$ and $Ne$, we can complete our understanding of how the edges corresponding to Eqn (4.6) in $E$ and $\bar{E}$ differ.

We can intuitively think of $\bar{S}_{\bar{\mathcal{J}}}$ as being obtained from $S_{\bar{\mathcal{J}}}$ by a 2-step procedure. In the first step, we remove the alternative arcs associated to job $\mathcal{J}$ from $S_{\bar{\mathcal{J}}}$, and denote the intermediate selection as $S_{im}$. To $S_{im}$, we then add just those alternative arcs present in $\bar{S}$ which are associated to job $\mathcal{J}$ but not also with $\bar{\mathcal{J}}$ to obtain $\bar{S}_{\bar{\mathcal{J}}}$. The observation suggests a 2-step procedure for computing $Ne$ from $Ol$. In the first step, we compute the predecessor set buffer $Im$ for $S_{im}$ by incrementally modifying the entries of $Ol$. In the second step, we incrementally modify $Im$ to obtain $Ne$. Next we discuss the computation of $Im$ from $Ol$, computing $Ne$ from $Im$ is analogous.

In the buffer $Ol$, operations belonging to job $\mathcal{J}$ can appear in the predecessor maps for nodes belonging to other jobs, denote the set of those other jobs by $Q$. For nodes belonging to jobs other than those in $Q$, the predecessor maps will be identical in both $Ol$ and $Im$, a property we can exploit. Using Theorem 5, we prove in Proposition 11 (shown in Appendix B.4) that $|Q| \leq \mathbf{M}^2 + \mathbf{M} - 1$. Suppose we compute a schedule for jobs in $Jo \backslash \bar{\mathcal{J}}$ using the same ordering of operations on machines as implied by $S_{\bar{\mathcal{J}}}$, the proof of Proposition 11 characterizes a job in $Q$ as either one whose service interval overlaps with the service interval of $\mathcal{J}$, or is among the earliest set of jobs that occupies a machine once job $\mathcal{J}$ vacates all machines. Given the temporal proximity of the service intervals of jobs in $Q$ with the service interval of job $\mathcal{J}$, the key observation is that the service intervals of jobs in $Q$ are mutually close to each other. Consequently, nodes belonging to jobs in $Q$ won't be spread far apart in any topological ordering $T_{S_{\bar{\mathcal{J}}}}$ of $G(F \cup S_{\bar{\mathcal{J}}})$. For obtaining $Im$ using Algorithm 10 with $G(F \cup S_{im})$ and $T_{S_{\bar{\mathcal{J}}}}$ as inputs, $Ol$ takes the role of $f$ in Algorithm 10. We then simply update the entries in $Ol$ by restricting the loop counter $i$ in line 3 to take values between the smallest and largest topological index (TI) corresponding to nodes belonging to jobs in $Q$. Using the bound on $|Q|$, we can expect the range of $i$ that gets updated to be $\approx \mathcal{O}(\mathbf{M}^3)$, as there are $\mathbf{M} + 1$ nodes in $G(F \cup S_{im})$ for every job.

We now turn our attention to computing $Ne$ from, $Im$, and overall strategy will be very similar to computing $Im$ from $Ol$. Recall that the selection $\bar{S}_{\bar{\mathcal{J}}}$ is obtained by inserting job $\mathcal{J}$ into $S_{im}$ . From Proposition 11, we know that operations belonging to job $\mathcal{J}$ can appear in the predecessor maps for nodes belonging to at most $\mathbf{M}^2 + \mathbf{M} - 1$ jobs other than $\mathcal{J}$ in $Ne$, and once again note that these jobs will not be spread very far apart in any topological ordering $T_{\bar{S}_{\bar{\mathcal{J}}}}$ for nodes in the graph $G(F \cup \bar{S}_{\bar{\mathcal{J}}})$. Denote those $\mathbf{M}^2 + \mathbf{M} - 1$ set of jobs by $R$. For nodes belonging to jobs other than those in $R \cup \mathcal{J}$, the predecessor maps will be identical in both $Im$ and $Ne$. The set $R$ however is unknown in advance, unlike $Q$. To use the previous strategy of incrementally updating the data in the predecessor buffer, we need a way to determine a range enclosing the topological indices spanned by nodes belonging to

jobs in $R \cup \mathcal{J}$. We can estimate this range dynamically. We know that the node corresponding to the first TI in $T_{\bar{S}_{\bar{\mathcal{J}}}}$ for which the predecessor maps differs between $Im$ and $Ne$ is the TI corresponding to the first node belonging to job $\mathcal{J}$, denote that TI by $first$. We also know that the last TI (denoted by $last$) for which the maps differ is no lesser than the TI of the last node belonging to job $\mathcal{J}$ in $T_{\bar{S}_{\bar{\mathcal{J}}}}$. We can dynamically update the estimate for $last$ based on the following natural idea. Suppose the predecessor map for node $v$ is different in $Im$ and $Ne$, then since $\gamma(v), \delta_{\bar{S}_{\bar{\mathcal{J}}}}(v)$ are the outgoing neighbors of $v$, the predecessor maps for those pair of nodes may also be different in $Im$ and $Ne$. So our estimate of $last$ will be updated to $max(last, T^{-1}_{\bar{S}_{\bar{\mathcal{J}}}}(\gamma(v))), T^{-1}_{\bar{S}_{\bar{\mathcal{J}}}}\left(\delta_{\bar{S}_{\bar{\mathcal{J}}}}(v)\right)$. We can convert this idea into an algorithm by updating the entries in $Im$ to get the correct values for $Ne$. This is accomplished by executing Algorithm 10 between the range $[first, last]$ (for variable $i$ in line 3), where $last$ is updated dynamically as execution progresses. At some stage $last$ will no longer be updated. Based on the bound on $|R|$, $last - first$ will be $\approx \mathcal{O}(\mathbf{M}^3)$.

Overall this approach that we have outlined for computing $Ne$ from $Ol$ is more efficient than executing Algorithm 10 on all $\mathbf{M}(\mathbf{J}+1)$ indices in line 3. Although we leave quantification of the benefits of exploiting these structural characteristics of the BJS for future research, we do not expect a significant increase in number of iterations in our experiments since the SMCP cost component in Algorithm 9 may far exceed the cost of constructing a conflict graph.

## 4.11 Summary and discussion

In this work, we presented an efficient local search heuristic to obtain feasible solutions for the Blocking Job Shop (BJS) problem with no swap. The local search heuristic is based on the popular *N4* neighborhood for the Job Shop (JS) problem, and is embedded within a tabu search framework. One important distinction in the use of the *N4* neighborhood for solving the BJS problem (as opposed to the JS problem) is that the constraints associated with BJS lead to generation of substantial numbers of infeasible solutions, and mechanisms are needed for recovering feasible solutions from these generated starting points for good search performance. An efficient procedure to quickly identify infeasible solutions was presented. Two different methods for recovering a feasible solution from an infeasible one were described, and one of those two methods is novel. Both methods employ a job insertion mechanism from the literature, and an efficient algorithm for realizing this mechanism was proposed. Critical to the performance of this job insertion procedure was an efficient computational procedure that was presented for obtaining the job insertion *polytope*. To demonstrate the efficacy of the approach, an experimental analysis was performed using existing benchmark problem instances provided by Lawrence and Taillard. On the Lawrence instances, we obtained new best results on 28 out of the 40 instances. Our approach matches the previous best result on 11 out of the remaining 12 instances. To the best of our knowledge, we are the first to report results on the Taillard instances. Improvements in speed and solution quality with our method over previously published work were analyzed, and given that one of the two methods employed in this work for converting an infeasible neighbor into a feasible solution has not been studied before, a short analysis to understand its impact was included. Finally, for instances with more jobs than machines, a structural property obeyed by all solutions to the BJS problem was

identified. Based on the insight gained from the structural property, ways to improve the local search implementation were outlined. A precise description of the contributions made in this chapter are listed in Section 4.2.2.

Although this work has focused on the BJS no-swap case, the techniques presented here can be adapted for broader applicability. For example, the procedure presented for determining feasibility of neighbors can be easily adapted for applicability to the JS problem (i.e., job shop problem without blocking). Our algorithm for job insertion in Section 4.7 works with the alternative graph representation of a given solution, and relies on the existence of a topological ordering of the nodes in the alternative graph. In the case of BJS problem with swap, a topological ordering may not exist, since the corresponding alternative graph may contain zero length cycles. However, the presence of zero length cycles can be handled by compressing every maximal zero length cycle into a single node, and producing a new graph that does admit a topological ordering. On this new graph, one can then directly apply all job insertion techniques presented in Section 4.7 to the BJS problem with swap. Finally, we believe that the techniques presented in Section 4.7 for obtaining the Job insertion polytope in $\mathcal{O}(\mathbf{M}^2\mathbf{J})$ complexity can be extended with minor modifications so that the polytope based approach can be used to perform job insertion for other complex JS variants (e.g. flexible BJS problem [Gröflin *et al.*, 2011]) efficiently.

## 4.12 Summary of contributions

- Presented an efficient algorithm for identifying feasible *N4* neighbors. Presented an efficient algorithm to compute the makespan of the feasible *N4* neighbors. Our algorithm can be easily adapted to the JS problem.
- Presented an efficient algorithm for obtaining the H-representation of the Job insertion polytope in $\mathcal{O}(\mathbf{M}^2\mathbf{J})$ complexity. We demonstrated how our compact representation can be used to perform job insertion efficiently. Our algorithm can be easily extended to other variants of the JS problem.
- We presented a novel Job insertion based feasibility recovery procedure, see JIFR-2.
- When $\mathbf{J} > \mathbf{M}$, we identified new structural properties that all feasible schedules to the BJS problem needs to satisfy. Based on the insight gained from the structural property, ways to improve the local search implementation were outlined.

# Part III

# Chapter 5

# Scheduling for Multi-Robot Routing with Blocking And Enabling Constraints

Many multi-robot planning and scheduling applications can be abstracted as a problem of assigning a set of located tasks to a set of homogeneous robots, and developing task processing itineraries for each robot so that the overall time required to accomplish all tasks is minimized. Such applications include multi-robot search and rescue in tight spaces, multi-robot painting of large surfaces, and the application that motivates this work - a multi-robot system used by a major aerospace manufacturer for attaching the skin to the fuselage while building an aircraft. In all of these contexts, individual tasks can be specified as visits to specific locations for specific periods of time (e.g., a hole into which a fastener must be inserted, a spatial location that must be surveyed for surveyors). Generating optimal robot tours for a given set of task assignments is itself NP-Hard, as the problem is equivalent to a multiple Traveling Salesman problem (mTSP). However, the problems of interest here are complicated by additional factors: (1) enabling constraints between tasks that restrict when specific locations can be visited (e.g., at least one adjacent hole must be fastened prior to fastening a given hole), and (2) collision constraints that limit movements of robots contending for the same space.

We refer to this general class of problems as Multi-Robot Scheduling with Blocking and Enabling constraints (MRSBE). The MRSBE problem combines aspects of previously studied vehicle routing and robot path planning problems. Figure 5.1a shows a simple example, in which two robot arms are positioned behind a structure consisting of roughly 100 task locations to be visited (indicated by the colored circles). Each robot can reach 3 columns of locations out of four, and hence they must collaborate to visit all locations. The problem is to determine location task assignments, routes, and schedules for both robots such that (1) all task locations are visited, (2) all enabling and collision constraints are satisfied, and (3) the time required to complete all tasks (overall makespan) is minimized.

Given the complexity of the MRSBE problem, we make a few simplifying assumptions. We abstract the continuous (geometric) representation of the problem into a full (discrete) representation. We do not explicitly consider the robot motion planning problem as part of the optimization procedure. Instead, we assume that the time taken for a robot to move between a pair of task locations is specified as input to the problem. We describe how collisions are modeled in the next section.

We present a system for obtaining good quality feasible solutions to the MRSBE problem. In particular,

1. We introduce and formalize the MRSBE problem as a new challenge problem for the research community.

2. We show that generation of a feasible solution to MRSBE is NP-complete, as are several relaxations of the full MRSBE problem.

3. We develop a local-search heuristic procedure for quickly obtaining good quality feasible solutions to the MRSBE problem. The heuristic should be regarded as the main technical contribution of this work.

To analyze the performance of our heuristic, we publish a set of benchmark problems, some of which capture the structure of our multi-robot fastening application and others that generalize to other application settings. We provide initial performance results on these problems, and develop lower-bound solutions for comparison. This chapter previously appeared in Mogali *et al.* [2021b].

## 5.1 Problem description

The MRSBE problem consists of a set of $N$ task locations $\mathbf{H} = \{1, 2, \ldots, N\}$ that have to be serviced by a set of $M$ robots $\mathbf{R} = \{1, 2, \ldots, M\}$. Each robot $r \in \mathbf{R}$ starts its tour at a unique starting location $o^r$, services without preemption one or more locations, and finally returns to a unique terminal location $d^r$. To traverse from one location $h_i$ to another location $h_j$, where $h_i, h_j \in \mathbf{H}$, each robot $r \in \mathbf{R}$ follows a unique trajectory $P^r_{ij}$, which is typically a function of the robot's starting and ending pose, constrained by dynamics, kinematics, and its environment. A trajectory can be interpreted as a polyline in a 2 or 3 dimensional space. As is customary in related motion planning research (e.g., see path planning problems on Maklink graphs [Habib and Asama, 1991]), we assume that a trajectory $P^r_{ij}$ for a robot $r \in \mathbf{R}$ is modeled as a directed *path* from vertex $h_i$ to vertex $h_j$ which passes through an ordered sequence of one or more *Intermediate Nodes* (*INs*). The accuracy with which a trajectory $P^r_{ij}$ is represented is determined by the number of *INs* used to model the corresponding path, i.e., increasing the number of *INs* results in a more fine-grained representation, but increases the problem size. Without loss of generality, we assume that an *IN* is unique to a trajectory $P^r_{ij}$, i.e., trajectories for different combinations of $i$, $j$, $r$ do not share *INs*.

The MRSBE problem can be modeled on a simple directed *routing graph* $\mathbf{G}(\mathbf{V}, \mathbf{A})$, where $\mathbf{V} = \mathbf{O} \cup \mathbf{D} \cup \mathbf{H} \cup \mathbf{I}$, $\mathbf{O} = \bigcup_{r \in \mathbf{R}} \{o^r\}$ is the set of starting locations, $\mathbf{D} = \bigcup_{r \in R} \{d^r\}$ is the set of terminal locations, $\mathbf{H}$ is the set of locations that must be serviced by a robot, $\mathbf{I}$ is the set of *INs*, and $\mathbf{A} \subset \mathbf{V} \times \mathbf{V}$ is the set of arcs. Some locations cannot be reached by all robots. $\mathbf{R_i} \subseteq \mathbf{R}$ denotes the subset of robots that can reach location $i \in \mathbf{V}$. Throughout this work, the subscript $_r$ will be used to denote subsets of vertices (or arcs) reachable by a robot $r \in \mathbf{R}$. For example, $\mathbf{H_r}$ denotes the subset of task locations in $\mathbf{H}$ accessible to $r$. Likewise, $\mathbf{G_r}$ is the sub-graph induced in $\mathbf{G}$ by locations $\mathbf{V_r}$ reachable by $r$.

With each vertex $i \in \mathbf{V}$, we associate a positive, robot-dependent processing time $p_i^r$. Robot-dependent traversal times from one vertex to another are not explicitly encoded as cost on the arcs. Instead, traversal times are assumed to be included in the processing times of the vertices: the travel time for a robot $r \in \mathbf{R}$ from $i$ to $j$, where $i, j \in \mathbf{V} \setminus \mathbf{I}$, can simply be incorporated in the processing times of the *INs* encountered along the unique path $P_{ij}^r$.

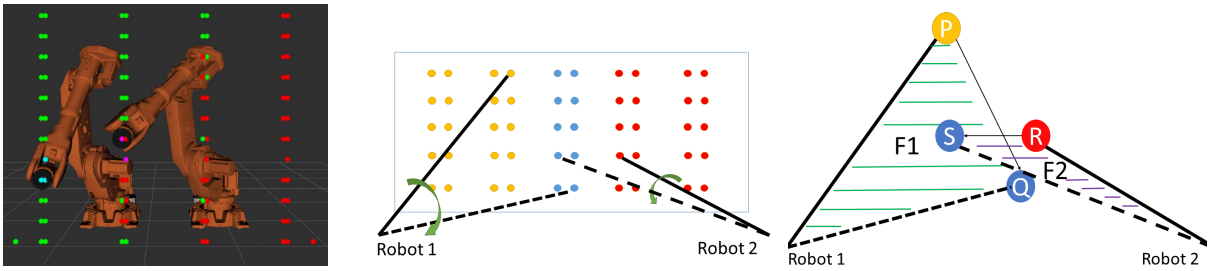
In addition to the *routing constraints* implicitly defined by the routing graph $\mathbf{G}(\mathbf{V}, \mathbf{A})$, the MRSBE problem includes two general types of *scheduling constraints*:

• *Enabling constraints (ECs)* - ECs specify a special type of precedence relation that define a (partial) ordering over the set of locations to be visited. Specifically, for each vertex $i \in \mathbf{H}$, a subset $\boldsymbol{\mathcal{E}}_i \subset \mathbf{H} \cup \mathbf{O}$ of vertices, also referred to as *enablers*, is defined. An EC requires that, prior to servicing a task location $i \in \mathbf{H}$, *at least* one of the task locations in the set $\boldsymbol{\mathcal{E}}_i$ is serviced. In the context of our fastening application, ECs are used to ensure that some holes are bolted (secured) or drilled prior to some other holes.

• *Collision constraints (CCs)* - Collisions occur when a pair of robots try to occupy the same physical space simultaneously. To determine when a collision occurs, for any node $w \in \mathbf{V_r}$, we associate a physical space that is required by the robot $r$ for servicing or waiting at $w$. Through this representation, CCs are expressed as a set of tuples $\mathbf{C}$, where each tuple $(u, r_i, v, r_j) \in \mathbf{C}$ represents an illegal state: robot $r_i$ cannot occupy the space associated with vertex $u \in \mathbf{V_{r_1}}$ when robot $r_j$ occupies the space associated with vertex $v \in \mathbf{V_{r_2}}$. Note that CCs can be expressed for collisions that may arise when robots move between locations, e.g., by expressing CCs between *INs*. We call these collision constraints *blocking* because, when a robot $r$ has processed $w \in \mathbf{V_r}$, and is waiting at $w$ before it can move to a different node in $\mathbf{V_r}$, then so long as $r$ waits idly at $w$, $r$ does not relinquish the space associated with $w$.

A solution to the MRSBE problem consists of (1) an assignment of all locations $i \in \mathbf{H}$ to robots, and (2) a feasible route and corresponding schedule for each robot. The latter requires sequencing the locations assigned to each robot and determining feasible start and end times for each of the servicing operations. Naturally, the schedules must adhere to all routing and scheduling constraints. The goal is to minimize the length of the schedule (makespan).

**Example 11.** *The problem in Figure 5.1 contains 2 robots, task locations are shown as shaded circles. Let task locations $P$ and $Q$ (shown in Figure) be accessible to robot $r_1$, so we have $P, Q \in \mathbf{H_{r_1}}$. For modeling the movement of $r_1$ from $P$ to $Q$, an intermediate node $F_1$ is present in $\mathbf{V_{r_1}}$, and directed arcs $(P \rightarrow F_1)$, $(F_1 \rightarrow Q)$ are present in $\mathbf{A_{r_1}}$. The space traversed by the robot $r_1$, if it were to move from $P$ to $Q$, is indicated by the shaded triangular region. The triangular region is the space we associate with the intermediate node $F_1$. Analogously, let task locations $R$ and $S$ be accessible to the robot $r_2$, so an IN $F_2$ associated with the movement of $r_2$ from $R$ to $S$ is present in $\mathbf{V_{r_2}}$, and arcs $(R \rightarrow F_2)$, $(F_2 \rightarrow S)$ is present in $\mathbf{A_{r_2}}$. As the physical spaces associated with $F_1$ and $F_2$ overlap, a CC $(F_1, r_1, F_2, r_2)$ is present in $\mathbf{C}$. Note that in our model $F_1$ is only accessible to $r_1$, and $F_2$ is only accessible to $r_2$. However, we should not conclude that $P, Q \notin \mathbf{H_{r_2}}$ and $R, S \notin \mathbf{H_{r_1}}$. A different IN would have been specified in $\mathbf{V_{r_2}}$ for moving from $P$ to $Q$ if $P, Q \in \mathbf{H_{r_2}}$. However, note that the physical space associated with this IN for $r_2$ may be different from the space associated to $F_1$,*




**(a)** **(b)** **(c)**

**Figure 5.1:** In Figure 5.1a, robots are at work. In Figure 5.1b, we show an abstraction of Figure 5.1a and one possible movement between task locations for the robots. In Figure 5.1c, we show the movement of $r_1$ from $P$ to $Q$. $F1$ (region shaded in green) is the space needed by $r_1$ for moving from $P$ to $Q$. Similarly, $F2$ is the space needed for $r_2$ to move from $R$ to $S$.

*and so the corresponding collision constraint profiles for those two INs may also be different. Also note from this example, as we need to check collisions for all robot pairs and all possible movements between task locations reachable by the robots, the total number of CCs in the problem can roughly scale as* $\mathcal{O}(M^2N^4)$.

*Continuing with the example, now suppose we are given a set of robot tours in which* $r_1$ *traverses from* $P$ *to* $Q$*, and* $r_2$ *traverses from* $R$ *to* $S$*. Say we are also given a schedule* $Sch$ *for those tours, where the start time of the node* $v \in \mathbf{V}$ *is denoted by* $S_v$*. The service interval for the node* $P$ *is* $[S_P, S_{F_1}]$ *and not,* $[S_P, S_P + p^r_P]$*, i.e., the service duration of a node includes any wait time the robot incurs until it can begin servicing the next node on its tour. The CC between* $F_1, F_2$ *is not violated in* $Sch$ *iff the intervals* $[S_{F_1}, S_Q]$ *and* $[S_{F_2}, S_S]$ *do not overlap.*

We next provide some user guidance for selecting the number of *INs* while modeling the problem. In Example 11, we only included a single *IN* between each pair of task locations. Ideally, selecting the number of *INs* should be based on the specific situation. As a general guideline, it is helpful to think of the trade-off between the impact that the choice of introducing more *INs* will have on the objective being optimized and model complexity. Consider the following two situations, (a) the robot's footprint is small and the time it takes to move between a pair of task locations is large, (b) the robot is moving through a region that one would expect to see a lot of traffic due to other robots also requiring that space or in the vicinity. In both situations, it may be beneficial to include more *INs* between the specific task location pairs, since including more *INs* allows a robot to relinquish physical space it no longer requires to other robots at a finer granularity, which in turn can help reduce wait times for other robots. However, when the processing duration of task locations dominates travel times between task locations, then one would not expect significant improvement in the solution quality by increasing the number of *INs*, as in such a scenario the contribution of the travel times to the objective is relatively small. In this case, increasing the number of *INs* increases problem complexity for little gain.

## 5.2 Problem formulation

The MRSBE problem can be formulated as a constraint programming (CP) model, defined on the routing graph $\mathbf{G}(\mathbf{V}, \mathbf{A})$ introduced previously. The model uses successor variables $next_i$ representing the node that is visited immediately after node $i$ for all $i \in \mathbf{V} \setminus \mathbf{D}$. Assignment variables $rob_i$, $i \in \mathbf{V}$ assign each vertex to a robot. Variables $S_i$, $i \in \mathbf{V}$, record the time at which servicing of the node $i$ starts. To simplify notation, a new parameter $d^r_{ij}$ is introduced: $d^r_{ij} = p^r_i$ if $(i, j) \in A_r$, $d^r_{ij} = 0$ otherwise. The complete CP model is stated in Algorithm 12, where $N^+(i)$ denotes the set of nodes that can be reached from $i$ by an arc $G(\mathbf{V}, \mathbf{A})$ whose tail is $i$.

In this formulation, the objective function (line 6) minimizes the makespan. The *allDifferent* constraint (line 7) is a global constraint that forces every decision variable in a given group to assume pairwise different values [Gervet, 1993]. Here, the *allDifferent* constraint ensures that every location $i \in \mathbf{H}$ is serviced and that proper sequences are formed for each robot. Whenever an *IN* $i \in \mathbf{I}$ is not used in any route, $next_i = i$ will hold, i.e., the node is assigned to itself. If, in contrast, *IN* $i$ is used in some route, then there must exist some other node $j \in \mathbf{V}$ for which $next_j = i$. Due to the *allDifferent* constraint, it follows that $next_i \neq i$, thereby certifying that no subtours involving a single *IN* are formed. The constraint on line 9, together with the domain specifications of variables $rob_i$ ensure that a node $i$ is serviced by a robot in $\mathbf{R_i}$, and that all nodes belonging to the same sequence are serviced by the same robot. These constraints are implemented as global Element constraints in CP. The constraint on line 10 assures that the servicing of a node $i \in \mathbf{V} \setminus \mathbf{O}$ does not commence before the servicing of its predecessor is completed, thereby ensuring that no subtours are formed. Whenever an *IN* $i \in \mathbf{I}$, is not used in any route, we noted earlier that $next_i = i$ will hold, but also notice that $d^r_{ii} = 0$, so the constraint in line 10 will not make the model infeasible. Notice that the inequality sign on line 10, as opposed to an equality sign, allows a robot to wait at a location before moving to the next location to avoid potential collisions. The constraints on lines 11 and 12 implement ECs and CCs, respectively. Implicit in the formulation is the fact that the robot servicing node $u$, i.e., $rob_u$, blocks $u$ for the duration $[S_u, S_{next_u}]$ instead of $\left[S_u, S_u + p^{rob_u}_u\right]$.

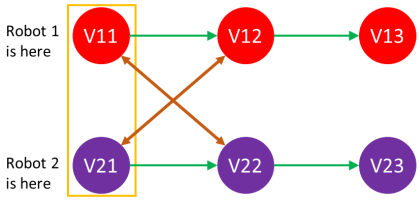
The $\geq$ symbol in the disjunctions of line 12 carries a special significance. Consider the scenario shown in Figure 5.1d, where tours for each robot are provided, and all that remains is to compute a schedule for those tours. Say, robot $r_1$ (resp. robot $r_2$) is currently located at node $V11$ (resp. $V21$), and the node it needs to service next on its tour is $V12$ (resp. $V22$). Due to the CCs $(V11, r_1, V22, r_2)$ and $(V12, r_1, V21, r_2)$, if only one robot advances to the next node on its tour, while the other robot remains on its current node, a collision will occur. If, however, we allow both robots to progress to their respective next nodes at the same time, i.e., set start times $S_{V12} = S_{V22}$, then the constraint in line 12 will not be violated.

### 5.2.1 Scheduling subproblem

MRSBE exhibits two strongly connected problems: a routing problem, similar to the Multiple TSP (mTSP), and a scheduling problem which bears resemblance to a machine scheduling problem. Even when the routes for each of the robots have been fixed, i.e., all assignment ($rob_i$) and sequencing ($next_i$) variables in Algorithm 12 have been fixed to some value satis-

**Algorithm 12** MRSBE problem description as a CP formulation

1: $next_i \in N^+(i) \quad \forall i \in \mathbf{O} \cup \mathbf{H}$
2: $next_i \in N^+(i) \cup \{i\} \quad \forall i \in \mathbf{I}$
3: $S_i \in \mathbb{Z}_{\geq 0} \quad \forall i \in \mathbf{V}$
4: $S_i = 0 \quad \forall i \in \mathbf{O}$
5: $rob_i \in \mathbf{R_i} \quad \forall i \in \mathbf{H}$
6: **min max**$_{i \in \mathbf{D}} S_i$
7: allDifferent$\big(next_i | i \in \mathbf{V} \setminus \mathbf{D}\big)$
8: **for all** $i \in \mathbf{V} \setminus \mathbf{D}$ **do**
9: $\quad rob_{next_i} = rob_i$
10: $\quad S_{next_i} \geq S_i + d^{rob_i}_{i,next_i}$
11: $S_i \geq \min_{w \in \boldsymbol{\mathcal{E}}_i} S_{next_w} \quad \forall i \in \mathbf{H}$
12: $rob_v = r_1 \wedge rob_w = r_2 \implies S_w \geq S_{next_v} \vee S_v \geq S_{next_w} \quad \forall (v, r_1, w, r_2) \in \mathbf{C}$




**(d)** Green arrow indicates the sequence of nodes each robot traverses. Brown double arrows indicate CCs.

fying constraints 7 and 9, the challenge to find starting times ($S_i$) that satisfy the scheduling constraints in lines 10, 11 and 12 remains. We refer to this problem as the *Scheduling Subproblem* (SSp). We will briefly digress away from the MRSBE problem to the SSp problem. Understanding the complexity of solving SSp will be helpful in understanding the complexity of MRSBE (Section 5.3), and help us in developing techniques to solve the MRSBE problem (Section 5.4).

Let $Seq_{r_1}, \ldots, Seq_{r_M}$ denote the set of robot tours satisfying constraints 7 and 9, where each tour is an ordered sequence of nodes, $\mathcal{N} = \bigcup_{i=1}^{M} Seq_{r_i}$ the set of nodes visited, $F = \{(v, next_v) | v \in \mathcal{N} \setminus \mathbf{D}\}$ the set of arcs traversed, and $pos(u)$ the position (index) of node $u$ in sequence $Seq_{rob_u}$. Generally, in this section, a node in a sequence can refer to either a task location or an *IN*. Throughout our discussions for SSp, we will not distinguish the node type. A feasible schedule $S$ assigns a start time to each node in $\mathcal{N}$ such that

$$S_u + p^{rob_u}_u \leq S_v, \forall (u, v) \in F \tag{5.1}$$

$$S_v \geq \min_{w \in \boldsymbol{\mathcal{E}}_v} S_{next_w}, \forall v \in \mathcal{N} \tag{5.2}$$

$$(S_w \geq S_{next_v}) \vee (S_v \geq S_{next_w}), \quad \forall v, w \in \mathcal{N} \text{ s.t. } (v, rob_v, w, rob_w) \in \mathbf{C} \tag{5.3}$$

The ability to solve the SSp efficiently is essential to the performance of the local search procedure we will propose for MRSBE. To facilitate the development of an efficient solution method for SSp, we model this problem on a directed, acyclic state transition graph $G^{sp}(V^{sp}, A^{sp})$. A **state** $s \in V^{sp}$ records the location of each robot. As such, a state $s$ can be represented by a vector of length $|M|$, where $s[i] \in \mathcal{N}$ denotes the location (node) of the $i^{th}$ robot in state $s$. Two special states representing the starting state $s_{\mathbf{O}} = [o^{r_1}, \ldots, o^{r_M}]$ and the terminal state $s_{\mathbf{D}} = [d^{r_1}, \ldots, d^{r_M}]$ are included in $V^{sp}$. The arc set $A^{sp}$ represents the set of feasible *state transitions*. During each state transition, one or more robots advance their

positions along their respective tours. There exists an arc $(s, s') \in A^{sp}$ between a pair of states $s, s' \in V^{sp}$, $s \neq s'$, if $s'[i] = s[i]$ or $s'[i] = next_{s[i]}$ for all $i \in [1, M]$. Corresponding to any arc $(s, s') \in A^{sp}$, we will refer to state $s'$ as a successor of $s$. An example of an SSp instance is given in Figure 5.2a and a corresponding State Space Graph (SSG) is given in Figure 5.2b. Note that the SSG only includes *feasible* states, i.e., states that satisfy ECs, CCs, and reachable from $s_{\mathbf{O}}$ by transitioning arcs in $A^{sp}$. We denote the set of nodes that must have been serviced by the robots in order to reach state $s$ together with those that are being serviced at $s$, by $s^{-\mathbf{1}} \subseteq \mathcal{N}$. A state $s$ is collision free and enabling feasible iff:

- **Collision Free:** $(s[i], r_i, s[j], r_j) \notin \mathbf{C}$, $i, j \in [1, M]$.
- **Enabling Feasible:** For each $i \in [1, M]$, $\exists u \in \boldsymbol{\mathcal{E}}_{s[i]}$ such that $next_u \in s^{-\mathbf{1}}$.

The sequence of states on any path from $s_{\mathbf{O}}$ to state $s$ in $G^{sp}$ will be referred to as a state transition path (STP) to $s$. Intuitively, to find a feasible solution to a given SSp, we must find an STP $P$ to terminal state $s_{\mathbf{D}}$ in the SSG. We will refer to such a $P$ as a *complete STP*. If no complete STP exists, then the SSp instance is infeasible. Conversely, if a complete STP $P$ exists, we can compute the earliest time that each state $s$ along the path $P$ could occur. In turn, we can use this information to derive the service starting times $S_v$ for each individual node $v \in \mathcal{N}$, thereby obtaining a complete schedule. We formalize the relation between the SSp and its corresponding SSG through the following proposition.

**Proposition 4.** *An SSp instance admits a feasible schedule iff there exists a complete STP* $P = (s_0 = s_{\mathbf{O}}, s_1, \ldots, s_l = s_{\mathbf{D}})$ *in its corresponding SSG.*

*Proof.* ($\Rightarrow$) Given a feasible schedule ($Sch'$) to the SSp instance, we can construct the desired complete STP as follows. Sort the start times of the nodes and iterate over those times in an ascending manner. At each new start time $t$, introduce a state $s$ such that $s[i] = v$ where $v \in \{Seq_{r_i}\}$, $t \in \left[S_v^{Sch'}, S_{next_v}^{Sch'}\right)$ and $S_v^{Sch'}$ denotes the start time of $v$ in schedule $Sch'$.

($\Leftarrow$) Given a complete STP, we can construct a valid schedule ($Sch$) to the SSp instance as follows. Initialize $S_{o^r}^{Sch} = 0, \forall r \in \mathbf{R}$. Assume we have already assigned start times for all the nodes up to state $s_k$ (i.e., nodes in $s_k^{-\mathbf{1}}$ ), then we use those times to assign start times for nodes in $s_{k+1}$. Let $T = \max_{i \in [1,M]} S_{s_k[i]}^{Sch}$, then, observe that $T$ denotes the earliest time state $s_k$ could have occurred. Let $J = \{j | j \in [1, M], s_k[j] \neq s_{k+1}[j]\}$ denote the set of robot indices that have advanced their position, then Eqn (5.4) can be used to assign start times for nodes corresponding to $J$.

$$S_{s_{k+1}[j]}^{Sch} = \max\left(T, \max_{i \in J} S_{s_k[i]}^{Sch} + p_{s_k[i]}^{rob_i}\right) \ \forall j \in J \tag{5.4}$$

Equation (5.4) assigns the earliest permissible start time for each node by the given complete STP. □

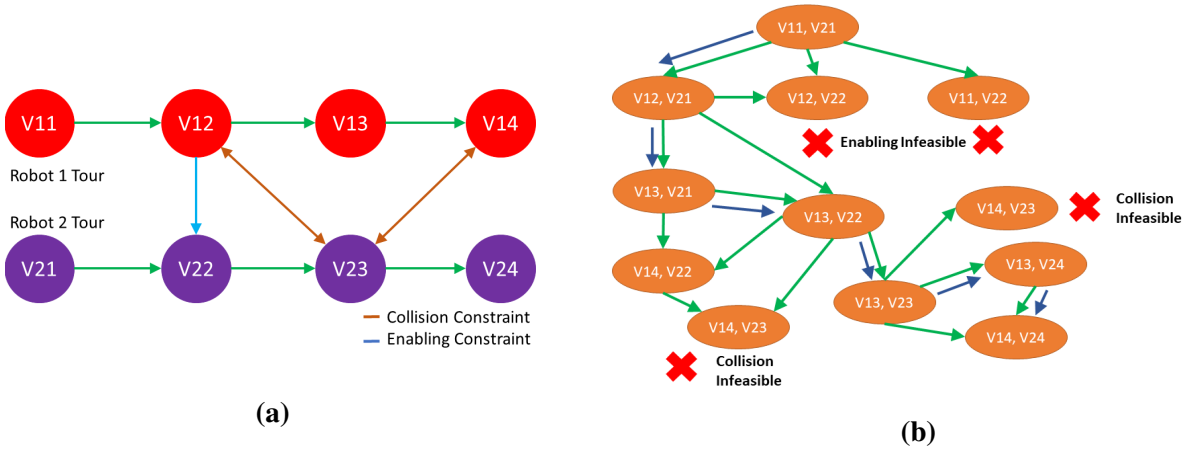



**Figure 5.2:** In Figure 5.2a, we show a graphical representation of SSp instance. $(V12, r_1, V23, r_2)$, and $(V14, r_1, V23, r_2)$ are the only CCs. $\mathcal{E}_{V22} = \{V12\}$ is the only EC, i.e., $V22$ can be serviced no earlier than servicing of $V13$ begins. In Figure 5.2b, the SSG for the SSp instance is shown. To help the reader understand CCs and ECs, we have also included infeasible states. A complete STP from starting state $s_{\mathbf{O}} = (V11, V21)$ to terminal state $s_{\mathbf{D}} = (V14, V24)$ is indicated in blue.

## 5.3 Problem complexity

Problem MRSBE is strongly NP-Hard, since in the absence of ECs and CCs, the MRSBE problem is simply the mTSP problem [Bektas, 2006]. More remarkable, however, is that determining whether the SSp (Section 5.2.1) permits a feasible solution is also an NP-Complete problem. We establish this result by reduction from the NP-Complete Deadlock Avoidance problem under the restriction "straight line", "single unit" and "single parameter" as stated in Araki *et al.* [1977]. Below, we provide an informal description of DA adapted from Garey and Johnson [1979], for a rigorous statement, see [Araki *et al.*, 1977].

**Deadlock avoidance.** *Assume a set $\{P_1, P_2, \ldots, P_m\}$ of "process flow diagrams" (line graphs), and a set $Q$ of "resources", where only a single unit of each resource type is present. The vertices of a process can either issue an allocation request for a single resource, or a deallocation request for a single resource allocated in a previous vertex of that process. A state $S$ of the system is given, where the current active vertex in each process describes $S$. The deadlock avoidance problem asks "Is $S$ 'safe'?", i.e., is there a sequence of resource allocations and deallocations that can enable the system to reach a "final" state starting from $S$?*

**Theorem 6.** *Determining whether SSp permits a feasible schedule is NP-Complete.*

*Proof.* In Appendix, see Section C.1. □

Even when the SSp instance is known to admit a feasible schedule, the complexity of computing a schedule with minimum makespan is strongly NP-Hard. We show the decision variant of this problem to be strongly NP-Complete by reducing the strongly NP-Complete Single Machine Scheduling with release times and deadlines problem [Garey and Johnson, 1977].

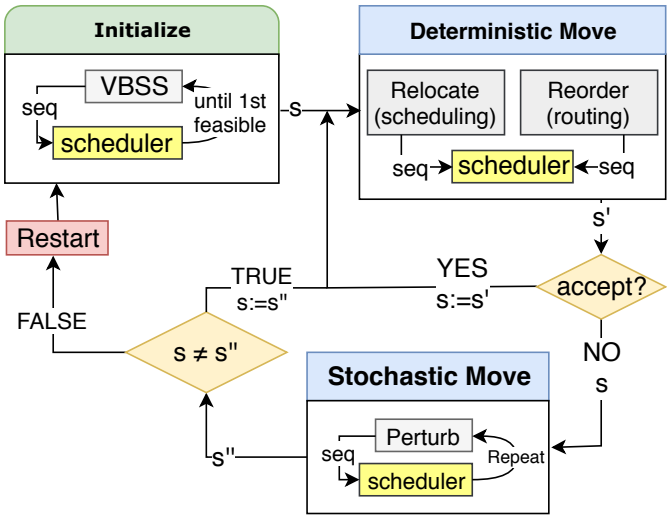



**Figure 5.3:** Heuristic Strategy

**Single machine scheduling with release times and deadlines [Garey and Johnson, 1977].** *Set $T = \{t_1, \ldots, t_M\}$ of tasks and for each task $t \in T$, a length $l(t) \in \mathbb{Z}^+$, a release time $r(t) \in \mathbb{Z}_0^+$, and a deadline $d(t) \in \mathbb{Z}^+$. Is there a one processor schedule for $T$ that satisfies the release time constraints and meets all the deadlines, i.e., a one-to-one function $\sigma : T \to \mathbb{Z}_0^+$ with $\sigma(t) > \sigma(t') \implies \sigma(t) \geq \sigma(t') + l(t')$, such that, $\sigma(t) \geq r(t)$ and $\sigma(t) + l(t) \leq d(t)$ for all $t \in T$?*

**Theorem 7.** *Given a SSp instance and time bound $B$, determining whether there exists a feasible schedule with makespan at most $B$ for the given SSp instance is strongly NP-Complete.*

*Proof.* In Appendix, see Section C.1. □

## 5.4 Overview of solution approach for MRSBE

Owing to the complexity results presented in the previous section, we propose a heuristic search procedure to obtain high quality MRSBE solutions in reasonable time. Within this search procedure, new feasible solutions are iteratively generated at each step, by (1) generating a new set of robot tours that visit all locations and (2) solving the SSp instance corresponding to those tours.

A schematic overview of the search is provided in Figure 5.3. An initial feasible solution $s$ is generated through a randomized constructive heuristic based on Value-Biased Stochastic Sampling (VBSS) [Cicirello and Smith, 2005]. A detailed description of this procedure is provided in Section 5.6. In short, the constructive heuristic first produces a set of robot routes, after which a scheduler is invoked to solve the corresponding SSp. The constructive heuristic is repeatedly invoked until a feasible solution is found. Next, our MRSBE heuristic attempts to improve upon this solution until some stopping criteria is met (e.g., a time limit). The details of the scheduler is provided in Section 5.5.

Conventional metaheuristic approaches often explore large, complex search spaces by randomly moving from one solution to the next, where new solutions are generated at random from some stochastic neighborhood structure. Preliminary experiments revealed that such random moves did not yield satisfactory results, as many of the random moves led to infeasible SSps. So instead of random neighborhood operators, we use 2 unique *deterministic* local search operators (REORDER, RELOCATE) to generate a new solution $s'$. These operators are designed to make well-informed, targeted moves and are more likely to yield feasible solutions than random moves. The moves are deterministic in the sense that the same input solution $s$ always leads to the same output solution $s'$, i.e., no randomness is involved.

After $s'$ has been generated, an acceptance criterion is used to determine whether $s'$ will be the starting solution for the next iteration (accept), or whether $s'$ will be discarded (reject). Here we use Late Acceptance (LA) [Burke and Bykov, 2017], a metaheuristic to determine whether a solution is accepted. Briefly, LA accepts a solution if the objective value of the solution (makespan) is better than the objective value of the solution obtained $L$ iterations ago, where $L$ is a fixed input parameter. To prevent cycling, $s'$ is rejected if it is identical to $s$.

If $s'$ is rejected, we generate a new solution $s''$ from a stochastic neighborhood of $s$ which becomes the starting solution for the next iteration. The stochastic neighborhood simply perturbs $s$ (comparable to a mutation in a Genetic Algorithm, where robots swap task locations). If the perturbation is successful, i.e., a new feasible solution $s''$ different from $s$ is obtained, then $s''$ becomes the starting point for the next iteration. If unsuccessful, a different permutation is attempted. A restart is performed to prevent the search from getting stuck when the Stochastic Move fails to produce a new feasible solution $s''$ after a fixed number of re-attempts.

In the subsections that follow, we discuss the principal components of our MRSBE procedure in detail. We begin by describing the scheduler to solve the SSp. In Sections 5.6 and 5.7, we describe the constructive heuristic and deterministic local search operators respectively.

## 5.5 Solving the scheduling sub-problem

The CP model introduced in Section 5.2 can be used to solve the SSp, but is generally too slow for this purpose. Since our MRSBE procedure (Figure 5.3) frequently solves SSp instances, we developed a fast but inexact scheduler for the SSp instead. The scheduler attempts to quickly find a path from the source to the terminal state in the SSG introduced in Section 5.2.1, or asserts that no such path exists, in which case the SSp instance is shown to be infeasible (Proposition 4). If the SSp instance is feasible, the STP returned from the scheduler is transformed into a feasible schedule using the procedure in Proposition 4.

The scheduler performs Depth First Search (DFS) on the SSG. A recursive implementation of DFS on the SSG is provided in Algorithm 13. Algorithm 13 takes as input a state $s$, a STP $P$ and start time of state $s$ denoted by $t_s$. If the SSp instance is feasible then Algorithm 13 returns a complete STP $P$, otherwise $P$ returned will be empty. The scheduler initiates the DFS search by passing $s_{\mathbf{0}}$, an empty list $P$, and 0 as the arguments. In line 2, we mark a state as discovered once it is expanded by the algorithm. We keep track of discovery status of each state so that we do not explore its successors more than once. In line 6, we enumerate the successors of each state, and compute the start time of each successor conditioned on the value

**Algorithm 13** Depth first search for SSp

```
1: DFS(s, P, t_s)
2:    Mark s as discovered
3:    P.push_back(s)
4:    if s = s_D then
5:       return 1;
6:    Enumerate successors of s in the state space graph of the SSp, and compute the start time of
      each successor
7:    Sort the successors states according to their start times computed using the procedure in Propo-
      sition 4, and store the sorted successors in vector vec_s
8:    for i = 1 : len(vec_s) do
9:       if v is not marked discovered then
10:         if 1 = DFS(v, P, t_v) then
11:            return 1;
12:   P.pop_back()
13:   return 0;
```

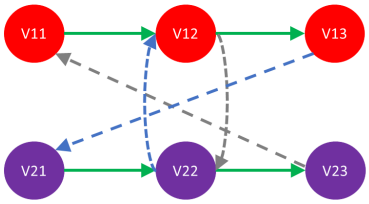
$t_s$, using the procedure in Proposition 4. In order to increase the likelihood that a schedule with short makespan is obtained at the end of the search, in line 7 we rank all the successors. A state with an earlier start time is ranked higher. In line 8, we only explore previously undiscovered successors, and in line 9 we recursively call DFS for the chosen successor. Finally in line 4, once we have found a path to the terminal state, the DFS procedure terminates and returns the complete STP. Using the start times computed for each state, the start times are obtained for each node in the SSp instance.

To improve the efficiency of this basic DFS procedure, we introduce two extensions. First, we specify and exploit conditions that allow us to restrict the number of successors that are enumerated at each step of the search (which could be no smaller than $\mathcal{O}\left(2^{\frac{M}{2}}\right)$ in the worst case) while continuing to ensure completeness of the search. Second, we specify techniques that reduce the search space by attempting to pruning states from which no path to the terminal state exists in SSG.

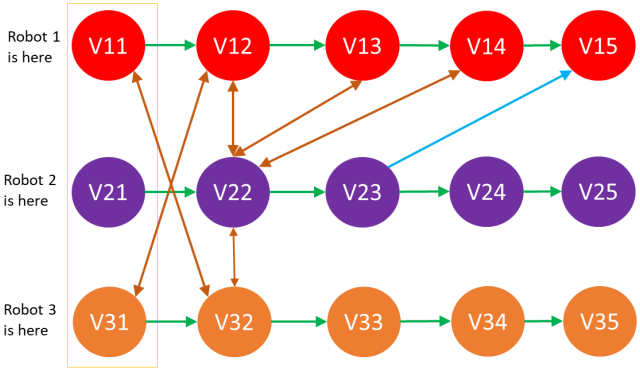
To facilitate the above extensions, we make use of the Alternative Graph (AG) introduced earlier in Section 2.2.2. Notice that the SSp problem specified in Section 5.2.1 is semantically identical to BJ-OPT problem in Section 2.2.2, so we can represent the SSp on an AG instead. The AG helps capture problem structure that is not reflected in the SSG. Particularly, an AG provides a basis for representing and reasoning about temporal constraints, which is useful both in performing look-ahead analysis of possible alternatives, and in pruning infeasible states. We first introduce the AG representation and then consider our DFS efficiency enhancements.

### 5.5.1 Alternative graph representation of SSp.

Both collision and enabling constraints in Eqns (5.2) and (5.3) are logical disjunctions. For instance, the EC in Eqn (5.2) can be written as $\bigvee_{w\in\boldsymbol{\mathcal{E}}_i}(S_i \geq S_{next_w})$ for $i \in \mathcal{N}$. A solution




**(a)** Alternative graph for the SSp instance shown in Figure 5.1d. The grey broken arrows represent the alternative arcs corresponding to the CC $(V11, r_1, V22, r_2)$, and the blue broken arrows represent the alternative arcs corresponding to CC $(V12, r_1, V21, r_2)$.




**(b)** Benefit of using AG with SSG, the blue arc is an EAA.

**Figure 5.4**

to the SSp can only be feasible if exactly one clause of each CC and at-least one clause for each EC disjunction evaluates to true. An AG can be interpreted as a special kind of temporal network which encodes these disjunctions. Arcs in a temporal network model temporal relations between pairs of nodes. For instance, an arc $(u, v)$ with length $l(u, v)$ corresponds to the constraint $S_v \geq S_u + l(u, v)$. Formally, an AG is a directed, weighted graph $G(\mathcal{N}, F, \mathcal{A}, En)$ where $\mathcal{N}$ resp. $F$ are the sets of nodes and arcs visited resp. traversed by the robots as defined in Section 5.2.1. $\mathcal{A}$ and $En$ are sets of arcs that relate to the collision and enabling constraints. For every CC $c = (v, rob_v, w, rob_w) \in \mathbf{C}$ that is relevant to the SSp instance, we define $\mathcal{A}_c = \{(next_v, w), (next_w, v)\}$, and $\mathcal{A} = \{\mathcal{A}_c | c \in \mathbf{C} \text{ s.t. } c \text{ is relevant to the SSp instance}\}$. Similarly, for every $v \in \mathcal{N}$ for which there does not exist a vertex $w \in \boldsymbol{\mathcal{E}}_v$ with $rob_w = rob_v$ and $pos(w) < pos(v)$, that is, for every vertex $v$ that is not enabled by another vertex serviced earlier in its own tour, we define:

$$En_v = \{(u, v) | w \in \boldsymbol{\mathcal{E}}_v, u = next_w, rob_w \neq rob_v\}$$

Let $En = \{En_v\}_{v \in \mathcal{N}}$. An arc $(u, v) \in \mathcal{F}$ has length $p_u^{rob_u}$, whereas arcs in $\mathcal{A}$ and $En$ have length 0. The AG for the SSp instance in Figure 5.1d is shown in Figure 5.4a.

Following from the disjunctive representation of a CC, let the set of arcs $\mathcal{A}_c$ correspond to the set of alternative temporal constraints which can satisfy a particular CC c. A single arc (constraint) chosen from this set is said to be a *selection* for CC c. Analogously, an arc selected from $En_v$ is a selection for the EC for node $v$. As both CCs and ECs offer a set of alternatives to choose from, those arcs will be referred to as alternative arcs. When we want to distinguish the type, we will refer to them as Enabling Alternative Arc (EAA) or Collision Alternative Arc (CAA). A *complete selection* $Cs$ is a set of alternative arcs containing a selection for every member in $En$ and $\mathcal{A}$. A partial selection $Ps$ is a set of alternative arcs which does not contain a selection for at least one member in the sets $En, \mathcal{A}$.

Given the above definition of an AG, computing a feasible solution to an SSp instance reduces to the problem of identifying a complete selection $Cs$ such that the AG with selection $Cs$, does not contain a positive-length cycle [Mascis and Pacciarelli, 2002]. Such a

selection is said to be a *complete consistent selection*. We denote the AG with selection $Cs$ by $G(\mathcal{N}, F \cup Cs)$, i.e., the arcs in $G$ are specified by the set $F \cup Cs$. Given a complete consistent selection $Cs$, it is possible to compute a schedule for the SSp instance by setting the start time of a node $v \in \mathcal{N}$ to the length of the longest path to that node in the graph $G(\mathcal{N}, F \cup Cs)$. In the absence of positive-length cycles, computing these longest paths can be accomplished in linear time. Finally, the problem of computing the optimal solution to an SSp instance, i.e., a schedule with minimum makespan, is equivalent to identifying a complete selection $Cs$ such that the length of the longest path in $G(\mathcal{N}, F \cup Cs)$ is least among all complete consistent selections.

Performing DFS on the SSG is equivalent to making selections in the AG by utilizing a search tree with backtracking. At each node of the search tree, we have a partial selection $Ps$, and a heuristic chooses the next alternative arc to include in $Ps$. To see the potential benefit of including the AG representation in our search procedure for finding a feasible solution to the SSp instance, consider the three robot SSp instance shown in Figure 5.4b. CCs are marked by brown double arrows. We include an EC $\mathcal{E}_{V15} = \{V22\}$, and so $En_{V15} = \{(V23, V15)\}$.

If we are using a SSG representation, then we are forced to reason forward in time. Say the robots are at state $s_0 = (V11, V21, V31)$. From this state there are only two collision free successor states, namely $s_1 = (V12, V21, V32)$, and $s_2 = (V11, V22, V31)$. If say DFS chooses to explore the state $s_1$ first, then notice that there can be no path to the terminal state from $s_1$ in the SSG. To see why, notice that once robot $r_1$ reaches $V14$, the only way it can start servicing $V15$ is if robot $r_2$ begins servicing $V23$, due to the enabling relationship. However, as long as robot $r_1$ is at any node in the set $\{V12, V13, V14\}$, $r_2$ cannot begin servicing $V22$. In this case, DFS would enumerate twelve states before backtracking from $s_1$.

Looking at the problem from an AG perspective, however, it is possible to avoid enumerating the state $s_1$ in the first place. Since $En_{V15}$ is a singleton, notice that any consistent selection for the given SSp instance must include the arc $(V23, V15)$ (shown in blue). Corresponding to CC between $V14$ and $V22$, we have the following CAAs $\{(V23, V14)$, $(V15, V22)\}$. If the CAA $(V15, V22)$ was included in our selection, then a cycle is created through the arcs $(V15, V22)$, $(V22, V23)$, $(V23, V15)$. Hence, it is implied that the paired CAA of $(V15, V22)$, which is $(V23, V14)$, must be included in the selection.

After including $(V23, V14)$ in the selection, and by repeating arguments similar to those made above for other CCs involving only $r_1$ and $r_2$, it can be inferred that the CAA $(V23, V12)$ must be present in any consistent selection for the given SSp instance. The presence of $(V23, V12)$ in any selection implies that the start time of $V12$ can occur no earlier than the start time of $V23$ (and by extension $V22$) in any feasible schedule. Clearly, state $s_1$ violates the temporal relationship between $V12$ and $V22$.

By making selections on the AG similar to those shown for the CCs between $r_1$ and $r_2$ in the example, we can provide better guidance to the DFS search procedure. Alternative selections enable us to cut-off portions of the SSG that do not lead to a feasible solution.

### 5.5.2 Linking the SSG and AG

To use the AG as a means of eliminating states from the SSG, we must synchronize the SSG and the AG. This is accomplished by simultaneously updating the alternative arc selections

in the AG each time the DFS procedure traverses to a new state in the SSG. Assume that the DFS has produced a STP $P$ to some state $s_{curr}$, and let $Ps$ denote the selections on the AG that have been updated during the search thus far. Then synchronization of $P$ and $Ps$ involves enforcement of two basic conditions. First, since $P$ contains decisions for all CCs involving at least one node in $s_{curr}^{-\mathbf{1}}$ and all ECs for nodes in $s_{curr}^{-\mathbf{1}}$, then for each of these CCs and ECs, $Ps$ must contain a selection which does not contradict $P$. This property of $Ps$ can be confirmed by carrying out the following set of checks:

1. Corresponding to each CC $c = (u, rob_u, v, rob_v)$ where $u, v \in s_{curr}^{-\mathbf{1}}$, $Ps$ has a selection for $c$. We can compute the start times of $S_u, S_v$ from $P$ using the procedure in Proposition 4. If $S_u < S_v$ then $Ps$ must contain the CAA $(next_u, v)$, else the CAA $(next_v, u)$ must be present in $Ps$. Similarly, for each node in $u \in s_{curr}^{-\mathbf{1}}$, we can determine which node among the enablers of $\boldsymbol{\mathcal{E}}_u$ has the earliest start time. The EAA in $En_u$ corresponding to the earliest enabler must be present in $Ps$.

2. Start times of all nodes in $s_{curr}^{+\mathbf{1}}$ cannot occur earlier than start time of any node in $s_{curr}^{-\mathbf{1}}$. This implies that any alternative arc $(h, k)$ with $h \in s_{curr}^{+\mathbf{1}}$ and $k \in s_{curr}^{-\mathbf{1}}$ must be absent from $Ps$. Further, if $(h, k)$ is of type CAA, then the paired CAA of $(h, k)$ must be present in $Ps$.

Any selection in $Ps$ satisfying 1 and 2 will be referred as a **consistent positional selection** for STP $P$, a terminology borrowed from Mascis and Pacciarelli [2002]. Notice that 1 and 2 do not preclude $Ps$ from containing arcs of the form $(h, k)$ where $h, k \in s_{curr}^{+\mathbf{1}}$, even when a relation between $S_h$ and $S_k$ could not have been derived using $P$. In fact it is precisely these kinds of arcs that will be used to eliminate states from being explored in the future by DFS. In Section 5.5.4, we provide methods to obtain such arcs. The arc $(V23, V12)$ from our previous example, is an arc that could not have been inferred from the STP to state $(V11, V21, V31)$.

The second basic aspect of synchronizing $P$ and $Ps$ is ensuring that DFS successor enumeration step does not generate successors that violate previously made selections in $Ps$. For instance, in the previous example, while enumerating successors of $s_0$, we should not generate $s_1$ if $Ps$ already contains the CAA $(V23, V12)$. Furthermore, $Ps$ must be properly updated whenever a new state is added to $P$ during DFS. Mechanisms for this are defined below.

### 5.5.3 State enumeration procedure

In this section, we describe our procedure for generating all feasible successors of a state to explore during DFS. Assume that at the current stage of the search there is a STP $P$ to state $s_{curr}$, and a consistent positional selection $Ps$ for $P$. Further, let $B_{s_{curr}}$ denote the set of all nodes in $\mathcal{N}$, immediately following those in state $s_{curr}$ in their respective robot tours (i.e., $B_{s_{curr}} = \{w | i \in [1, M], w = next_{s_{curr}[i]}\}$). The problem of generating successors to $s_{curr}$ then is that of determining which subsets of nodes in $B_{curr}$ define feasible extensions of $P$.

To illustrate, consider the example given in Figure 5.5a. Figure 5.5a shows state $s_{curr}$ and the CCs between nodes in $s_{curr}$ and $B_{s_{curr}}$. (All nodes in $s_{curr}^{-\mathbf{1}} \backslash s_{curr}$ have been omitted, as CCs associated with those nodes are inconsequential in determining the successors of $s_{curr}$). In Figure 5.5b the corresponding selections in $Ps$ are shown using red and black arcs. Notice that $Ps$ is indeed a consistent positional selection for $P$ and that all selections in Figure 5.5b

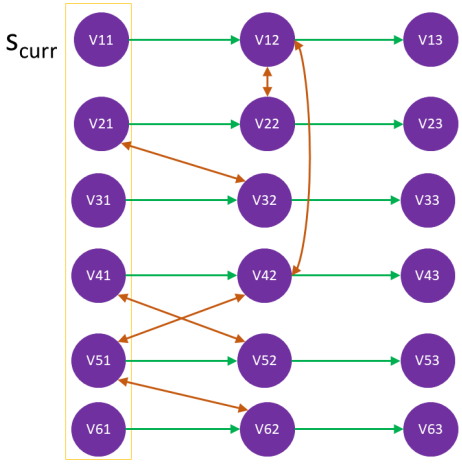



Figure 5.5(a) Brown double arrows indicate CCs.

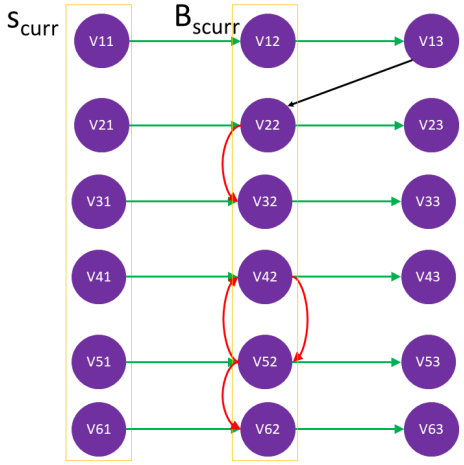



Figure 5.5(b) Alternative Graph with arcs in $Ps$ shown.

apart from the CAA $(V13, V22)$ are derived from $P$. Note also that no selection has been made for CC $(V12, r_1, V42, r_4)$.

Consider the situation in Figure 5.5a for robots $r_4, r_5$ in $s_{curr}$, which is similar to the deadlock situation in Figure 5.1d. Owing to presence of CAAs $(V42, V52)$, $(V52, V42)$ in $Ps$, the start time of $V42, V52$ must be the same in any complete consistent selection containing $Ps$. So, if $V42$ occurs in a successor of $s_{curr}$, then so must $V52$. Generalizing from this example, notice that $V42, V52$ forms a maximal strongly connected component (scc) in the induced sub-graph $G_{Ps}[B_{s_{curr}}]$ (i.e. the subgraph induced in $G(\mathcal{N}, F \cup Ps)$ by the nodes in $B_{s_{curr}}$). Since all nodes in an identified maximal scc will either all occur or all not occur in any feasible successor, we can potentially reduce the size of the search for feasible successors by reasoning about which maximal sccs of $G_{Ps}[B_{s_{curr}}]$ occur instead of which nodes in $B_{s_{curr}}$ occur. For our example, the maximal sccs of $G_{Ps}[B_{s_{curr}}]$ are $\mathbf{c_1} = \{V12\}, \mathbf{c_2} = \{V22\}, \mathbf{c_3} = \{V32\}, \mathbf{c_4} = \{V42, V52\}$, and $\mathbf{c_5} = \{V62\}$.

Equipped with this insight, we focus on specifying the set of constraints that dictate the feasible co-occurrence of maximal sscs in $G_{Ps}[B_{s_{curr}}]$ due to CCs and ECs. (We assume without loss of generality that such CCs will not appear between nodes in a given scc, because otherwise we know that $P$ cannot be extended to a complete STP). Let $SCC = \{c_1, \ldots, c_k\}$ denote the set of all maximal SCCs in $G_{Ps}[B_{s_{curr}}]$. For each maximal scc $c$ in $SCC$, we introduce a binary literal $y_c \in \{0, 1\}$ to indicate whether nodes in the maximal scc $c$ occur in the successor of $s_{curr}$. If $y_c = 1$, then all nodes in scc $c$ occur in the successor. Else if $y_c = 0$, then robots corresponding to the nodes in $c$ do not advance their position i.e. they continue to remain on the node they occupied in state $s_{curr}$. We then construct a SAT formula (denoted by $CF_{s_{curr}}$) using these binary literals that every feasible successor of $s_{curr}$ must satisfy. We develop the formula incrementally by considering different cases.

**Case 1:** The first clause ensures that for scc $c$ to appear in a successor, then all sccs on which $c$ depends on must also appear. In Figure 5.5b, the CAA $(V22, V32)$ in $Ps$ illustrates this case. We know that the start time of $V23$ cannot occur earlier than the start time of $V22$ in any complete consistent selection containing $Ps$. So for scc $\mathbf{c_3} = \{V32\}$ to appear in the

successor of $s_{curr}$, scc $\mathbf{c_2} = \{V22\}$ must also appear.

To specify this case more precisely, let $Pred^{Ps}(c) \subseteq \{SCC \backslash c\}$ denote a set of maximal sccs, where scc $d \in Pred^{Ps}(c)$ iff there exists an arc $a$ in $G_{Ps}[B_{s_{curr}}]$, whose tail belongs to some node in the scc $d$, and head belongs to some node in the scc $c$. For our example $Pred^{Ps}(\mathbf{c_3}) = \mathbf{c_2}$. We use the clause $CF1_c^{Ps}$ to impose the condition that for scc $c$ with $Pred^{Ps}(c) \neq \emptyset$ to appear in the successor of $s_{curr}$, then all sccs in $Pred^{Ps}(c)$ must also be included in the successor. The condition can be enforced as shown below:

$$CF1_c^{Ps} = \underset{\{j|c_j \in Pred^{Ps}(c)\}}{\wedge} \left(y_c \rightarrow y_{c_j}\right) \tag{5.5}$$

where $\rightarrow$ is the logical implication operator.

**Case 2:** The second clause we introduce ensures that no scc $c$ in the successor contains a node that violates an enabling constraint. To formalize, define the following sets for any node $u$ belonging to scc $c$ for which $En_u \neq \emptyset$ (i.e., $u$'s set of enablers is non-empty, refer Section 5.5.1 for definition):

$$EnT(u) = \{w|(w,u) \in En_u\} \tag{5.6}$$

$$K(u) = \{d|d \in SCC \text{ s.t. } d \text{ contains a node also present in } EnT(u)\} \tag{5.7}$$

$$U_c = \{u|u \in c, En_u \neq \emptyset, \mathbf{s_{curr}^{-1}} \cap EnT(u) = \emptyset\} \tag{5.8}$$

where $U_c$ denotes the set of nodes in $c$ that each need to be enabled if they are to appear in a successor. We introduce a conjunction of clauses in $CF2_c^{Ps}$, one for each node in $U_c$ to ensure they are enabled as shown below:

$$CF2_c^{Ps} = \underset{u \in U_c}{\wedge} \left(y_c \rightarrow 0 \vee \underset{c_j \in K(u)}{\vee} y_{c_j}\right) \tag{5.9}$$

In Equation (5.9), if there exists a node $u \in U_c$ that cannot be enabled by any other node in $B_{s_{curr}}$ i.e. $K(u) = \emptyset$, then we need to enforce $y_c = 0$. To do so, we include 0 as shown in Equation (5.9).

**Case 3:** The third clause ensures that states with collisions are avoided. Consider the set $CO$, where

$$\begin{aligned} CO = \{&(c_i, c_j)|c_i, c_j \in SCC, c_i \neq c_j \text{ and } \exists\,(u, rob_u, v, rob_v) \in \mathbf{C}, \\ &\text{where node } u \text{ is part of } c_i, \text{ node } v \text{ is part of } c_j.\} \end{aligned}$$

Each member of $CO$ is a pair of sccs both belonging to $SCC$, such that, if both sccs occur in a successor then the successor state will contain a collision. To avoid simultaneously including nodes from sccs that can lead to collisions, we define $CF3^{Ps}$ as:

$$CF3^{Ps} = \underset{(c_i,c_j) \in CO}{\wedge} \left(y_{c_i} \rightarrow (1 - y_{c_j})\right) \tag{5.10}$$

The clause $\left(y_{c_i} \rightarrow (1 - y_{c_j})\right)$ means that if nodes from $c_i$ occurs in the successor then nodes from $c_j$ cannot occur and vice versa.

**Case 4:** The fourth clause ensures that an scc $c$ whose start depends on the start of an unvisited node not in $B_{s_{curr}}$ cannot occur. Consider the example in Figure 5.5b once more. Due to CAA $(V13, V22)$ in $Ps$, we know that the start time of $V22$ cannot occur earlier than the start time of $V13$ in any complete consistent selection containing $Ps$. $V13$ cannot occur in any successor of $s_{curr}$ since it does not belong to $B_{s_{curr}}$, and consequently, we know that $V22$ cannot occur in any successor of $s_{curr}$ either.

To incorporate this constraint, Let $Y_c \in \{0, 1\}$ be 0 if there exists a path in $G(\mathcal{N}, F \cup Ps)$ from some node in $s^{+\mathbf{1}}_{curr} \backslash B_{s_{curr}}$ to nodes in $c$, and 1 otherwise. Then, to eliminate nodes that are part of scc $c$ with $Y_c = 0$ from occurring in the successor, we introduce a clause $CF4^{Ps}_c$:

$$CF4^{Ps}_c = (y_c \rightarrow 0) \tag{5.11}$$

**Case 5:** Finally, we define $CF^{Ps}_{s_{curr}}$, our target construction, by aggregating all clauses defined for each of the above cases with a clause that ensures that at least one of the sccs from $SCC$ appears in the successor.

$$CF^{Ps}_{s_{curr}} = \left(\bigvee_{c \in SCC} y_c\right) \wedge CF3^{Ps} \wedge \bigwedge_{c \in SCC} \left(CF1^{Ps}_c \wedge CF2^{Ps}_c \wedge CF4^{Ps}_c\right) \tag{5.12}$$

For the example considered, if we assume the presence of one additional EC $\boldsymbol{\mathcal{E}}_{V62} = \{V11, V51\}$, then the clauses introduced to construct $CF^{Ps}_{s_{curr}}$ are the following:

$$\begin{aligned} CF^{Ps}_{s_{curr}} = \; & (y_{\mathbf{c_5}} \rightarrow y_{\mathbf{c_4}}) \wedge (y_{\mathbf{c_3}} \rightarrow y_{\mathbf{c_2}}) \wedge (y_{\mathbf{c_5}} \rightarrow y_{\mathbf{c_4}} \vee y_{\mathbf{c_1}}) \\ & \wedge (y_{\mathbf{c_1}} \rightarrow (1 - y_{\mathbf{c_4}})) \wedge (y_{\mathbf{c_2}} \rightarrow 0) \wedge \left(\bigvee_{i=1}^{5} y_{\mathbf{c_i}}\right) \end{aligned} \tag{5.13}$$

Any feasible successor represented in terms of the Boolean literals $y_.$ must satisfy Equation (5.12). Conversely, from any solution to Equation (5.12), we can derive a successor. So a basic strategy for deriving all feasible successors of $s_{curr}$ is to simply enumerate all solutions to $CF^{Ps}_{s_{curr}}$. For the clauses listed in Eqn (5.13), there are three solutions to $CF^{Ps}_{s_{curr}}$. If we denote a solution as the list of the literals corresponding to those sccs $c_i$ for which $y_{c_i} = 1$, the three solutions are: (a) $y_{\mathbf{c_1}}$, (b) $y_{\mathbf{c_4}}$ and (c) $y_{\mathbf{c_4}}, y_{\mathbf{c_5}}$. However, it is not necessary to enumerate all feasible solutions of $CF^{Ps}_{s_{curr}}$ to ensure completeness of search. It is sufficient to just enumerate the minimal models of Equation (5.12) to ensure completeness of search. The minimal solutions for our example are (a) $y_{\mathbf{c_1}}$, (b) $y_{\mathbf{c_4}}$ and the successors derived corresponding to each of those minimal solutions are $(V12, V21, V31, V41, V51, V61)$ and $(V11, V21, V31, V42, V52, V61)$.

Further, although the number of minimal models (solutions) is still exponential in the number of robots in the worst case (an example is provided in Appendix C.2), it is also possible to establish conditions where it is not.

**Proposition 5.** *For state $s_{curr}$, if for each scc $c \in SCC$ and each node $u \in U_c$ (refer Equation (5.8)) we have $|K(u)| \leq 1$ (refer Equation (5.7)), then $CF^{Ps}_{s_{curr}}$ has at most $M$ minimal solutions.*

*Proof.* In Appendix C.1. □

Informally Proposition 5 says that if for every node that needs to be enabled in order to appear in the successor (i.e. nodes in the set $\bigcup_{c \in SCC} U_c$), there is at most one enabler for each node, then the number of minimal solutions is at most $M$. Clearly if there are no ECs in the problem, the statement in Proposition 5 holds true at every state that DFS expands. The proof given in the appendix also provides an efficient algorithm for enumerating minimal solutions in this case.

**Selection update step**

Once we have generated successors and selected a particular successor to continue the search, we need to update the AG with selections for the updated STP $P$. Let $\bar{y}$ denote a feasible solution to $CF^{Ps}_{s_{curr}}$, and let state $s_{next}$ be the successor generated using $\bar{y}$. Let $W$ denote the set of nodes appearing in $s_{next}$ that do not appear in $s_{curr}$. Let $\bar{\mathcal{A}}_{Ps}$ denote members of $\mathcal{A}$ for which $Ps$ does not contain a selection. Let DFS choose to transition to $s_{next}$ from $s_{curr}$. Selections added to $Ps$ are:

1. Since $Ps$ already includes previous selections for CCs involving nodes in $s^{-\mathbf{1}}_{curr}$, it is sufficient to include only selections for CCs involving at least one node in $W$. If a CAA $a$ with its head to a node in $W$ exists among members of $\bar{\mathcal{A}}_{Ps}$, then CAA $a$ is excluded from $Ps$ and the paired CAA of $a$ is included in $Ps$.

2. Corresponding to each node $u$ in $W$ for which an enabling clause was added in $CF2^{Ps}_c$ in Eqn (5.9), we would have to append the corresponding selection from $En_u$ into $Ps$.

If it becomes necessary to subsequently backtrack from $s_{next}$ back to $s_{curr}$, then $Ps$ is updated by simply removing all arcs that were added to $Ps$ during the transition from $s_{curr}$ to $s_{next}$.

### 5.5.4 Simplifying the SSp formulation through logical inference

Complementary to our use of the AG to improve the efficiency of state enumeration, the size of the SSp formulation can also be reduced (sometimes dramatically) by filtering the constraints in the AG. Specifically, both redundant constraints and infeasible alternative arcs can be removed from the AG through logical inference, and as these constraints are removed it becomes possible to merge nodes in the original AG, and as the number of nodes in the AG decreases, the number of states DFS needs to explore also decreases. We illustrate this with an example.

**Example 12.** *Consider the 2 robot example in Figure 5.6, and assume that* $\boldsymbol{\mathcal{E}}_{V22} = \{V12, V23\}$, $\boldsymbol{\mathcal{E}}_{V24} = \{V11\}$, *and* $c = (V12, r_1, V23, r_2) = \mathbf{C}$. *With respect to* $\boldsymbol{\mathcal{E}}_{V22} = \{V12, V23\}$, *notice that for these specific robot tours* $V22$ *can only be enabled by* $V12$, *so the start time of* $V22$ *in any feasible schedule can be no earlier than the completion time of* $V12$. *Hence* $\{(V13, V22)\}$ *is selected for* $En_{V22}$ *(indicated by the blue arrow), and* $Ps$ *is initialized with that selection.*

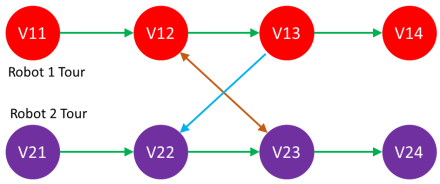



**Figure 5.6:** Collision constraint filtering, the blue arc represents the EAA $(V13, V22)$.

*In $G(\mathcal{N}, F \cup Ps)$, presence of the path $V12 \rightarrow V13 \rightarrow V22 \rightarrow V23 \rightarrow V24$ implies the following temporal relations between start times of service locations:*

$$S_{V12} < S_{V13} \leq S_{V22} < S_{V23} < S_{V24} \tag{5.14}$$

*For CC $c$, Eqn (5.14) excludes the choice of CAA $(V24, V12)$.*

*So the only feasible selection for CC $c$ is the arc $a_c = (V13, V23)$. However, notice that $G(\mathcal{N}, F \cup Ps)$ already contains a path from $V13$ to $V23$ $(V13, V22, V23)$ whose length is at least as long as the 0 length arc $a_c$. Hence, adding arc $a_c$ is redundant and by extension CC $c$ is also redundant in $G(\mathcal{N}, F \cup Ps)$. So the pair of alternative arcs associated with CC $c$ can be removed. Finally, observe that $V11 \in \boldsymbol{\mathcal{E}}_{V24}$, and that $V11$ is serviced prior to $V24$ in any feasible solution. As such, we can infer that $V24$ is guaranteed to be enabled, and any EAAs associated with $\boldsymbol{\mathcal{E}}_{V24}$ can be removed from the AG.*

*Once CC $c$ is shown to be redundant, notice that there are no CCs involving any node from $V11, V12, V23, V24$. Further, only the start time of $V13$ is needed to ensure that EC for $V22$ is satisfied. Hence, we can contract the arc $(V11, V12)$, thereby merging nodes $V11$ and $V12$ into a single node with a processing time equal to the sum of processing times of $V11$ and $V12$. Similarly, we can merge nodes $V23$ and $V24$.*

To be able to make inferences like the one in our example, we must first deduce precedence relations between nodes in the AG. While any transitive reduction algorithms for DAGs can be used to deduce transitive relations between nodes, we propose an algorithm that exploits the fact that the nodes within the AG can be associated with a membership to a robot. Given a partial selection $Ps$ and graph $G(\mathcal{N}, F \cup Ps)$, we define $f_v^r \in \mathcal{N}$ as the last node in the tour of robot $r \in \mathbf{R}$ that must be serviced no later than node $v \in \mathcal{N}$. $f_v^r$ is defined in Eqn (5.15).

By definition of $f_v^r$, we have $S_v \geq S_{f_v^r}$ in all feasible schedules of the SSp instance whose corresponding complete selection contains $Ps$. Algorithm 14 shows how to compute $f_v^r$ efficiently. This algorithm takes as input graph $G(\mathcal{N}, F \cup Ps)$ and the set $TOP$ containing the vertices in $\mathcal{N}$ stored in topological order. The values of $f_v^r$ are computed by iterating over the nodes in topological order, and by comparing the value of $f_v^r$ for node $v$ to the corresponding values of its immediate predecessors $(N_G^-(v))$ in graph $G(\mathcal{N}, F \cup Ps)$. The runtime complexity of Algorithm 14 is bounded by $\mathcal{O}(M^2|\mathcal{N}|)$. This follows from the fact that every node in $v \in \mathcal{N}$ has at most one non-redundant incoming arc from a different robot, i.e., $|N_G^-(v)| \leq M$.

$$f_v^r = \begin{cases} v & \text{if } r = rob_v \\ o^r & \text{if no path exists from a node in } Seq_r \text{ to } v \\ \left\{ \arg\max_{w \in Seq_i} pos(w) \,\middle|\, \substack{\exists \text{ a path from } w \text{ to } v \\ \text{in } G(\mathcal{N}, F \cup Ps)} \right\}, & \\ \text{otherwise} & \end{cases} \tag{5.15}$$

**Algorithm 14** Computing $f_v^r$

**Data:** $G(\mathcal{N}, F \cup Ps), TOP$

$$f_v^r = \begin{cases} v & \text{if } r = rob_v \\ o^r & \text{otherwise} \end{cases}$$

**for all** $v \in TOP$ **do**
  **for all** $w \in N_G^-(v)$ **do**
    **for all** $r \in \mathbf{R}$ **do**
      **if** $pos(f_w^r) > pos(f_v^r)$ **then**
        $f_v^r = f_w^r$

Using the definition of $f_v^r$, we state the following two propositions to filter redundant alternative arcs from the AG.

**Proposition 6.** *Given a feasible partial selection $Ps$, and assume $Ps$ does not contain a selection for CC $c = (u, rob_u, v, rob_v)$. CC c is redundant if:*

$$(pos(f_v^{rob_u}) \geq pos(next_u)) \vee (pos(f_u^{rob_v}) \geq pos(next_v)) \tag{5.16}$$

*Proof.* Corresponding to CC $c$, exactly one arc from the set $\mathcal{A}_c = \{(next_u, v), (next_v, u)\}$ must be selected. W.l.o.g assume that $pos(f_v^{rob_u}) \geq pos(next_u)$ is satisfied. This implies that there already exists a path between $next_u$ to $v$ in $G(\mathcal{N}, F \cup Ps)$, i.e., $next_u$ precedes $v$, so a collision between $u$ and $v$ can never occur. As such, CC $c$ can be safely removed. □

**Proposition 7.** *Given a feasible partial selection $Ps$, the ECs for node $v \in \mathcal{N}$ are redundant if $w \in \boldsymbol{\mathcal{E}}_v$ and $pos(f_v[rob_w]) \geq pos(next_w)$.*

*Proof.* There exists a path from $newt_w$ to $v$ in $G(\mathcal{N}, F \cup Ps)$, so $v$ is guaranteed to be enabled by the time it is serviced. □

In addition to identifying redundant alternative arcs, it is also possible to infer compulsory selections for some alternative arcs. In particular, selecting a specific alternative arc $(u, v)$ often implies that some other alternative arcs must also be selected to ensure that the selection remains feasible and does not introduce a positive-length cycle in $G(\mathcal{N}, F \cup Ps)$. Simple conditions can be derived for inferring such compulsory selections, we skip those details here. When selecting an alternative arc $(u, v)$ for inclusion in the set $Ps$, we can infer that:

- If there exists a $c = (u, rob_u, v, rob_v) \in \mathbf{C}$, then from its alternative arc set

$\mathcal{A}_c = \{(next_u, v), (next_v, u)\}$ arc $(next_u, v)$ must be selected, because selecting $(next_v, u)$ would induce the positive length cycle $u \to v \to next_v \to u$.

By similar reasoning:

- If $c = (u, rob_u, w, rob_w)$ with $rob_w = rob_v$ and $pos(w) > pos(v)$, then arc $(next_u, w)$ must be selected from $\mathcal{A}_c$.

• Let $prev_v$ be the node previous to $v$ in sequence $Seq_{rob_v}$. If $c = (w, rob_w, prev_v, rob_v)$ where $rob_w = rob_u$ and $pos(w) < pos(u)$, then arc $(next_w, prev_v)$ must be selected.

The AG filtering techniques for removal of redundant constraints and for selection of unconditional alternative arcs, are applied by the SSp solver in a preprocessing step, prior to conducting the DFS. This preprocessing step terminates only when no more filtering and arc selections can be made. If at any stage during the preprocessing step, the graph $G(\mathcal{N}, F \cup Ps)$ contains a positive-length cycle, the scheduler terminates since the instance is infeasible.

## 5.6 Initial solution generator

Having presented the SSp solver, we turn our attention to the other components making up the MRSBE metaheuristic procedure. In this section, we summarize the approach taken to seeding the metaheuristic local search procedure with a feasible solution.

We generate an initial feasible solution though use of a constructive heuristic that is randomized via Value Biased Stochastic Sampling (VBSS) [Cicirello and Smith, 2005]. VBSS assumes that the heuristic adopted is a good one (although not infallible), and thus modulates the degree of randomness introduced to make any given choice to the heuristic's discriminatory power in that decision context. In the procedure to be described, the degree of randomness is controlled by a single positive parameter called the weight factor, denoted by $\mathbf{wt}$.

The search starts with a set of empty robot itineraries, and each robot residing at its the starting location. In the first step, task locations are assigned to the robots randomly, with the probability of assigning a task location to a robot proportional to $\exp(-\mathbf{wt}\|d\|)$, where $\|d\|$ is the distance of the task location from the start location of the robot. If in case a task location is not reachable by the robot, then the corresponding probability of being selected is set to 0.

The task locations assigned to the robots are then sequenced in an iterative manner. At each iteration for each robot, one of the task locations assigned to it that is not yet sequenced is appended to the end of the robot's partial route. Only those task locations that have at least one of its enablers previously sequenced by some robot are considered as candidates for insertion. A task location is selected for insertion with probability $\propto \exp(-\mathbf{wt}\|d\|)$, where $\|d\|$ is the distance between the task in question and the last location in the robot's partial route. If there are multiple selections of near-equal distance from a given robot's last selection, then the choice is made with greater randomization. If, however, there is a particular selection for a given robot that is much closer than all other alternatives, then the decision is made more deterministically. This process continues until all task locations have been sequenced, after which the SSp is solved to determine feasibility of the resulting routes. Since there is no guarantee of feasibility when all scheduling constraints are considered, this constructive search is repeated until a feasible initial solution is found. As generation of a feasible set of robot tours is NP-Hard, it can take several iterations to establish an initial feasible solution.

## 5.7 Deterministic moves for local search

We define two complementary heuristic moves for improving a given feasible solution. The first, called RELOCATE, selects a task location from some robot's current tour for relocation and either places it in a different position within the same tour or inserts it into the tour of a different robot. The second move, called REORDER, selects a particular robot and locally optimizes the order in which the robot visits the task locations currently assigned to it. Both moves assume as input a set of valid robot tours for which the corresponding SSp problem has previously been determined to be feasible, and with the assistance of the SSp solver, both moves output a feasible solution. The first move, RELOCATE, attempts to improve the schedulability of the tours; the second move, REORDER, emphasizes routing improvements by locally optimizing the sequence of locations visited by some robot. Both these moves are designed to produce new tours that are more likely to be feasible than would be the case with simple randomized moves.

### 5.7.1 RELOCATE move

The RELOCATE move is inspired by previously developed local search operators for exchanging tasks between tours [Van Breedam, 1994]. Each RELOCATE move consists of the following procedural steps. In the first step, a set of candidate task locations is identified for *relocation*, and the candidates are ranked according to some metric. Then, starting from the highest ranked candidate, we try to relocate the candidate and solve the SSp problem corresponding to the modified routes that have been generated. If the makespan of the new solution generated is accepted by the Late Acceptance meta-heuristic, the RELOCATE move successfully ends here for the current iteration. Otherwise, we try to relocate the next highest ranked candidate, and process repeats until all candidates are exhausted. In the following subsections we discuss which task locations are selected as candidates for relocation, how we decide where to attempt insertion of the candidate, and finally how to obtain a feasible schedule (assuming one exists) for the modified tours after insertion.

#### Candidate nodes for RELOCATION

Let $R_{Seq} = \{Seq_1, \ldots, Seq_M\}$ denote the set of robot tours given as input to RELOCATE at the current iteration of our meta-heuristic search procedure. Let $Sch$ be the feasible schedule for $R_{Seq}$ previously computed, and let $G$ denote the AG containing a complete consistent selection for SSp instance corresponding to $R_{Seq}$ and schedule $Sch$. To select a candidate node $h_{rel} \in \mathbf{H}$ for relocation, only nodes in the current *Critical Path* of $G$ are considered in this work. Given the presence of ECs, we rank these nodes based on their movement flexibility, which for node $v$ is defined as: $\underset{\{w|v\in\boldsymbol{\mathcal{E}}_w, v\rightarrow w\}}{min}(\infty, S_w) - \underset{\{x|x\in\boldsymbol{\mathcal{E}}_v, x\rightarrow v\}}{max}(S_{next_x}, 0)$ , where, $\rightarrow$ designates an active EC in $Sch$. $v \rightarrow w$ is said to be active, if among all enablers of $w$, the start time of $v$ in $Sch$ is least. $S_{next_x}$ denotes completion time of $x$ in $Sch$. We rank all task location nodes on the critical path based on the movement flexibility, with a location having higher movement flexibility ranked higher. Intuitively movement flexibility gives us a

conservative estimate of how much temporal flexibility we have in translating the start time of $h_{rel}$ without causing disruption to previously computed active ECs.

### Determining a location for insertion

Once $h_{rel}$ has been selected for relocation in the previous step, let $R^{'}_{Seq}$ denote the set of robot tours after $h_{rel}$ has been removed from $R_{Seq}$. The task of selecting an insertion location for $h_{rel}$, is equivalent to identifying a robot $r$ and successive task locations (say $h_1, h_2$) on the tour of $r$ in $R^{'}_{Seq}$ between which $h_{rel}$ can be inserted. Let $R_{NewSeq}$ denote the set of robot tours obtained after inserting $h_{rel}$ into $R^{'}_{Seq}$. In the following subsection where we show how we generate a feasible schedule for $R_{NewSeq}$, it will be mentioned that the first step involves generating a feasible schedule for $R^{'}_{Seq}$. Denote the feasible schedule to be computed for $R^{'}_{Seq}$ by $Sch'$. Our approach to identifying $h_1, h_2$ for inserting $h_{rel}$ is motivated from CSP scheduling research [Rubinstein *et al.*, 2012], where the value associated with some insertion location $h_1, h_2$ is measured by the slack associated with $h_1$ in $Sch'$. Intuitively, slack is the duration by which the processing duration of $h_1$ can be increased without increasing the makespan. So, selecting $h_1$ with the highest amount of slack is a sensible choice for insertion.

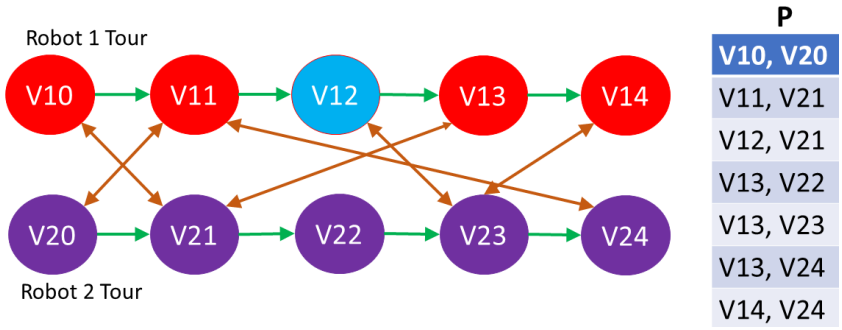
### Confirming feasibility of new tours

After relocating a node, the SSp instance corresponding to $R_{NewSeq}$ must be solved. Obviously, this could be accomplished through the scheduler outlined in Section 5.5, but we can actually exploit the close similarity between $R_{Seq}$ and $R_{NewSeq}$ to define a cheaper feasibility check. Specifically, most of the selections made in the AG $G$ for $R_{Seq}$ and $Sch$ can be reused, while repairing just the portions affected by the relocation.

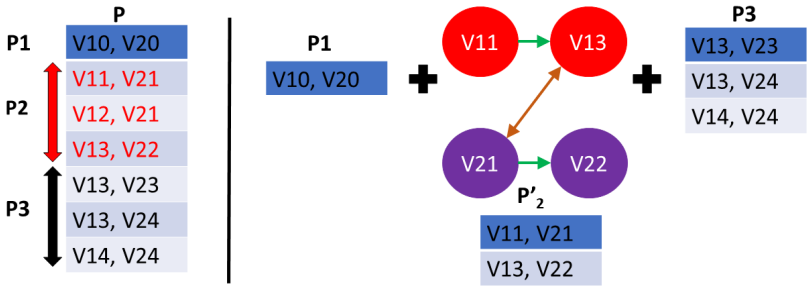
Our incremental procedure for generating a schedule proceeds in two steps, as $h_{rel}$ is retracted and then reinserted. Let $P$ denote the complete STP corresponding to schedule $Sch$ for $R_{Seq}$. When $h_{rel}$ is retracted, $P$ is used to establish whether the SSp instance corresponding to $R^{'}_{Seq}$ is feasible. If it is determined to be infeasible, then candidate $h_{rel}$ is rejected and RELOCATE proceeds to consider the next identified move candidate. Otherwise, a feasible schedule for $R^{'}_{Seq}$ is generated. Let $P'$ denote the complete STP corresponding to this feasible schedule.

After subsequently inserting $h_{rel}$ into its new location, $P'$ is used to determine the feasibility status of $R_{NewSeq}$. As before, if $R_{NewSeq}$ is found to be infeasible, candidate $h_{rel}$ is rejected and RELOCATE proceeds to the next candidate. Otherwise, a feasible schedule is generated for $R_{NewSeq}$, and a complete STP $P_{NewSeq}$ is returned.

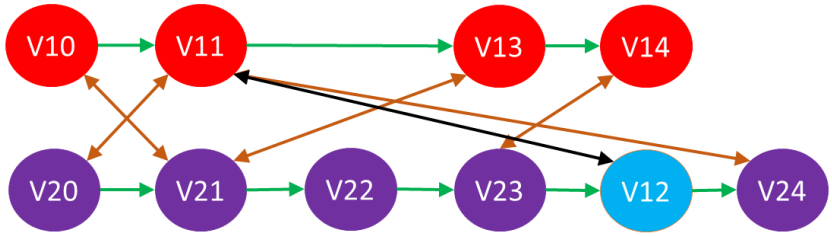
To construct $P^{'}$ from $P$ (and subsequently to construct $P_{NewSeq}$ from $P^{'}$), we focus on partitioning $P$ into three sub-sequences: $P_1$, corresponding to the segment of $Sch$ preceding $h_{rel}$'s original location that is unaffected by the move, $P_2$, corresponding to the segment of $Sch$ that has been disrupted by the retraction of $h_{rel}$ and must be resolved, and $P_3$, corresponding to the segment of $Sch$ following $h_{rel}$'s original location that is unaffected by the move. We hypothesize the smallest possible $P_2$ SSp whose solution could confirm the feasibility of $P^{'}$, and then iteratively expand the number of states contained in $P_2$ as necessary to establish feasibility or infeasibility.



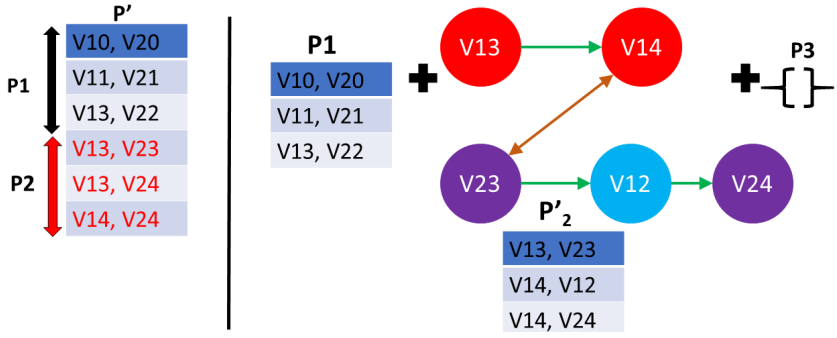

**(a)** Retraction Example: Brown double arrows indicate CCs. Node $V12$ is chosen for relocation.




**(b)** Feasibility certificate for retraction. The smaller SSp instance is shown graphically in the middle.




**(c)** The black double arrow indicates a CC that was absent between nodes in $R_{Seq}$.




**(d)** The concatenation of $P1, P_2', P_3$ is the feasibility certificate for $R_{NewSeq}$.

**Figure 5.7:** RELOCATE example

Consider the two robot example shown in Figure 5.7a, where $R_{Seq}$ and STP $P$ are graphically represented, and node $V12$ indicated in blue is chosen for relocation. Assume for now there is no active EC of the form $V12 \rightarrow \cdot$ in $G$. To determine feasibility of $R'_{Seq}$, we first construct a small SSp instance containing only a subset of nodes appearing in $R'_{Seq}$. If this SSp instance is feasible, we stitch the state transition path of the smaller SSp instance into $P$ such that, the resulting state transition path is a complete STP for $R'_{Seq}$.

To formulate the initial SSp, a continuous block of states $P_2$ in $P$ is selected such that all states in which $V12$ appears in $P$ are contained within $P_2$. $P_2$ is said to satisfy the containment requirement of $V12$. Denote the block of states on $P$ which occur before the first state in $P_2$ by $P_1$, denote the block of states on $P$ which occur after the last state on $P_2$ as $P_3$. Construct an SSp instance induced only by the nodes appearing in $P_2$ other than $V12$ with the following additional modifications:

• If the node appears in both $P_1$ and $P_2$ then its processing time in the SSp instance should be set to the processing time remaining at the start time of the first state in $P_2$.

• Nodes which have an enabler that does not appear in any state belonging to $P_2, P_3$ are considered to be enabled. Hence ECs for such nodes are removed from the SSp formulation.

We apply our SSp scheduler to this small-sized $P_2$ SSp instance to determine the feasibility status. Depending on the outcome we consider two cases. If feasible, let $P'_2$ denote the STP of the SSp instance constructed and denote the concatenation of $P_1, P'_2, P_3$ by $P_{feas}$. Note that $P_{feas}$ is a certificate of feasibility for $R'_{Seq}$ in this case as it is a complete STP. We illustrate this part for our example in Figure 5.7b.

If alternatively the status is infeasible, we consider a larger block of states for $P_2$ satisfying containment of $V12$ and repeat the process of obtaining a feasibility certificate for $R'_{Seq}$. By increasing the size of $P_2$ we are considering a larger (state) search space, the hope being that a feasible solution might be found in the larger search space.

The assumption that there was no active EC of the form $V12 \rightarrow \cdot$ in $Sch$ is needed to assert that $P_{feas}$ is a feasibility certificate for $R'_{Seq}$. If this assumption is not satisfied, then some states in $P_{feas}$ may violate some EC. If there exists a violated EC, then if we presume the existence of a feasible schedule for $R'_{Seq}$, we can obtain one such schedule by progressively increasing the block size of $P_2$ (correspondingly shrinking $P_1$ and $P_3$) and recomputing $P'_2$ until we obtain a feasibility certificate $P_{feas}$ that does not violate any EC. In our actual implementation we initially start with a block size for $P_2$ that is somewhat larger than the smallest STP block satisfying containment, and we terminate the removal step if the block size for $P_2$ exceeds a user defined threshold, since solving SSp instances corresponding to large block sizes is expensive and defeats the purpose of local search.

The insertion step is exactly analogous to the retraction step, in this case using $P'$ to construct $P_{NewSeq}$. Given a robot and a position within its tour to insert $h_{rel}$ as input, the containment requirement for $P_2$ in the insertion step is that the nodes within which the insertion takes place must each appear in some state of $P_2$. Continuing with our earlier example, we chose to insert $V12$ between $V23$ and $V24$. Figures 5.7c and 5.7d shows the SSp problem that is solved and the complete STP generated after insertion.

### 5.7.2 REORDER move

The REORDER move is designed to focus on the routing component of the problem. Similar to the RELOCATE move, the REORDER move takes as input a set of robot tours $R_{Seq}$ and a feasible schedule $Sch$ for those tours. The REORDER move selects a robot $r$, and reorders its visit sequence of task locations while simultaneously keeping the sequences of all other robots fixed and all the interdependent scheduling constraints satisfied.

REORDER is based on a Traveling Salesperson Problem (TSP) heuristic of [Balas, 1999] which we modified to accommodate ECs and CCs. We first state the TSP heuristic of Balas exactly as it appeared in Balas [1999]:
*Consider the n-city TSP defined on a complete directed or undirected graph, and fix city 1 as the home city, where all tours start and end. Suppose now that we are given an integer* $\mathbf{k}$*,* $1 \leq \mathbf{k} < n$*, and an ordering* $(1, \ldots, n)$ *of the set N of cities, and we want to find a minimum cost permutation* $\pi$ *of* $(1, \ldots, n)$ *(and associated tour) subject to the condition:*

$$\forall\, i, j \in (1, \ldots, n)\,,\; j \geq i + \mathbf{k} \implies \pi(i) < \pi(j) \tag{5.17}$$

*then the TSP with condition in Eqn (5.17) can be solved in time* $\mathcal{O}\left(\mathbf{k}^2 2^{\mathbf{k}-2} n\right)$ *.*

Remarkably, the TSP heuristic searches a neighborhood of size $\mathcal{O}\left(\mathbf{k}^n\right)$ in time complexity which depends only linearly on $n$. We chose to adopt this heuristic due to its attractive complexity [Balas, 1999; Simonetti and Balas, 1996], and the ease of adaptation for our application.

#### Adapting the TSP heuristic for REORDER

Denote the tour of robot $r$ in $R_{Seq}$ by $Seq_r$. The task locations belonging to $\mathbf{H_r}$ (refer Section 5.1 for definition) that occur on $Seq_r$ take the role of cities in the description of the TSP heuristic. Processing time of nodes belonging to $\mathbf{I_r}$ that occur between task locations define the cost between cities. For a given $\mathbf{k}$, the routing heuristic of Balas reorders the sequence of task locations in $Seq_r$ subject to the condition in Eqn (5.17). We cannot guarantee that replacing $Seq_r$ in $R_{Seq}$ by the tour generated by applying Balas's heuristic on $Seq_r$ will admit a feasible schedule. To increase the likelihood of generating a feasible schedule, we must account for the inter-dependencies between robots (CCs and ECs) while generating the new tour for robot $r$.

A natural idea to account for the inter-robot dependencies is to generate constraints based on the schedules available in $Sch$ for robots other than $r$, and incorporate them into the optimization procedure of the TSP heuristic. In other words, instead of solving a purely routing problem to generate the new tour for $r$, we will solve a routing problem with additional constraints included, where some constraints are temporal and others are precedence. In the following section, we identify these constraints and explain modifications to Balas's procedure to handle these constraints.

#### Constraints for inter-robot dependencies

Let $Seq_r^{'}$ denote the new tour to be generated for $r$ by reordering the task locations in $Seq_r$. To incorporate temporal constraints during the optimization procedure for generating $Seq_r^{'}$, we

need to introduce variables for the start times of nodes in $Seq_r^{'}$. Let us denote the schedule to be determined for $Seq_r^{'}$ by $Sch_r^{'}$. The inter-robot dependency constraints that we introduce are:

***Collision free time intervals:*** For each candidate node $u$ that must appear in $Seq_r^{'}$ in case $u \in \mathbf{H_r}$ or can appear in case $u \in \mathbf{I_r}$ (refer Section 5.1 for definition), compute the union of collision free time intervals ($CF_u$) based on the schedules available for the other robots. Let $S_u$ denote the start time of $u$ in $Sch_r^{'}$. The intervals in $CF_u$ represent the duration within which we can assign a value to $S_u$ with the assurance that $r$ will not collide with other robots for the entire processing duration of $u$, i.e., the interval $[S_u, S_u + p_u^r]$.

***Service time deadline:*** In the previous MRSBE schedule $Sch$, it may be the case that robots in $\mathbf{R}\backslash r$ contain nodes which are enabled by a task location in $Seq_r$. In this work, we prefer to preserve the enabling relationship, so a deadline can be imposed for the completion time of the enabler node using the start time of the node it enables. Denote the deadline for a node $u$ by $Dl_u$.

***Enabling constraint:*** The enablers for any task location in $Seq_r$ may occur in $Seq_r$ itself or be present in the tour of a different robot. For instance, if locations $u, v, w \in Seq_r$, $x \notin Seq_r$ and $\boldsymbol{\mathcal{E}}_u = \{v, w, x\}$, then the following disjunctive constraint can be imposed:

$$(\pi(u) > \min(\pi(v), \pi(w))) \vee \left(S_u \geq \mathbf{S}^{\mathbf{Sch}}_{\mathbf{next_x}}\right) \tag{5.18}$$

where $\pi(\cdot)$ represents the position of the location in $Seq_r^{'}$, $S_u$ represents the start time of $u$ in $Sch_r^{'}$ and $\mathbf{S}^{\mathbf{Sch}}_{\mathbf{next_x}}$ is the completion time of $x$ in $Sch$. Note, $\pi(\cdot)$ and $S.$ are unknowns that need to be determined during the procedure for generating $Seq_r^{'}, Sch_r^{'}$.

### Handling inter-robot dependency constraints

We first briefly describe Balas's procedure which takes $Seq_r$ as input and outputs the optimal tour satisfying Eqn (5.17). We then describe modifications to Balas's procedure in order to handle the inter-robot dependency constraints. Assume without loss of generality that the task locations in $Seq_r$ are given by the sequence of locations $(1, \ldots, n)$. Balas's heuristic defines a DAG $G^*$. $G^*$ contains a 1:1 correspondence between source-sink paths in $G^*$ and permutations of $Seq_r$ satisfying Eqn (5.17). Every node in layer $i$ of $G^*$ can be characterized by a tuple $(i, j, Q^-_{ijm}, Q^+_{ijm})$, where $j$ is a location and $Q^-_{ijm}, Q^+_{ijm}$ are a set of locations both not containing $j$ such that $|Q^-_{ijm}| = |Q^+_{ijm}| \leq |\frac{\mathbf{k}}{2}|$; the precise definition of $Q^-_{ijm}, Q^+_{ijm}$ is involved. It suffices to know that all source-sink paths in $G^*$ corresponding to tours satisfying Eqn (5.17) with location $j$ at position $i$ in the tour and locations in the set $\left([i-1]\setminus Q^+_{ijm}\right) \cup Q^-_{ijm}$ occupying all positions before $i$ in the tour pass through the node $(i, j, Q^-_{ijm}, Q^+_{ijm})$, where $[i-1] = \{1, 2, \ldots, i-1\}$.

The shortest tour satisfying Eqn (5.17) corresponds to the sequence of task locations occurring on the shortest source-sink path in $G^*$. The shortest path is computed by recursively computing the shortest path to each node in $G^*$ in a layer wise fashion. The shortest path cost to node $(i, j, Q^-_{ijm}, Q^+_{ijm})$ on $G^*$ is to be interpreted as the earliest start time of location $j$ at position $i$ on any robot tour satisfying Eqn (5.17) with task locations in the set $\left([i-1]\setminus Q^+_{ijm}\right) \cup Q^-_{ijm}$ occupying all positions before $i$.

Handling inter-robot dependencies while generating the optimal tour essentially reduces to increasing arc costs of $G^*$ dynamically. Consider some node $n_2 = (i, v, Q^-_{ivm_2}, Q^+_{ivm_2})$ in layer $i$ of $G^*$. Recall from the previous paragraph that the shortest path cost to node $n_2$ provides a start time value for location $v$. If the start time of location $v$ needs to satisfy the constraints generated in section 5.7.2, the length of every incoming arc $a$ into node $n_2$ is increased by the smallest amount such that the shortest path cost to node $n_2$ through arc $a$ satisfies the constraints generated for location $v$ in section 5.7.2. To be collision free, the new arc length of $a$ must be set such that the path cost to $n_2$ through $a$ falls within an interval of $CF_v$ and does not exceed $Dl_v$. To check whether the enabling constraint for $v$ is satisfied at node $n_2$, we first check if an enabler of $v$ appears in the set $\left([i-1] \setminus Q^+_{ivm_2}\right) \cup Q^-_{ivm_2}$. This check can be computed in $\mathcal{O}(\mathbf{k} + |En_v|)$, through the use of pre-computed sorted lists. If an enabler is present then Eqn (5.18) is satisfied since an enabler of $v$ is serviced by robot $r$ before servicing $v$, if not the length of $a$ needs to be increased such that the path cost to $n_2$ through $a$ satisfies the temporal constraint in Eqn (5.18). If no feasible arc length exists that satisfies our requirements above we remove $a$ from $G^*$. This idea of dynamically increasing the arc cost to handle temporal constraints was previously also done in Balas *et al.* [2008], where the TSP heuristic of Balas was adapted to solve job shop problems with time windows.

After computing the shortest path to all nodes in $G^*$ with the modified arc costs, the new tour for $r$ and $Sch_r^{'}$ is extracted from the shortest source-sink path in $G^*$. Replacing $Sch_r$ with $Sch_r^{'}$ in $Sch$ is not guaranteed to yield a feasible schedule to the MRSBE problem. This is because, in our definition of collision free interval ($CF.$) for a node that may appear in $Seq_r^{'}$, the intervals guaranteed no collisions only for the processing duration of the node if the start time for that node falls within one of its intervals, which however does not include any waiting time a robot incurs at that node before it can move to the next node in $Seq_r^{'}$. While the robot waits a collision can still occur. So we discard $Sch_r^{'}$ and only retain $Seq_r^{'}$. We replace the tour for $r$ in $R_{Seq}$ by $Seq_r^{'}$ and solve the corresponding SSp instance to generate a feasible schedule (if one exists).

## 5.8 Experimental evaluation

In Section 5.8.1 we describe the procedure used to synthesize problem instances that match closely with our real world problem, and introduce a closely related application. In Section 5.8.3, we report the performance of our heuristic on those instances.

### 5.8.1 Benchmark data

#### Two robot benchmark data

We generated four benchmark sets, covering a total of 80 instances. All instances have two robots. Robots are represented as rectangular polygons attached to a pivot point, similar to the case shown in Figure 5.1. This abstraction resembles closely the motivating application shown in Figure 5.1a. For checking if a collision occurs when robot $i$ services node $v$ and robot $j$ services node $w$ simultaneously, we approximate the space occupied by the robots using poly-

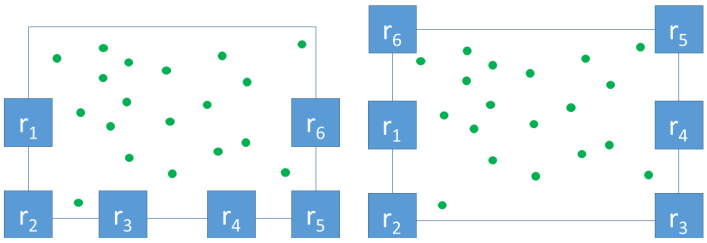



**Figure 5.8:** The figure on the left is for scenario type 1. The large rectangle is the tennis court, the green circles represent tennis balls. The initial (and final) position for the robots $r_1, \ldots, r_6$ are shown. The figure on the right is likewise for scenario type 2.

gons and check whether they both intersect. The total number of CCs in our problems roughly scales as $\mathcal{O}(M^2N^4)$, precomputing all the CCs is intractable for most reasonably sized instances. On our test instances, $N$ varies between $44 - 764$. So instead of pre-computation, CCs were generated by performing collision checks prior to solving each SSp instance separately. A single *IN* is encountered between a pair of task locations (resulting in $550590$ *INs* for the largest instance). A small subset of task locations $T \subset \mathbf{H}$, $|T| = 15$, is marked as enabled, i.e., $\boldsymbol{\mathcal{E}}_v = \emptyset$; the remaining locations $\mathbf{H} \setminus T$ are enabled by task locations in proximity (within a small radius).

Each of the four benchmark sets exhibits different characteristics. The first set, *'Real'*, is derived from data provided by our partner in the aerospace industry and contains instances having $44 - 764$ task locations that are arranged in rows and columns (grid structure), are enabled by nearby adjacent task locations, and have processing times that dominate the travel times between them. For tours (SSp instances) generated by VBSS in the initial solution generator module, we observed that the largest real world instance on average contained $\approx 22 \times 10^3$ CCs per SSp instance.

The remaining three benchmark sets generate $100$ task locations for each instance by randomly sampling points within a rectangular region. The first random set, *Random*, is identical to the *Real* set, except for the randomness in the locations. The second random set, *Low Proc*, has long travel times between locations, and short processing times $p_i^r$, thereby putting a stronger emphasis on the routing component. Finally, the problems in the third random set, *Perturb* are similar to the *Random* set, except that the distances between task locations are randomly perturbed after enablers are generated to force overlaps in tours and increase the likelihood of collisions. The purpose of additional three benchmark sets is to analyze the performance of our MRSBE heuristic on problems with different characteristics, thereby providing a differential analysis of the heuristic.

### Four and six robot benchmark data

These problems help understand how our algorithm scales with more robots. A motivating application for creating these instances is a tennis ball picking problem. Say we have a $6 \times 4$ sized rectangular tennis court, with many balls littered across the court. These balls need to be picked which can be done by any of the robots. The problem is similar to our real world problem, except there are no ECs. Like in Section 5.8.1, we modeled the mobile robots

using rectangles of dimension $1 \times 1$. The robots can traverse between task locations along the straight line connecting those locations. CCs were generated identically to the procedure in Section 5.8.1.

A total of ninety-six instances were generated with each instance having 100 tennis balls (task locations) randomly distributed across the tennis court. Forty-eight of these instances have six robots, and the remaining have four robots. The initial locations of the robots may have a significant impact on the performance of an algorithm for the MRSBE problem. So we experimented with two different types of scenarios to study the impact of the initial position of the robots. The 96 test instances can be classified as belonging to one of these two scenarios. In the first scenario, the starting locations for the robots are closer to the bottom edge, while in the second scenario the robots are uniformly located across the perimeter of the tennis court. The two scenarios are shown in Figure 5.8. The time it takes a robot to pick a ball (processing time of a task location) is set to a low value as compared to the travel time. For tours (SSp instances) generated by VBSS, we observed that for both four and six robot instances on average the SSp instance generated contained $\approx 10^3$ CCs.

The four and six robot test cases as well as the experimental results can be found online at `https://github.com/jmogali/MRSBE_Test_Cases`.

### 5.8.2 mTSP bound

To obtain insight into the quality of the heuristic MRSBE solutions, we compute lower bounds on the optimal MRSBE solutions. These bounds are obtained by solving a mTSP problem, thereby ignoring scheduling constraints such as the ECs and CCs. In all our experiments, the travel times are symmetric, i.e., the sum of processing times of *INs* from $i$ to $j$ (where $i, j \in \mathbf{O} \cup \mathbf{D} \cup \mathbf{H}$) is identical to sum of processing times of *INs* from $j$ to $i$. So lower bounds can be computed by solving a symmetric version of the mTSP problem, with edge weights adjusted to include processing times of task locations. Our mTSP formulation is modeled as an undirected, weighted graph $G'(V', E)$ with $V' = \mathbf{O} \cup \mathbf{D} \cup \mathbf{H}$ and $E = V' \times V'$. The weight $w_e^r$ of an edge $e = (i, j) \in E$ is set equal to: $w_e^r = \frac{p_i^r + p_j^r}{2} + \sum_{u \in P_{ij}^r} p_u^r$, where the second term sums over the processing times of the *INs* encountered on the trajectory from $i$ to $j$. Given graph $G'$, the mTSP can be formulated as follows:

$$\text{minimize} \ \max_{r \in \mathbf{R}} \sum_{e \in E_r} w_e^r x_e^r \tag{5.19}$$

$$\text{s.t.} \sum_{r \in \mathbf{R_i}} y_i^r = 1 \qquad \forall i \in V' \tag{5.20}$$

$$\sum_{e \in \delta_r(i)} x_e^r = 2 y_i^r \qquad \forall r \in \mathbf{R}, i \in V_r^{'} \tag{5.21}$$

$$\sum_{r \in \mathbf{R}} \sum_{e \in \delta_r(S)} x_e^r \geq 2 \qquad \forall S \subset V', |S| \geq 3 \tag{5.22}$$

$$y_i^r, x_e^r \in \{0, 1\} \qquad \forall r \in \mathbf{R}, i \in V_r^{'}, e \in E_r \tag{5.23}$$

Here binary variables $y_i^r$ (resp. $x_e^r$) assign vertices (resp. edges) to robots. $\delta_r(S)$ is the set of

all edges between vertices in the set $V_r^{'} \backslash S$ and vertices in $S$. The objective function minimizes makespan. Constraint (5.20) assigns every node to a robot. Every location must be incident to exactly two edges (Constraint (5.21)). Constraints (5.22) address subtour elimination. A standard separation procedure is used to separate the subtour elimination constraints. The mTSP model is solved by IBM Cplex 12.8.1, using a time limit. The lower bound value computed by the solver after the time limit is used as the mTSP lower bound for the problem instance.

### 5.8.3 Computational results

Based on empirical tests, the following parameters were selected for the various algorithms. The length of the LA window is set to $25$. The maximum number of consecutive failed perturbation moves before a restart is performed in the MRSBE heuristic is set to $40$. For the 2 robot experiments, the $\mathbf{wt}$ (refer Section 5.6) parameter for VBSS within the constructive procedure was set to a value which resulted in the best performance over "Real" dataset instances. For the 4 and 6 robot experiments, we present results with different values of $\mathbf{wt}$ parameter.

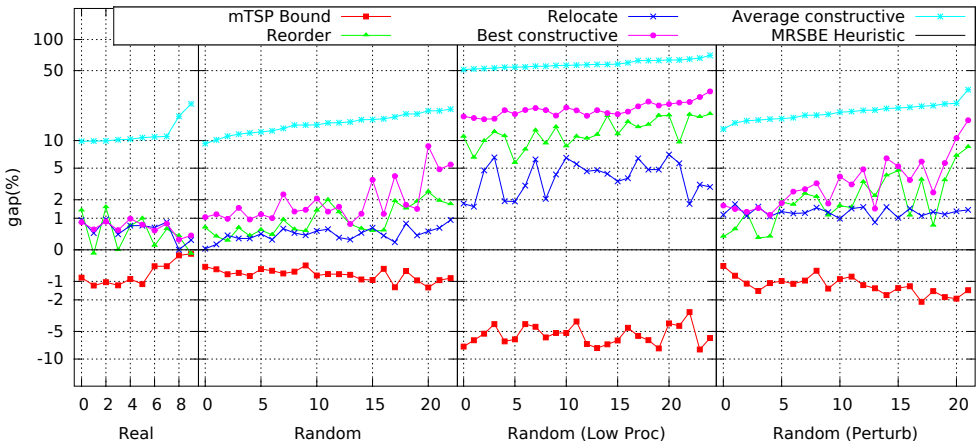
In each iteration of the MRSBE heuristic, we randomly select a Deterministic Neighborhood. The RELOCATE (resp. REORDER) neighborhood is selected with probability $\frac{2}{3}$ (resp. $\frac{1}{3}$). The choice of these probabilities is motivated by the fact that the REORDER move is generally computationally more expensive than the RELOCATE move. For computational reasons, the maximum length of the block $P_2$ considered in RELOCATE was set to 30. The $\mathbf{k}$ parameter in REORDER was set to 12. The collision free time intervals (refer $CF.$) for nodes in $\mathbf{I_r}$ were set to the single time interval $[-\infty, +\infty]$. In other words, we do not take into consideration the collision constraints for $\mathbf{I_r}$ when applying REORDER. We did this simplification because, the number of nodes in $\mathbf{I_r}$ for which we need to compute the collision free time intervals are too many. In cases when the processing times dominate travel times, we should not expect to lose too much information by ignoring exact computation of those collision free time intervals for nodes in $\mathbf{I_r}$.

We set a time limit of $5$ seconds for the SSp scheduler to solve a single SSp instance. If the scheduler failed to return a solution within the time limit, we simply declared the SSp instance to be infeasible. All experiments were carried out on an Intel i7-4790 3.6 GHz processor using single thread, and the code was written in C++.

#### Results for the two robot case

To evaluate the performance of the MRSBE heuristic, various experiments are conducted using different configurations of the heuristic. In the first experiment, we repeatedly compute initial solutions using the Constructive heuristic (described in Section 5.6). For each instance, the best and average constructive solution were recorded over a time period of 10 minutes. In the second experiment, we ran the complete MRSBE heuristic for 10 minutes (on each problem instance) and compared the solutions against the mTSP bounds, where the solve time for mTSP was set to $30$ minutes. Finally two more tests were conducted to determine the influence of REORDER and RELOCATE on the overall heuristic. In the first experiment, the MRSBE heuristic was run without RELOCATE move; in the second, it was run without RE-

**Figure 5.9:** Computational results of the four benchmark classes for 2 robot case. To fit the results compactly in one graph, a log-modulus transformation ($L(y) = \text{sgn}(y) \times \log(|y| + 1)$ ( [John and Draper, 1980])) is applied to the y-values.




ORDER move. In both these tests, the time limit was set to 10 minutes per problem instance. Computational results using the CP model (Algorithm 12) are omitted since the mTSP solver produced better lower bounds and the MRSBE heuristic produced better upper bounds on all instances.

A summary of the computational experiments is provided in Figure 5.9. The problem instances for each of the 4 benchmark sets are listed on the x-axis. The results of the complete MRSBE Heuristic (black line) are taken as the base line for all the experiments; other results are reported *relative* to this baseline (percentage gap).

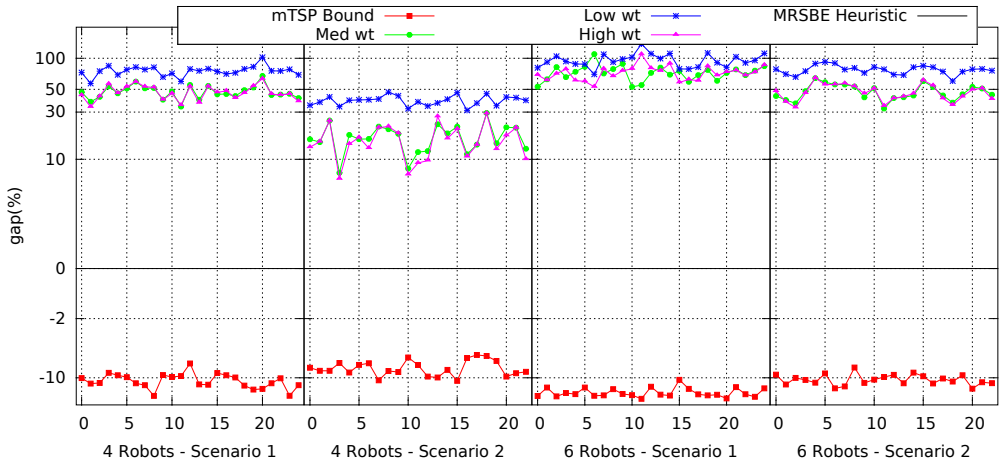
When comparing the mTSP bound to the MRSBE Heuristic, we can conclude that the MRSBE Heuristic obtains very good results, and that the mTSP provides strong lower bounds. For a given problem instance, we computed the optimality gap as:

$$\text{Optimality Gap } \% = \frac{\text{MRSBE heuristic - mTSP lower bound}}{\text{mTSP lower bound}} \times 100$$

For three benchmark sets, the optimality gap is less than 2%. Only for the *Low Proc* benchmark, this gap is slightly bigger (between 3-10%). It is not surprising that the gap is larger for this set: since we omitted computing the collision free time intervals for *INs* as mentioned in Section 5.8.3 for computational reasons. As a result, the REORDER works with a less informed model when computing the new sequence of visit task locations.

Similarly, we can observe that the MRSBE Heuristic improves dramatically over the initial solutions. For the *Real* instances, the gap between the average initial solution and the MRSBE Heuristic solution is in the range of 10-20%. However, for the *Perturb* and *Low Proc* benchmarks, improvements in the range of 20-60% are witnessed. When comparing the impact of the REORDER and RELOCATE moves on the overall heuristic, some interesting observations can be made. From the *Random*, *Real* and *Perturb* benchmarks we can conclude that neither move dominates the other move, and that they both complement each other, i.e., the MRSBE heuristic utilizing both moves achieves better results than a variation of the heuristic where

**Figure 5.10:** Computational results for the multi robot cases under different scenarios.




one of the moves is left out.

Only for the *Low Proc* benchmark, where the emphasis is on the travel times, the heuristic variant including the RELOCATE move seems to dominate the variant with only the REORDER move. This difference may be due to the fact that we ignored computation of collision free time windows for travel duration, and also may be due in part to the computation time required by each move. RELOCATE is a magnitude faster than REORDER, allowing for many more iterations to explore the search space in the allotted ten minutes of execution. Furthermore, in contrast to the RELOCATE move, REORDER moves only affect the routes of an individual robot, which makes it harder for the REORDER move alone to reach different areas of the search space. The results suggest that the REORDER move seems to be primarily effective when used in combination with another move, e.g., the RELOCATE move, as the combination of the two moves yields the best results overall.

### Results on 4 and 6 robot cases

For the multi-robot case, for each multiplicity of robot and scenario type, we performed 5 experiments. In the first experiment, we computed the mTSP lower bound, where solve time for mTSP was set to 2 hours for each problem. In the second experiment, we ran the MRSBE heuristic for 10 minutes on each problem instance. Observe that depending on the topology of the problem, i.e., placement of initial robot locations and task locations, choosing a single value for the $\mathbf{wt}$ parameter for all the different topologies is not likely to yield good overall performance. So in experiments 3-5, we ran the Constructive procedure with 3 different values of $\mathbf{wt}$ parameter. The $\mathbf{wt}$ parameters for the experiments was chosen in the ratio $1:5:10$, where recall higher the $\mathbf{wt}$ value, the more greedily VBSS performs. For convenience, we shall refer to the 3 $\mathbf{wt}$ parameter values as *Low, Med, High*. In these experiments, a time limit of 10 minutes was set for the constructive procedure on each problem instance, and the best objective across all iterations was recorded. Similar to the two robot case, we take the output of the MRSBE heuristic as the baseline, and the outputs of the remaining four experiments are compared against this baseline. The results are summarized in Figure 5.10. Also, note that the

wt parameter was set to *Med* value, when the constructive procedure was required to provide seed solutions to the MRSBE heuristic.

From Figure 5.10, we can observe that for the 4 robot case, the average gap between MRSBE heuristic and the mTSP lower bound is $\approx 12\%$ for problems of scenario type 1, and $\approx 9\%$ for scenario 2. For the six robot case, the average gap for scenario type 1 is $\approx 15\%$, and that of scenario type 2 is $\approx 11\%$. These numbers indicate as the number of robots increase, our gaps marginally increase. Also, interestingly, the gap between mTSP solution and MRSBE heuristic is larger for problems in scenario 1 as compared to scenario 2. As the robots in scenario-1 are less uniformly distributed across the perimeter of the tennis court, we can expect CCs to play a more significant role in shaping the schedules for instances in scenario-1 as opposed to scenario-2. Our mTSP bound however completely neglects all CCs, hence gaps are larger for scenario-1.

From Figure 5.10, we clearly see that the gap between the MRSBE heuristic solution and the constructive solutions are substantial. For each problem instance, we took the best objective among the three constructive solutions, and report the difference between it and the MRSBE solution. For the 4 robot case and scenario 1, on average, the gap when measured relative to the MRSBE solution is $\approx 46\%$, and for scenario 2 is $\approx 16\%$. For the 6 robot case and scenario 1, the corresponding average gap for scenario 1 is $\approx 66\%$, and $\approx 46\%$ for scenario 2. These results indicate the efficacy of our heuristic, as it clearly implies that the MRSBE heuristic is able to substantially improve over the seed solution given by the constructive procedure.

Also, notice that the gap between the best constructive procedure and MRSBE heuristic solution is larger for problems in scenario 1 than those in scenario 2. As we explained earlier, CCs play a more dominant role in shaping the schedule for instances in scenario-1 as opposed to scenario-2. The constructive procedure in Section 5.6, however completely disregards all CCs while generating robot tours, and hence its performance is somewhat inferior for scenario-1.

The results on these 4 and 6 robot instances indicate a slight dependence in the performance of our MRSBE heuristic on the geometry of the instance structure.

### 5.8.4 Alternate approaches

In this Section, we report our computational experience when we tried to solve the MRSBE instances exactly using CP\IP. When we tried to solve the CP formulation provided in description 12, for any but the smallest instance, CP went out of memory (12GB). For perspective, the smallest instances contained $\approx 22 \times 10^6$ CCs. For the smallest instances, CP could not find feasible solutions, unless an initial feasible solution was provided as warm-start. In the latter case, CP could not improve upon the initial solution.

A second CP model, which utilizes Interval Variables, and omits all CCs except the ones between task locations, was constructed to compute lower bounds (LBs). All LBs computed with this second model were inferior to the mTSP bounds that we presented, and so were not included in this work. We also experimented with a MIP model based on a time-indexed formulation for mTSP. To solve the MIP model, a branch and cut framework was used: the model was solved without CCs; whenever a solution was found, it was checked whether the

solution violated any CCs. If so, the violated CCs were added to the model. Similar to CP, this approach was unable to solve any of our realistically sized problem instances.

## 5.9 Related work

There has been a lot of work in the area of task assignment and sequencing in different settings, and the solution approaches are varied. Traditional Operations Research (OR) literature has primarily focused on routing (generalized TSP) and scheduling (job shops, project scheduling, etc.) separately, due to the enormous number of real world applications for each of these problem types. The MRSBE problem however lies at the intersection of both.

A well studied application in the OR literature that considers both routing and collision constraints, much like our application, is the vast literature on the quay crane scheduling problem (QCSP) in port container terminals [Bierwirth and Meisel, 2010]. The QCSP problem is very similar to our problem, where cranes take the role of robots and enabling constraints are replaced with precedence constraints. There are important differences, however. The QCSP is generally not modeled with all blocking constraints, and in practice the number of CCs in QCSP benchmarks turns out to be only a tiny fraction of the CCs considered in our work. For solving the QCSP problem exactly, MILP approaches are currently at the forefront [Kim and Park, 2004]. These MILP formulations are typically solved using branch and cut procedures [Moccia *et al.*, 2006]. While MILP approaches are currently only able to solve small sized instances to optimality, there is also in interest in developing local search approaches for the QCSP problem. One such representative local search approach for this problem is the work of [Sammarra *et al.*, 2007], where a Tabu search metaheuristic method that decomposes the problem into a routing and scheduling problem is proposed. They model the QCSP problem as a parallel uniform machine scheduling problem with precedence constraints, where cranes take the role of machines. The routing heuristic they implement is task swap, which similar to RELOCATE, swaps tasks between cranes, and the scheduling sub-problem is modeled using a disjunctive graph. Similar to our work, the moves in the neighborhood are based on relocating tasks on the critical path.

A very closely related work to the problem setting addressed in this chapter is that of [Gombolay *et al.*, 2018], where multiple heterogeneous robots working in proximity need to be scheduled to service a set of tasks while avoiding collisions. Two key differences from our work is that they do not model the problem using blocking constraints, and they do not consider enabling constraints. They propose an iterative hybrid approach, where at each iteration the task assignment is delegated to an IP, while sequencing and scheduling of tasks across robots is performed together. The sequencing-scheduling module is fast but incomplete, and draws ideas from real time processor scheduling techniques.

There has been a growing number of applications that require combining task and motion planning. A good review of existing literature on both applications, methodologies and challenges can be found in Mansouri *et al.* [2021]. We next review some of these applications. In Mansouri *et al.* [2016], the authors consider a multi-robot drill planning problem in open pit mines. Similar to our approach, they decompose the problem into different subcomponents, in which task planning, motion planning and coordination are modeled as constraint satisfaction

problems. Interdependencies between these models are captured using meta-constraints. In Mansouri *et al.* [2017], the authors model the mining problem in Mansouri *et al.* [2016] as a multi-vehicle routing problem with nonholonomic constraints and dense obstacles.

Another line of work that is recently gaining popularity is manufacturing using dual-arm robots [Behrens *et al.*, 2019; Wessén *et al.*, 2020]. Similar to our work, the challenges in these applications typically include allocating tasks to each arm, sequencing the tasks, and scheduling the robot arms such that the arms do not collide with each other, while the objective that is typically minimized is makespan. Both [Behrens *et al.*, 2019] and [Wessén *et al.*, 2020] propose a CP based approach for their respective problems. In Behrens *et al.* [2019], a motion series model is proposed to handle the movement of the arms, and conflict tables are pre-computed to handle collision checking. They, however, assume that the task sequence is provided to them. In Wessén *et al.* [2020], in addition to the challenges listed earlier, they also consider the problem of optimizing the layout design of the robot workspace. They explore the concept of dividing the workspace into two parts, which significantly helps simplify their CP model with respect to collision handling. Further, in this application domain, there is also interest in integrating motion planning for the robot arms more tightly into the optimization procedure [Kabir *et al.*, 2020; Behrens *et al.*, 2019].

The Multi-Agent Path Finding (MAPF) problem is yet another problem class that considers robots working in close proximity, that we studied earlier in Chapter 3. Given a start and end location for each robot, the task in this problem is to compute conflict free paths one for each robot, and the objective that is typically minimized is either makespan or sum of travel costs. Solution techniques for MAPF and its variants typically are based on either branch and bound techniques [Sharon *et al.*, 2015], polyhedral techniques involving cutting planes [Mogali *et al.*, 2020] or variants of A* [Wagner and Choset, 2011]. Unlike MRSBE, crucially, MAPF does not require task sequencing since each robot has only one task location to service, i.e., its end location. On the other hand, unlike MAPF, the route between any pair of task locations is pre-specified for each robot in MRSBE using *INs*.

The literature dedicated to blocking constraints has mostly focused on applications where the underlying problem can be modeled as a Job Shop problem or a Flow Shop problem [Mascis and Pacciarelli, 2002]. Typical applications include train scheduling [Lange and Werner, 2018], scheduling and material handling in flexible manufacturing systems [Mati *et al.*, 2011]. Unlike MRSBE, these applications lack a task assignment and sequencing component, and so the problems considered in these works are more comparable to the SSp problem. More recent work in scheduling with blocking constraints has emphasized the use of iterative improvement search procedures. In Groeflin and Klinkert [2009], a new neighborhood structure was coupled with tabu search to produce new best known solutions for a set of reference blocking job shop problems. Building on the work of [Groeflin and Klinkert, 2009], specialized local search neighborhoods have been more recently proposed for solving more general structured scheduling problems [Bürgy, 2017] with blocking requirements. However, the properties of these neighborhoods are not well understood even for those structured problems, see Chapter 4, and so whether they can recover the optimal solution to the SSp problem (assuming feasibility) is currently unknown.

## 5.10 Summary

In this chapter, we introduced and analyzed a new multi-robot planning problem called the Multi-Robot Scheduling Problem with Blocking and Enabling Constraints (MRSBE), where multiple robots are tasked to service an overlapping set of locations to minimize overall makespan, while taking both enabling constraints (ECs) and collision constraints (CCs) into account. We analyzed the complexity of MRSBE and its Scheduling Subproblem (SSp) restriction, and showed that even the problem of determining whether a set of robot tours is feasible is NP-Complete. To solve MRSBE for realistically sized instances, a meta-heuristic approach was developed that utilizes two specialized neighborhood search operators to promote exploration of the feasible solution space.

A set of benchmark problems were created to assess the performance of the approach, including one set of benchmarks representative of the real-world application that has motivated this research. We compared and reported the performance of our meta-heuristic against a strong lower bound and as well as an alternative method for generating upper bound solutions.

## 5.11 Summary of contributions

- We introduced a new class of multi-robot routing and scheduling problem.
- We studied the complexity of the MRSBE problem, and the scheduling sub problem where routes are specified for each robot, i.e., the SSp sub problem.
- We presented a scheduler for the SSp problem.
- We presented a meta-heuristic scheme, and developed two complementary local search operators. One of the operators is guided by the scheduling aspect of the problem, while the other emphasizes the routing aspect of the problem.

# Part IV

# Chapter 6

# Conclusions

Problems with a spatio-temporal flavor occur frequently in industrial environments. In this thesis, we presented heuristics for three such problems, namely, Multi-agent Path Finding (MAPF), Blocking Job Shop (BJS), and Multi-robot routing and scheduling with collision (Blocking) and generalized precedence (Enabling) constraints (MRSBE). Although, all three problems can be modeled and solved using a single framework, e.g., Mixed-Integer Linear Programming, Constraint Programming etc. We took the position that developing a specialized algorithm for each problem separately often yields better results.

In Chapter 3, we contributed to the MAPF problem by developing a novel cutting plane based lower bounding procedure that combines, Lagrangian Relax-and-Cut with Decision Diagrams. Intuitively, our contribution can be characterized as an approach to analyze the paths of multiple ($\geq$ 2) robots within a spatio-temporal neighborhood. We used that analysis to derive cuts, ultimately leading to strong lower bounds. We demonstrated that incorporating our lower bounding procedure improves the performance of Conflict-based search, a state-of-the-art search based approach for MAPF. The results demonstrated that our approach is more effective in congested settings. On the *Random* layout instances, we reported better results on layouts with 15, 20 and 25 obstacle %. Relative to CBS, the number of problems that we could solve to optimality increased between 4 - 16%. Some ideas presented in this paper, such as the projection based cut generation via Decision Diagrams, may be more broadly applicable to other combinatorial problems.

In Chapter 4, we presented an efficient local search heuristic for the Blocking Job Shop (BJS) problem with no swap, based on the popular *N4* neighborhood. Efficient algorithms for identifying feasible *N4* neighbors and for computing their makespan were presented. For infeasible *N4* neighbors, we presented an efficient job insertion based algorithm. Critical to the performance of this job insertion procedure was an efficient computational procedure that we developed for obtaining the job insertion polytope. Our algorithms for identifying feasible *N4* neighbors and their makespan, and the polytope based job insertion procedure, can all be easily extended to be applicable for other complex Job Shop variants. With these efficient implementations, we obtained new best results on 28 out of the 40 Lawrence benchmark instances, and matched the previous best on 11 out of the 12 remaining instances. Finally, we showed new structural properties of the Job insertion polytope. Based on those properties, we outlined ways to further improve the local search.

In Chapter 5, we introduced and analyzed a new class of problems called Multi-Robot Scheduling Problem with Blocking and Enabling Constraints (MRSBE). In this problem, multiple robots work in proximity, and are tasked to service an overlapping set of locations without colliding. The objective to minimize is makespan. We analyzed the complexity of MRSBE and its subproblem, namely Scheduling Subproblem (SSp). Given a tour for each robot, SSp is the problem of computing a schedule for those tours. We showed that determining whether the SSp admits a feasible schedule is NP-Complete. We developed an efficient scheduler for the SSp problem, that leverages techniques that we developed akin to constraint propagation. To solve MRSBE for realistically sized instances, we developed a meta-heuristic approach that utilizes two specialized neighborhood search operators inspired from vehicle routing literature. One of the operators is based on exchanging tasks between robots, and the other is based on a TSP heuristic that reorders tasks assigned to a robot. Both these operators rely on the SSp scheduler to convert their outputs into feasible schedules. We compared our meta-heuristic against a strong lower bound (mTSP) as well as an alternative method for generating upper bound solutions. On the real world problems instances that motivated this chapter, we showed that our heuristic outputs solutions within 2% of the optimal.

## 6.1 General Takeaways Retrospectively

We believe that several of the ideas and techniques introduced in this thesis in the context of solving a particular problem are likely to have broader applicability in addressing other problems. In this section, we highlight these potentially more general takeaways. We begin with our work presented in Chapter 3 on the MAPF problem. We introduced ideas such as reasoning through the use of polytopes and localization through templates; and coupling of Lagrangian Relax and Cut (LRC) with Decision Diagrams (DDs). We make a few comments about these ideas.

For the MAPF problem, through the use of projection polytopes, our aim was to gather inferences (e.g. cutting planes) for the lower bound through a localized view of the feasible region. By localized view, we generically mean a projection (or relaxation) of the feasible region over a low dimensional space. We gathered inferences by posing queries and tried to obtain answers by only considering the localized views. For the MAPF problem, this approach allowed us to understand how robots would need to navigate within a local spatio-temporal neighborhood to avoid collisions, and this information as it was shown in Chapter 3 proved useful for computing the lower bound. We believe this approach may be practically useful for solving other combinatorial problems, especially in applications where such local views can be highly informative. A few considerations that make this approach computationally attractive are:

- There are procedures available for representing the feasible region of 0-1 problems using Decision Diagrams (DDs), see Bergman *et al.* [2016]. Obtaining a projection from a complete DD description is straightforward, and thereby we have a generic way to construct a local view.

- There is flexibility in the type of queries for which we can obtain answers (inferences)

efficiently from the local views. This is because each local view is essentially a polyhedra, and so we can borrow tools from Linear Programming for answering.

We used templates to specify local views of the MAPF problem. By spatio-temporally shifting the template, we were able to obtain different local views. We were able to associate different local views to the same templates due to the underlying symmetry in the problem instances we considered, see Section 3.6.1. However such symmetries frequently occur in practical use cases of many combinatorial problems and are worth exploiting. In our case, templates were useful for cut generation because, after having constructed the relaxation to the projection polytope associated to a template, we were able to reuse the facial structure of the relaxation by just spatio-temporally shifting the parameters, see Section 3.6.1. For problems where symmetries exist, and if solution methodologies that can benefit from local views are being used, then the idea of associating local views through templates may be useful as it can save us the computational effort of building different relaxations. While templates only provide a local view of the problem, in Sections 3.7.1 - 3.7.2, we showed how concurrently using multiple templates can help expand the scope of our inference procedures developed for local views to larger regions in computationally tractable ways.

By combining LRC with DDs, we empirically showed that we are able to obtain tight lower bounds for the MAPF problem. As we mentioned earlier, given the fact that there are generic ways to construct DDs for combinatorial constraints, it may be promising to apply LRC with DDs for obtaining tight lower bounds to other combinatorial problems. Since the procedure for generating cuts from DDs is also simple, coupling LRC with DDs is computationally elegant. Also in our work, by combining LRC with Conflict Based Search (CBS), we integrated a polyhedral technique into a search-based method. Such integration schemes can be beneficial for search based methods, as search methods can harness the power of polyhedral techniques.

In Chapter 4, we saw that local search for the BJS problem is expensive, i.e., the complexity of the local search operators scales with problem dimensions. For the case when the number of jobs exceeds the number of machines, we showed that blocking imposes some structure in the schedule (see the discussion on states 1 - 4 in Section 4.10). Using that insight, very broadly in Section 4.10, we attempted to structurally characterize feasible schedules, and presented ideas that exploit structure to improve the complexity of local search schemes. Although exploiting structural insights is a common strategy in optimization literature in general, for the BJS problem and it's variants however, the structure that blocking imposes has so for not received considerable attention. We speculate an analysis similar to that of ours can lead to interesting insights for other BJS variants, and may prove useful for developing efficient local search schemes.

In Chapter 5, we designed our scheduler by hybridizing search on a state space graph with Constraint Programming. For some applications, the concept of a state naturally arises and it makes algorithm development for the application simple and intuitive. However, in many such applications (e.g. robot task planning), the search procedure typically only reasons forward in time when constructing a plan, and so due to the poor guidance, the search may frequently backtrack. Constraint propagation on temporal constraint graphs can reason both forward and backward in time [Cheng and Smith, 1997]. Adapting from such an approach, we designed our scheduler, where at every step in the state space search procedure, the scheduling

decisions made by search are propagated on the temporal constraint graph, and the inferences obtained through propagation are utilized to guide the next search step. It is unclear to us how widely this hybrid approach is currently used in applications where state space search approaches dominate. Hybridizing search with constraint propagation for such applications may be a worthwhile strategy.

## 6.2 Future Research Directions

### 6.2.1 Multi-Agent Path Finding

In Section 3.7, we provided a few directions for extending our work. In this section, we mention some more future directions of research. In recent literature [Lam *et al.*, 2019; Lam and Le Bodic, 2020], the branch-cut-and-price (BCP) technique was shown to outperform CBS. Although the BCP approach is fundamentally different to our approach, it shares the commonality that cutting planes are used to derive strong lower bounds. Since our template based approach provides a generic way to analyze the paths of multiple robots concurrently for generating cuts, an interesting direction of future work is to incorporate our cuts to strengthen the BCP technique.

While the original MAPF problem is somewhat idealized, there is growing interest in extensions of the MAPF problem [Atzmon *et al.*, 2018; Li *et al.*, 2019b]. In Atzmon *et al.* [2018], the authors introduced a new notion of robustness, called k-robustness. It is a notion to generate conflict-free plans that are robust to execution delays up to k time units. In [Li *et al.*, 2019b], the authors consider the MAPF variant, where some agents need to occupy multiple locations on the grid simultaneously. In future work, we can adapt the Lagrangian Relax-and-Cut lower bounding scheme to both these variants of the MAPF problem. We simply have to add more constraints to the relaxation polytope $P(S)$ described in Section 3.3.4, to account for the additional constraints present in these extensions of the MAPF problem.

### 6.2.2 Blocking Job Shop Problem

The Job insertion class of algorithms introduced in Section 4.6 has been amongst the state-of-the-art local search algorithms for several extensions to the Job Shop (JS) problem. Previously, through Sections 4.7.1 - 4.7.7, we introduced an efficient implementation of the Job insertion algorithm based on a compact representation of the **JIP**. An interesting direction of future work is to extend our polytope based approach for other variants of the JS problem.

Despite the efficient updates to the Job insertion class of algorithms proposed in our work, the complexity of obtaining a neighbor is still quite high, i.e., it is at least $\mathcal{O}(\mathbf{M}^2\mathbf{J})$. In most real world problems, $\mathbf{M} << \mathbf{J}$, and so it is beneficial if we can develop neighbor generation schemes that are independent of $\mathbf{J}$. In Section 4.10.1, we proposed one such scheme motivated from the result presented in Theorem 5. Implementing this scheme is an interesting direction of future work.

### 6.2.3 Multi-Robot Routing and Scheduling with Blocking and Enabling Constraints (MRSBE)

We see a couple of directions for future research. Firstly, we have treated MRSBE as a deterministic problem, but in practice there is variability in the service times, and robot failures can occur reasonably often. This suggests the utility of exploring stochastic variants of MRSBE. Second, there are other strategies for avoiding collisions where robots take explicit evasive actions. Such strategies could provide a better basis for coping with blocking constraints. Thirdly, we fixed the trajectory of a robot between a pair of task locations. It would be interesting to extend our work where a robot has multiple trajectory options between a pair of task locations, as it can potentially reduce makespan. Lastly, it was sufficient for just one of the enablers to be serviced earlier for an Enabling Constraint (EC) to be satisfied. In future work we can extend the scope whereby $k$ of the enablers need to be serviced earlier for the EC to be satisfied, where $k$ is a positive integer.

# Bibliography


Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

Toshiro Araki, Yuji Sugiyama, Tadao Kasami, and Jun Okui. Complexity of the deadlock avoidance problem. In *In 2nd IBM Symp. on Mathematical Foundations of Computer Science IBM*, pages 229–257, 1977.

Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Barták, and Neng-Fa Zhou. Robust multi-agent path finding. In *Eleventh Annual Symposium on Combinatorial Search*, 2018.

Barrie M Baker and Janice Sheasby. Accelerating the convergence of subgradient optimisation. *European Journal of Operational Research*, 117(1):136–144, 1999.

Egon Balas and Alkis Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management science*, 44(2):262–275, 1998.

Egon Balas, Neil Simonetti, and Alkis Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11(4):253–262, 2008.

Egon Balas. New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, 86:529–558, 1999.

Jan Kristof Behrens, Ralph Lange, and Masoumeh Mansouri. A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8705–8711. IEEE, 2019.

Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209 – 219, 2006.

Thierry Benoist, François Laburthe, and Benoît Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems. In *CPAIOR*, volume 1, pages 15–26, 2001.

David Bergman, Andre A Cire, and Willem-Jan van Hoeve. Improved constraint propagation via lagrangian decomposition. In *International Conference on Principles and Practice of Constraint Programming*, pages 30–38. Springer, 2015.

David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John Hooker. *Decision Diagrams for Optimization*. Springer, 2016.

Christian Bierwirth and Frank Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, 2010.

Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European journal of operational research*, 93(1):1–33, 1996.

Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Eyal Shimony. Icbs: improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Peter Brucker and Thomas Kampmeyer. Cyclic job shop scheduling problems with blocking. *Annals of Operations Research*, 159(1):161–181, 2008.

Reinhard Bürgy. A neighborhood for complex job shop scheduling problems with regular objectives. *Journal of Scheduling*, 20(4):391–422, 2017.

Edmund K Burke and Yuri Bykov. The late acceptance hill-climbing heuristic. *European Journal of Operational Research*, 258(1):70–78, 2017.

Margarita P Castro, Andre A Cire, and J Christopher Beck. An mdd-based lagrangian approach to the multicommodity pickup-and-delivery tsp. *INFORMS Journal on Computing*, 32(2):263–278, 2020.

Cheng-Chung Cheng and Stephen F Smith. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research*, 70:327–357, 1997.

Vincent A. Cicirello and Stephen F. Smith. Enhancing stochastic search performance by value-biased randomization of heuristics. *Journal of Heuristics*, 11:5–34, 2005.

Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014.

Richard Walter Conway, William L Maxwell, and Louis W Miller. *Theory of scheduling*. Courier Corporation, 2003.

Adel Dabah, Ahcene Bendjoudi, and Abdelhakim AitZai. An efficient tabu search neighborhood based on reconstruction strategy to solve the blocking job shop scheduling problem. *Journal of Industrial & Management Optimization*, 13(4):2015–2031, 2017.

Adel Dabah, Ahcene Bendjoudi, Abdelhakim AitZai, and Nadia Nouali Taboudjemat. Efficient parallel tabu search for the blocking job shop scheduling problem. *Soft Computing*, 2019.

Danial Davarnia and Willem-Jan van Hoeve. Outer approximation for integer nonlinear programs via decision diagrams. *Mathematical Programming*, pages 1–40, 2020.

Mauro Dell'Amico and Marco Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations research*, 41(3):231–252, 1993.

Andrea D'ariano, Dario Pacciarelli, and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643–657, 2007.

Esra Erdem, Doga Gizem Kisa, Umut Oztok, and Peter Schüller. A general formal framework for pathfinding problems with multiple agents. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

Laureano F Escudero, Monique Guignard, and Kavindra Malik. A lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, 50(1):219–237, 1994.

Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, TK Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

Graeme Gange, Daniel Harabor, and Peter J Stuckey. Lazy cbs: Implicit conflict-based search using lazy clause generation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 155–162, 2019.

Michael R Garey and David S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal on Computing*, 6(3):416–426, 1977.

Michael R Garey and David S Johnson. *Computers and intractability*. New York: Freeman, 1979.

Arthur M Geoffrion. Lagrangean relaxation for integer programming. In *Approaches to integer programming*, pages 82–114. Springer, 1974.

Carmen Gervet. New structures of symbolic constraint objects: sets and graphs (extended abstract). In *Third workshop on constraint logic programming*, 1993.

Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

Matthew C Gombolay, Ronald J Wilcox, and Julie A Shah. Fast scheduling of robot teams performing tasks with temporospatial constraints. *IEEE Transactions on Robotics*, 34(1):220–239, 2018.

H. Groeflin and A. Klinkert. A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics*, 157(17):3643–3655, 2009.

Heinz Gröflin and Andreas Klinkert. Feasible insertions in job shop scheduling, short cycles and stable sets. *European Journal of Operational Research*, 177(2):763–785, 2007.

Heinz Gröflin and Andreas Klinkert. A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics*, 157(17):3643–3655, 2009.

Heinz Gröflin, Dinh Nguyen Pham, and Reinhard Bürgy. The flexible blocking job shop with transfer and set-up times. *Journal of combinatorial optimization*, 22(2):121–144, 2011.

M. K. Habib and H. Asama. Efficient method to generate collision free paths for an autonomous mobile robot based on new free space structuring approach. In *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, pages 563–567 vol.2, Nov 1991.

Nicholas G Hall and Chelliah Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525, 1996.

Silvia Heitmann. Job-shop scheduling with limited buffer capacities. 2007.

J. A. John and N. R. Draper. An alternative family of transformations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 29(2):190–197, 1980.

Ariyan M Kabir, Shantanu Thakar, Prahar M Bhatt, Rishi K Malhan, Pradeep Rajendran, Brual C Shah, and Satyandra K Gupta. Incorporating motion planning feasibility considerations during task-agent assignment to perform complex tasks using mobile manipulators. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5663–5670. IEEE, 2020.

Mohand Ou Idir Khemmoudj, Hachemi Bennaceur, and Anass Nagih. Combining arc-consistency and dual lagrangean relaxation for filtering csps. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 258–272. Springer, 2005.

Kap Hwan Kim and Young-Man Park. A crane scheduling method for port container terminals. *European Journal of operational research*, 156(3):752–768, 2004.

Tamás Kis and Alain Hertz. A lower bound for the job insertion problem. *Discrete Applied Mathematics*, 128(2-3):395–419, 2003.

Andreas Klinkert. *Optimization in design and control of automated high-density warehouses*. PhD thesis, PhD thesis, University of Fribourg, Switzerland, 2001.

Edward Lam and Pierre Le Bodic. New valid inequalities in branch-and-cut-and-price for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 184–192, 2020.

Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter J Stuckey. Branch-and-cut-and-price for multi-agent pathfinding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), International Joint Conferences on Artificial Intelligence Organization*, pages 1289–1296, 2019.

Julia Lange and Frank Werner. Approaches to modeling train scheduling problems as job-shop problems with blocking constraints. *Journal of Scheduling*, 21(2):191–207, 2018.

Julia Lange and Frank Werner. A permutation-based heuristic method for the blocking job shop scheduling problem. *IFAC-PapersOnLine*, 52(13):1403–1408, 2019.

Misha Lavrov. An upper bound on the number of chordless cycles in an undirected graph. Mathematics Stack Exchange, 2018. URL:https://math.stackexchange.com/q/2811761 (version: 2018-06-07).

S Lawrence. Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

Jiaoyang Li, Eli Boyarski, Ariel Felner, Hang Ma, and Sven Koenig. Improved heuristics for multi-agent path finding with conflict-based search. In *International Joint Conference on Artificial Intelligence*, pages 442–449, 2019.

Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019.

Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J Stuckey, Hang Ma, and Sven Koenig. New techniques for pairwise symmetry breaking in multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 193–201, 2020.

Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. New techniques for pairwise symmetry breaking in multi-agent path finding. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

Shi Qiang Liu and Erhan Kozan. Scheduling trains as a blocking parallel-machine job shop scheduling problem. *Computers & Operations Research*, 36(10):2840–2852, 2009.

Abilio Lucena. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140(1):375–410, 2005.

Masoumeh Mansouri, Henrik Andreasson, and Federico Pecora. Hybrid reasoning for multi-robot drill planning in openpit mines. 2016.

Masoumeh Mansouri, Fabien Lagriffoul, and Federico Pecora. Multi vehicle routing with nonholonomic constraints and dense dynamic obstacles. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3522–3529. IEEE, 2017.

Masoumeh Mansouri, Federico Pecora, and Peter Schüller. Combining task and motion planning: Challenges and guidelines. *Frontiers in Robotics and AI*, 8:133, 2021.

Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

Yazid Mati and Xiaolan Xie. Multiresource shop scheduling with resource flexibility and blocking. *IEEE transactions on automation science and engineering*, 8(1):175–189, 2011.

Yazid Mati, Nidhal Rezg, and Xiaolan Xie. Geometric approach and taboo search for scheduling flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 17(6):805–818, 2001.

Yazid Mati, Nidhal Rezg, and Xiaolan Xie. A taboo search approach for deadlock-free scheduling of automated manufacturing systems. *Journal of Intelligent Manufacturing*, 12(5-6):535–552, 2001.

Yazid Mati, Chams Lahlou, and Stephane Dauzere-Peres. Modelling and solving a practical flexible job-shop scheduling problem with blocking constraints. *International Journal of Production Research*, 49(8):2169–2182, 2011.

Carlo Meloni, Dario Pacciarelli, and Marco Pranzo. A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131(1-4):215–235, 2004.

Lingyun Meng and Xuesong Zhou. Simultaneous train rerouting and rescheduling on an n-track network: A model reformulation with network-based cumulative flow variables. *Transportation Research Part B: Methodological*, 67:208–234, 2014.

Luigi Moccia, Jean-François Cordeau, Manlio Gaudioso, and Gilbert Laporte. A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. *Naval Research Logistics (NRL)*, 53(1):45–59, 2006.

Jayanth Krishna Mogali, Willem-Jan van Hoeve, and Stephen F Smith. Template matching and decision diagrams for multi-agent path finding. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 347–363. Springer, 2020.

Jayanth Krishna Mogali, Laura Barbulescu, and Stephen F Smith. Efficient primal heuristic updates for the blocking job shop problem. *European Journal of Operational Research*, 2021.

Jayanth Krishna Mogali, Joris Kinable, Stephen F Smith, and Zachary B Rubinstein. Scheduling for multi-robot routing with blocking and enabling constraints. *Journal of Scheduling*, pages 1–28, 2021.

Rolf H Möhring, Martin Skutella, and Frederik Stork. Scheduling with and/or precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004.

Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management science*, 42(6):797–813, 1996.

Eugeniusz Nowicki and Czesław Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.

Angelo Oddi, Riccardo Rasconi, Amedeo Cesta, and Stephen F Smith. Iterative improvement algorithms for the blocking job shop. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

Florian Pommerening, Gabriele Röger, Malte Helmert, Hadrien Cambazard, Louis-Martin Rousseau, and Domenico Salvagnin. Lagrangian decomposition for optimal cost partitioning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 338–347, 2019.

Jens Poppenborg, Sigrid Knust, and Joachim Hertzberg. Online scheduling of flexible job-shops with blocking and transportation. *European Journal of Industrial Engineering*, 6(4):497–518, 2012.

Marco Pranzo and Dario Pacciarelli. An iterated greedy metaheuristic for the blocking job shop scheduling problem. *Journal of Heuristics*, 22(4):587–611, 2016.

Zachary B Rubinstein, Stephen F Smith, and Laura Barbulescu. Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. In *AAAI*, 2012.

Marcella Sama, Andrea D'Ariano, Paolo D'Ariano, and Dario Pacciarelli. Scheduling models for optimal aircraft traffic control at busy airports: tardiness, priorities, equity and violations considerations. *Omega*, 67:81–98, 2017.

Marcello Sammarra, Jean-François Cordeau, Gilbert Laporte, and M Flavia Monaco. A tabu search heuristic for the quay crane scheduling problem. *Journal of Scheduling*, 10(4-5):327–336, 2007.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. *SoCS*, 1:39–40, 2012.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

David Silver. Cooperative pathfinding. *Aiide*, 1:117–122, 2005.

Neil Simonetti and Egon Balas. Implementation of a linear time algorithm for certain generalized traveling salesman problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 316–329. Springer, 1996.

Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019.

Roni Stern. *Multi-Agent Path Finding – An Overview*, pages 96–115. Springer International Publishing, Cham, 2019.

Christian Strotmann. Railway scheduling problems and their decomposition. *Ph.D. Dissertation*, 2008.

Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient sat approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pages 810–818. IOS Press, 2016.

Eric Taillard. Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285, 1993.

Christian Tjandraatmadja and Willem-Jan van Hoeve. Target cuts from relaxed decision diagrams. *INFORMS Journal on Computing*, 31(2):285–301, 2019.

Johanna Törnquist and Jan A Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B: Methodological*, 41(3):342–362, 2007.

Alex Van Breedam. *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle-related, Customer-related, and Time-related Constraints*. RUCA, 1994.

Thomas Van den Bossche, Hatice Çalik, Evert-Jan Jacobs, Tulio Toffolo, and Greet Vanden Berghe. Truck scheduling in tank terminals. *EURO Journal on Transportation and Logistics*, 9(1):100001, 2020.

Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 3260–3267. IEEE, 2011.

Johan Wessén, Mats Carlsson, and Christian Schulte. Scheduling of dual-arm multi-tool assembly robots and workspace layout optimization. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 511–520. Springer, 2020.

Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

Jingjin Yu and Steven M LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 3612–3617. IEEE, 2013.

Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

# Appendices

# Appendix

## A Multi-agent Path Finding

### A.1 Additional Proofs

**Proposition** 1: LR-WDG($sn$) is a valid lower bound to the problem shown in Eqn (3.42).

*Proof.* Observe that since $\hat{\lambda} \geq 0$, and $\hat{E}x \leq \hat{f}$ are valid inequalities for $\mathbf{P}$, we have:

$$\begin{aligned} & c^{\mathsf{T}}x + \hat{\lambda}^{\mathsf{T}}(\hat{E}x - \hat{f}) \leq c^{\mathsf{T}}x, \quad \forall x \in \mathbf{P} \\ \Rightarrow & -\hat{\lambda}^{\mathsf{T}}\hat{f} + \min_{x \in P_{sn}} c^{\mathsf{T}}x + \hat{\lambda}^{\mathsf{T}}\hat{E}x \leq \min_{x \in P_{sn}} c^{\mathsf{T}}x, \text{ where } P_{sn} = \{x \in \mathbf{P} | x(a) = 0, \forall a \in \bar{\mathcal{A}}_{sn}\} \end{aligned} \tag{A.1}$$

Notice that the RHS of Eqn (A.1) is identical to Eqn (3.42). Given an optimal solution to the minimization problem shown in the LHS of Eqn (A.1), let us denote the start-end path cost for robot $r_k$ in the optimal solution by $y_k$, and so we have the relation shown in Eqn (A.2). In reality solving the minimization problem in the LHS of Eqn (A.1) is as hard as solving the original MAPF problem, so following the approach in [Li *et al.*, 2019a] we will instead characterize $y_k$, and use it to derive a lower bound for the LHS of Eqn (A.1), and by extension we obtain a lower bound to Eqn (3.42). More specifically, we will prove that the $y.$ values satisfy Eqns (3.45) - (3.46). Observe that the objective in the LR-WDG bound and the LHS of Eqn (A.1) are identical due to Eqn (A.2), and so the LR-WDG bound is trying to find an assignment of values to $y.$ that satisfies Eqns (3.45) - (3.46), and minimizes the objective. So assuming correctness of Eqns (3.45) - (3.46), the LR-WDG bound will produce a lower bound to the LHS of Eqn (A.1).

$$\min_{x \in P_{sn}} c^{\mathsf{T}}x + \hat{\lambda}^{\mathsf{T}}\hat{E}x = \sum_{k \in [\mathbf{N}]} y_k \tag{A.2}$$

For $i, j \in [\mathbf{N}]$, and $i < j$, let us denote the convex hull of the feasible region of $l^{\hat{\lambda}}_{sn}(i, j)$ by $Q_{sn}$, and so $Q_{sn} = \mathbf{conv}(x \in \{0, 1\}^{|A|} | x$ satisfies Eqns (3.35) - (3.38)). Observe that any assignment of values to the variables $x(A_i \cup A_j)$ in $P_{sn}$ is also feasible to $Q_{sn}$. This is because, unlike in $P_{sn}$, all robots other than robots $r_i, r_j$ are disregarded in $Q_{sn}$, so an assignment of values to $x(A_i \cup A_j)$ in $Q_{sn}$ is unaffected by the assignment to $x\left(A \backslash \{A_i \cup A_j\}\right)$ through conflict related constraints, since $x\left(A \backslash \{A_i \cup A_j\}\right) = 0$ in $Q_{sn}$. So, $\text{Proj}_{x(A_i \cup A_j)}(P_{sn}) \subseteq$

$\text{Proj}_{x(A_i \cup A_j)}(Q_{sn})$, and we can thereby infer:

$$l^{\hat{\lambda}}_{sn}(i,j) = \min_{x \in Q_{sn}} c^{\mathsf{T}}x + \hat{\lambda}^{\mathsf{T}}\hat{E}x \leq \min_{\substack{x(A_i \cup A_j) \in \text{Proj}_{x(A_i \cup A_j)}(P_{sn}) \\ x(A \setminus \{A_i \cup A_j\}) = 0}} c^{\mathsf{T}}x + \hat{\lambda}^{\mathsf{T}}\hat{E}x \tag{A.3}$$

Finally, note that since $y_i + y_j$ is the sum of path costs for robots $r_i, r_j$, where the paths are mutually non-conflicting, then, those paths specified in terms of the $x(A_i \cup A_j)$ variables must belong to $\text{Proj}_{x(A_i \cup A_j)}(P_{sn})$. So a simple lower bound on $y_i + y_j$ is given by Eqn (A.4).

$$\min_{\substack{x(A_i \cup A_j) \in \text{Proj}_{x(A_i \cup A_j)}(P_{sn}) \\ x(A \setminus \{A_i \cup A_j\}) = 0}} c^{\mathsf{T}}x + \hat{\lambda}^{\mathsf{T}}\hat{E}x \leq y_i + y_j \tag{A.4}$$

Equations (A.3), (A.4) imply that $l^{\hat{\lambda}}_{sn}(i,j) \leq y_i + y_j$. Also, it is immediate from the definition of $y_k$ and $LB_{\hat{\lambda}}(k, sn)$, that Eqn (3.46) must hold. □

## A.2 Templates for Experiments

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G |
| H | I | J | K | L | M | N |
| O | P | Q | R | S | T | U |
| V | W | X | Y | Z | AA | AB |
| AC | AD | AE | AF | AG | AH | AI |
| AJ | AK | AL | AM | AN | AO | AP |
| AQ | AR | AS | AT | AU | AV | AW |

**(a)** Blocks for horizontal movement

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G |
| H | I | J | K | L | M | N |
| O | P | Q | R | S | T | U |
| V | W | X | Y | Z | AA | AB |
| AC | AD | AE | AF | AG | AH | AI |
| AJ | AK | AL | AM | AN | AO | AP |
| AQ | AR | AS | AT | AU | AV | AW |

**(b)** Blocks for vertical movement

**Figure A.1**

We will begin by describing how the 3 robot templates used for our experiments are generated. Consider the following blocks of locations in the grids shown in Figure A.1:

1. LFT1 = P, Q, W, X, AD, AE
2. LFT2 = Q, R, X, Y, AE, AF
3. LFT3 = R, S, Y, Z, AF, AG
4. LFT4 = S, T, Z, AA, AG, AH.

1. TP1 = J, K, L, Q, R, S
2. TP2 = Q, R, S, X, Y, Z
3. TP3 = X, Y, Z, AE, AF, AG
4. TP4 = AE, AF, AG, AL, AM, AN.

Consider the ordered set of sets shown below. We will use them for specifying templates.

- LEFT = (LFT1, LFT2, LFT2, LFT3, LFT4)

- RIGHT = (LFT4, LFT3, LFT3, LFT2, LFT1)
- TOP = (TP1, TP2, TP2, TP3, TP4)
- BOTTOM = (TP4, TP3, TP3, TP2, TP1)

Consider a 3 robot template denoted by $S$, w.l.o.g., we will assume that $\mathcal{R}(S) = [3]$ (refer Section 3.1.1 for notation of $[\cdot]$), and $T(S) = [5]$. All the 3-robot templates used in our experiments differ only in $L(S)$. To generate the 64 different 3-robot templates in our experiments, we assigned each $i \in \mathcal{R}(S)$ to one among {LEFT, RIGHT, TOP, BOTTOM} independently. So if all $i \in \mathcal{R}(S)$ were assigned to LEFT, then for that template $S$, $L(S)$ is specified as: $L_i^1(S) =$ LFT1, $L_i^2(S) =$ LFT2, $L_i^3(S) =$ LFT2, $L_i^4(S) =$ LFT3, $L_i^5(S) =$ LFT4. Clearly, since we have 4 choices for each $i \in \mathcal{R}(S)$, we can create 64 templates with this approach. The spatial center for all 64 different 3-robot templates is the location Y, and temporal center is 3, we will recall this information later in Section A.3.

The 2 robot template in our experiments is parameterized by: $\mathcal{R}(S) = [2]$, $T(S) = [7]$. Further, $\forall i \in \mathcal{R}(S)$ and $\forall t \in T(S)$, we assign $L_i^t(S)$ to the set of locations falling within the $5 \times 5$ square centered at Y in Figure A.1.

## A.3 Choosing a 3-robot template for a conflict

For a given conflict, we first compute a score for each template, and then select the template with the highest score for generating objective cuts for that conflict. The idea behind our template selection scheme is to prefer the template that overlaps with the paths of the robots the most. We will describe how the score is computed using an example. Assume that the paths for the robots are as follows:

1. Robot $r_1$ path: $\mathrm{O}_1^{20}$, $\mathrm{P}_1^{21}$, $\mathrm{Q}_1^{22}$, $\mathrm{R}_1^{23}$, $\mathrm{S}_1^{24}$, $\mathrm{L}_1^{25}$, $\mathrm{E}_1^{26}$.
2. Robot $r_2$ path: $\mathrm{M}_2^{20}$, $\mathrm{L}_2^{21}$, $\mathrm{K}_2^{22}$, $\mathrm{R}_2^{23}$, $\mathrm{Y}_2^{24}$, $\mathrm{Z}_2^{25}$, $\mathrm{AA}_2^{26}$.
3. Robot $r_3$ path: $\mathrm{V}_3^{20}$, $\mathrm{W}_3^{21}$, $\mathrm{X}_3^{22}$, $\mathrm{Q}_3^{23}$, $\mathrm{P}_3^{24}$, $\mathrm{I}_3^{25}$, $\mathrm{J}_3^{26}$.

Clearly, robots $r_1$ and $r_2$ conflict at R at time 23. Assume, we are interested in computing the score for the 3-robot template $S$ that was generated by assigning LEFT, TOP, and BOTTOM. As we previously saw in the example shown in Section 3.6.1, we will first spatio-temporally shift the template generated in the previous section, so that the template's spatial and temporal center coincide with the conflict. With this spatio-temporal shift operation, $L(S)$ needs to be shifted, so for e.g., $L_1^{21}(S)$ becomes $\{\mathrm{I}_1^{21}, \mathrm{J}_1^{21}, \mathrm{P}_1^{21}, \mathrm{Q}_1^{21}, \mathrm{W}_1^{21}, \mathrm{X}_1^{21}\}$. To compute the score for the template, we count the number of arcs in the robot paths that are present in $S$. So, for example, consider the first arc $(\mathrm{O}_1^{20}, \mathrm{P}_1^{21})$ in robot's 1 path. Clearly, since $\mathrm{P}_1^{21} \in L_1^{21}(S)$, the arc $(\mathrm{O}_1^{20}, \mathrm{P}_1^{21}) \in S$. Consider the arc $(\mathrm{V}_3^{20}, \mathrm{W}_3^{21})$, since $\mathrm{W}_3^{21} \notin L_3^{21}(S)$, the arc $(\mathrm{V}_3^{20}, \mathrm{W}_3^{21}) \notin S$. Repeating this exercise for all other arcs in the robot paths, the score for the template $S$ is 15.

## A.4 Additional Figures

To interpret the plots in this section, refer to the discussions on Figure 3.8 in Section 3.6.2.

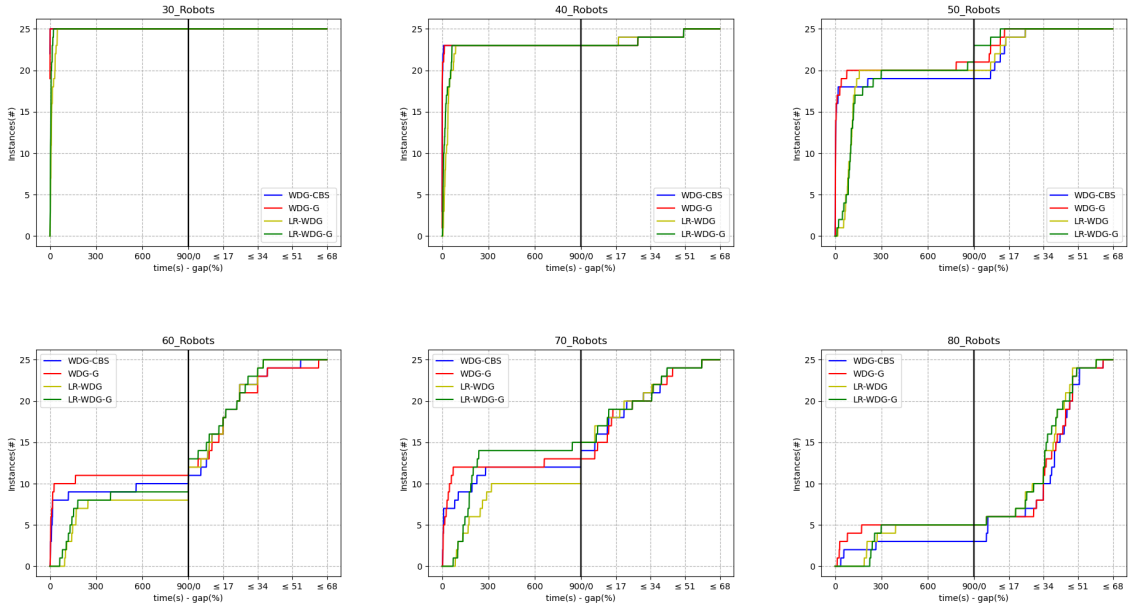



**Figure A.3:** Runtime plots and optimality gap for 10% obstacle instances.

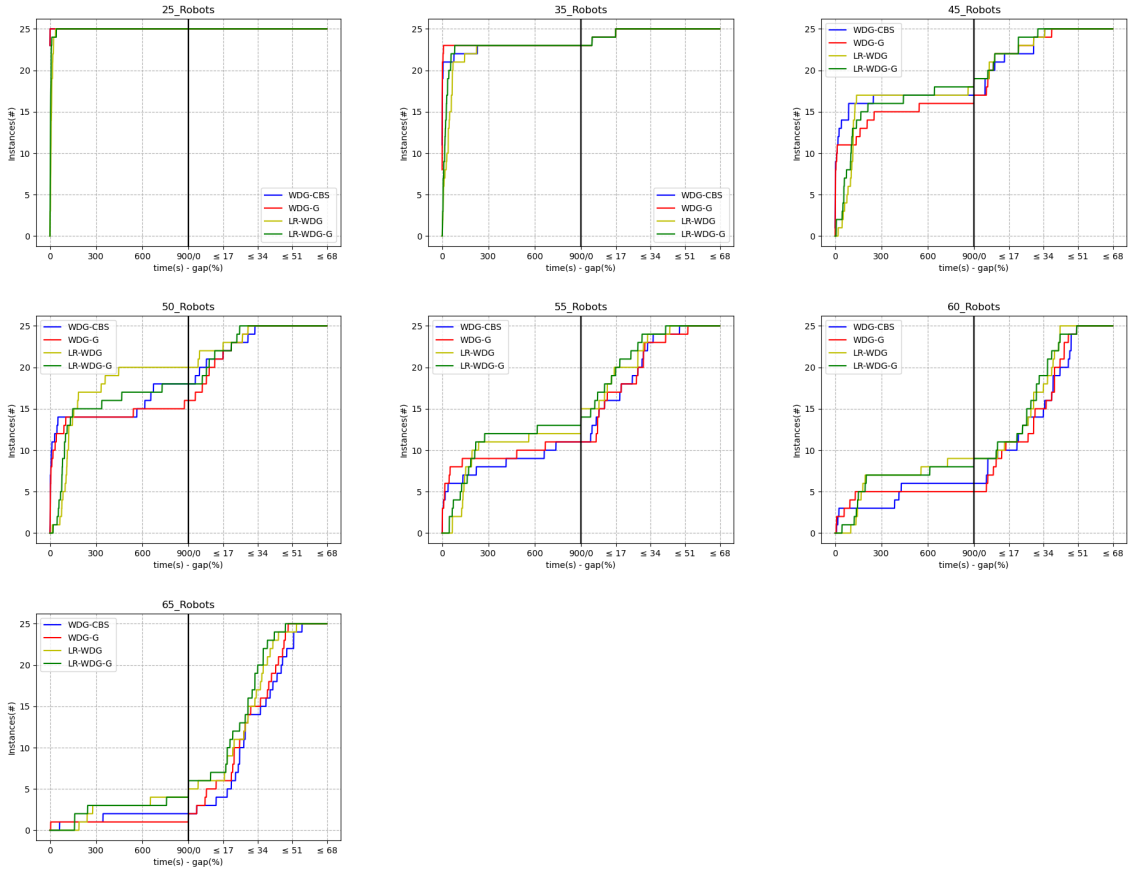



**Figure A.4:** Runtime plots and gap for 15% obstacle instances.

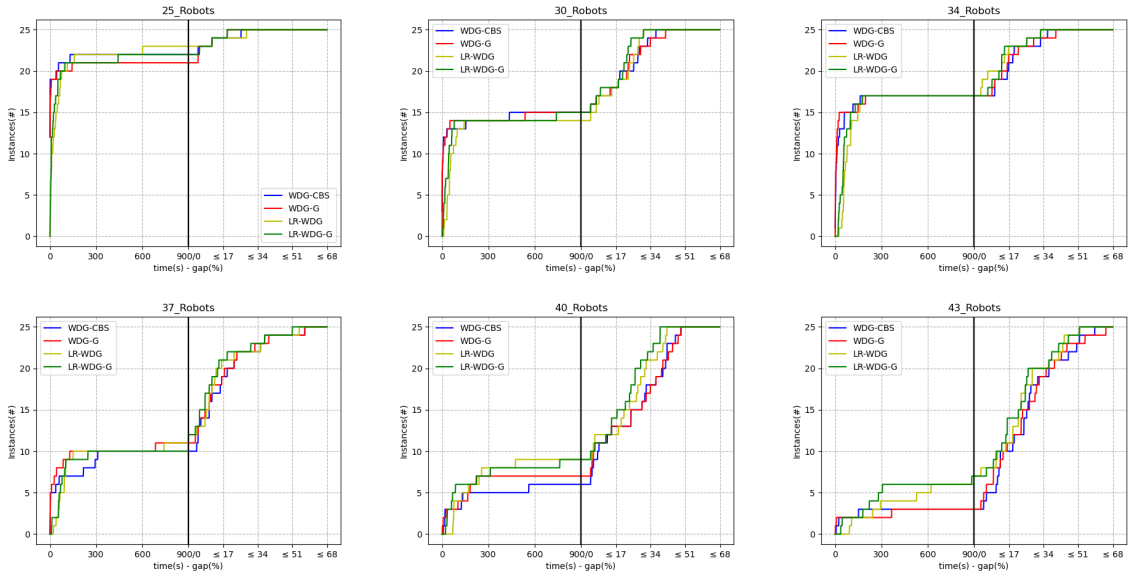



**Figure A.5:** Runtime plots and gap for 25% obstacle instances.

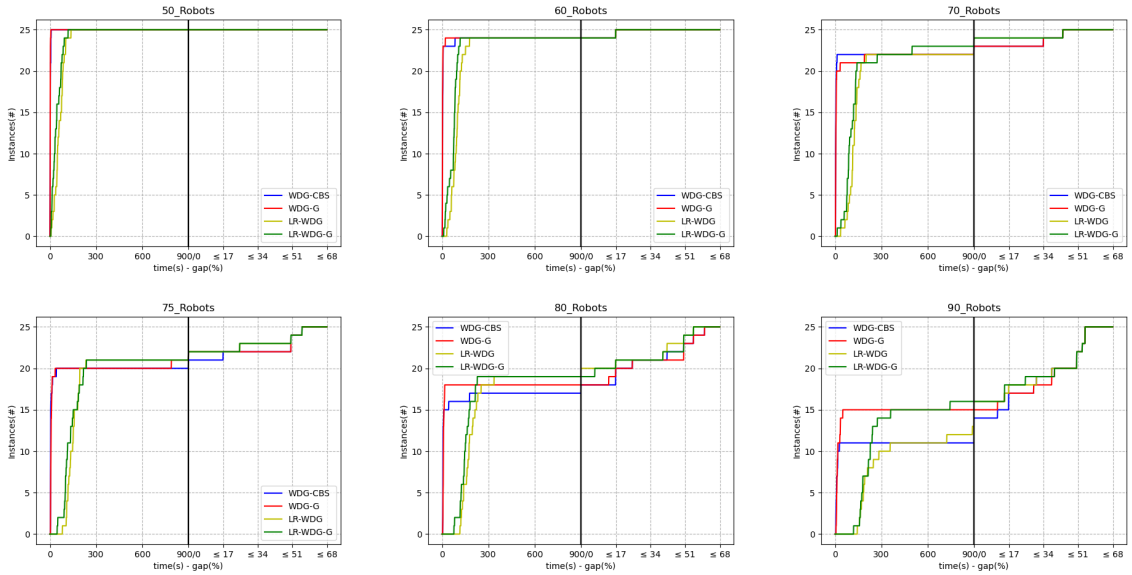



**Figure A.6:** Runtime plots and gap for *Empty* obstacle instances.

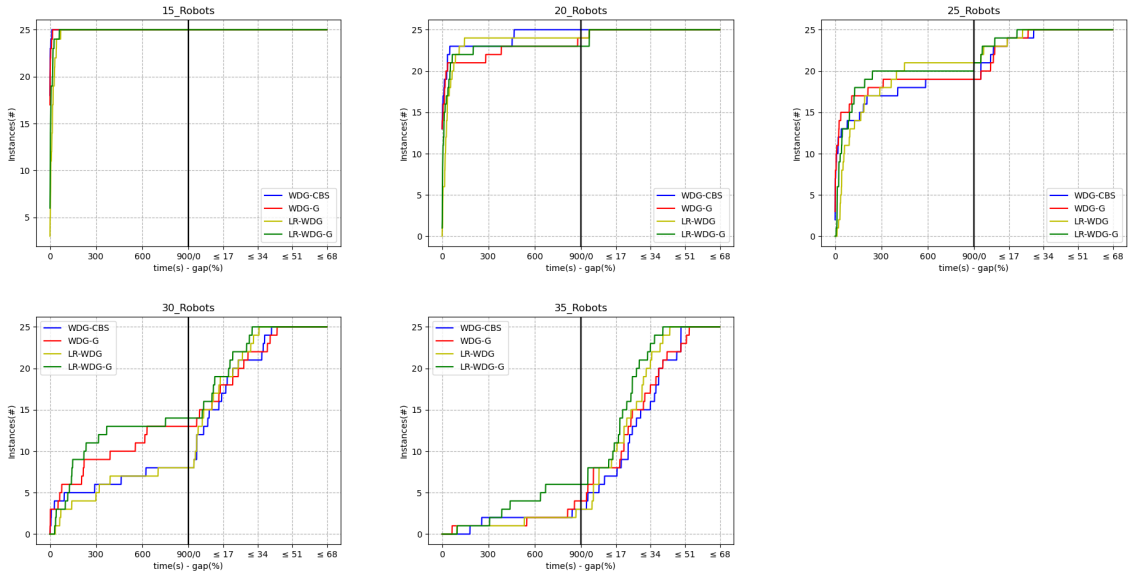



**Figure A.7:** Runtime plots and gap for *Room* instances.

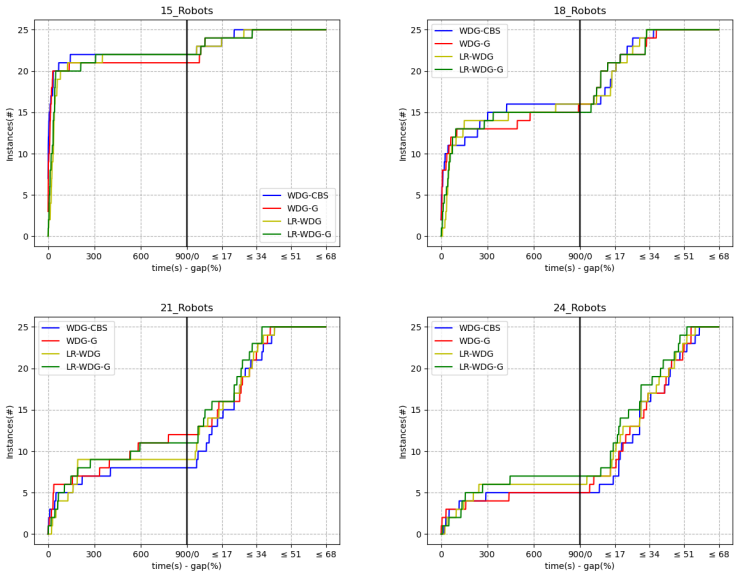



**Figure A.8:** Runtime plots and gap for *Maze* instances.

## A.5 Additional Tables

To interpret the tables in this section, refer to the caption of Table 3.1 in Section 3.6.2.

| Alg. Comp \ Robots | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|
| LR-WDG | 100, 10.9 | 100, 10.9 | 94.1, 31.2 | 87.5, 33.1 | 66.6, 53.7 | 66.6, 42.5 |
| LR-WDG-G | 100, 13.7 | 100, 7.4 | 73.3, 23.2 | 100, 44.8 | 90, 25.5 | 80, 19 |

**Table A.1:** Comparison of # of search nodes expanded in the conflict tree for 10% obstacle instances.

| Alg. Comp \ Robots | 25 | 35 | 45 | 50 | 55 | 60 | 65 |
|---|---|---|---|---|---|---|---|
| LR-WDG | 100, 9.6 | 90.9, 18.6 | 94.1, 35.9 | 94.1, 26.8 | 100, 30.5 | 100, 26.1 | 100, 35.4 |
| LR-WDG-G | 100, 13.7 | 95.4, 17.1 | 93.3 34.4 | 85.7, 20.6 | 87.5, 32 | 100, 33.7 | 100, 72.7 |

**Table A.2:** Comparison of # of search nodes expanded in the conflict tree for 15% obstacle instances.

| Alg. Comp \ Robots | 25 | 30 | 34 | 37 | 40 | 43 |
|---|---|---|---|---|---|---|
| LR-WDG | 90.9, 17.2 | 100, 18.7 | 94.1, 31.9 | 90, 8.9 | 100, 12.9 | 100, 14.7 |
| LR-WDG-G | 100, 18.4 | 92.8, 16.6 | 100, 25.8 | 100, 28 | 100, 13.5 | 100, 8.6 |

**Table A.3:** Comparison of # of search nodes expanded in the conflict tree for 25% obstacle instances.

| Alg. Comp \ Robots | 50 | 60 | 70 | 75 | 80 | 90 |
|---|---|---|---|---|---|---|
| LR-WDG | 100, 3.5 | 100, 13.7 | 80, 22.9 | 42.8, 12.8 | 57.1, 80.9 | 60, 92.8 |
| LR-WDG-G | 100, 4.7 | 100, 2.9 | 100, 5.3 | 80, 12 | 75, 21.6 | 71.4, 53.6 |

**Table A.4:** Comparison of # of search nodes expanded in the conflict tree for *Empty* instances.

| Alg. Comp \ Robots | 15 | 20 | 25 | 30 | 35 |
|---|---|---|---|---|---|
| LR-WDG | 87.5, 28.4 | 95.4, 23.7 | 89.4, 40.4 | 71.4, 37.5 | 66.6, 35.7 |
| LR-WDG-G | 100, 30.9 | 90.4, 18.9 | 94.7, 23.6 | 100, 36.6 | 100, 26.9 |

**Table A.5:** Comparison of # of search nodes expanded in the conflict tree for *Room* instances.

| Alg. Comp \ Robots | 15 | 18 | 21 | 24 |
|---|---|---|---|---|
| LR-WDG | 72.7, 22 | 93.7, 42 | 75, 36.2 | 80, 33.4 |
| LR-WDG-G | 95, 29.2 | 92.8, 25.5 | 81.8, 30.1 | 60, 7.5 |

**Table A.6:** Comparison of # of search nodes expanded in the conflict tree for *Maze* instances.

# B Blocking Job Shop Problem

## Makespan computation of *N4* neighbors

We describe an efficient procedure to compute the makespan of all feasible *N4* neighbors. Like in the case for determining feasibility of the output of the *N4* moves, we will work with the minimal representation of a feasible selection to compute the makespan of the feasible *N4* neighbors. When applying any *N4* move to the selection $S$, three arcs are removed from $S$, and three new arcs are added to obtain a new selection $S'$. The efficiency in computation of the makespan of $S'$ relies on a careful analysis of changes in the selection. Let $T_S$ denote some valid topological ordering for selection $S$, and further assume that $L_S(\theta, o), L_S(o, \Lambda)$ are known for all nodes $o \in \tilde{O}$ (these values can be computed in linear time, i.e. $\mathcal{O}(\mathbf{MJ})$). The procedure for computing the makespan of a feasible *N4* neighbor uses $T_S, L_S(\theta, o), L_S(o, \Lambda)$, and a few other values computed for each critical block separately.

## B.1 Makespan computation for forward move

Assume $S$ is a consistent selection, and let $cB = (u_1, \ldots, u_n, l)$ be a critical block in some critical path of $G(F \cup S)$. For $u \in \{u_1, \ldots, u_n\}$, we present below an algorithm to compute the makespan for the neighbor generated by the simple forward move which makes operation $u$ the machine successor of operation $l$ (assume that $S'$ is consistent). Portions of the graphs $G(F \cup S), G(F \cup S')$ that are affected by the move are shown in Figures 4.3a, 4.3b.

We begin by providing a $\theta - \Lambda$ cut for $G(F \cup S')$, which will be crucial for our makespan computations. The cut will make use of the topological ordering $T_S$ for selection $S$.

**Proposition 8.** *$E_{S'}$ defined in Eqn (B.5) (see Table B.7) is a valid $\theta - \Lambda$ cut for $G(F \cup S')$.*

*Proof.* Consider the $\theta - \Lambda$ cut $E_S$ for $G(F \cup S)$ defined in Eqn (B.6). Notice $E_S$ separates nodes in $G(F \cup S)$ in two sets: the set of all nodes that are ordered before $u$ in $T_S$, and the set containing $u$ and all the nodes ordered after $u$ in $T_S$. Recall that no arcs of the form $(x, y)$, with $T_S^{-1}(x) > T_S^{-1}(u)$ and $T_S^{-1}(y) < T_S^{-1}(u)$ were added to obtain $S'$ from $S$. Hence, the corresponding source-sink cut for selection $S'$ can be obtained by adding to $E_S$ the new arc in $S'$, $(\gamma(\beta_S(u)), \delta_S(u))$, and removing the arc $(\gamma(\beta_S(u)), u)$, since this arc is not in $S'$. This can be rewritten exactly as shown in Eqn (B.5). □

Note that the cardinality of $E_{S'}$ is at most $|\mathbf{M}| + |\mathbf{J}|$, since there can be at most one alternative arc corresponding to each machine, and one fixed arc (i.e. arcs in $F$) corresponding to each job in the cut.

The longest $\theta - \Lambda$ path cost (makespan) in $G(F \cup S')$ i.e. $L_{S'}(\theta, \Lambda)$, can be computed by separately considering the following 2 cases:

1. Cost of the longest path from $\theta$ to $\Lambda$ passing through $\gamma(l)$ in $G(F \cup S')$, denoted by $L_{S'}^{\gamma(l)}(\theta, \Lambda)$.

2. Cost of the longest path that does not pass through $\gamma(l)$ in $G(F \cup S')$, denoted by $L_{S'}^{-\gamma(l)}(\theta, \Lambda)$.

**Table B.7:** Makespan computation for feasible neighbors produced by *N4* forward move

$$E_{S'} = \{(v,w)|(v,w) \in F \cup S, T_S^{-1}(v) < T_S^{-1}(u), T_S^{-1}(w) > T_S^{-1}(u)\} \cup \{(\gamma(\beta_S(u)), \delta_S(u)), (\alpha(u), u)\} \tag{B.5}$$

$$E_S = \{(v,w)|(v,w) \in F \cup S,\ T_S^{-1}(v) < T_S^{-1}(u), T_S^{-1}(w) >= T_S^{-1}(u)\} \tag{B.6}$$

$$\textbf{Makespan} : L_{S'}(\theta, \Lambda) = \max\left(L_{S'}^{\gamma(l)}(\theta, \Lambda), L_{S'}^{-\gamma(l)}(\theta, \Lambda)\right) \tag{B.7}$$

$$L_{S'}(\theta, \gamma(l)) = \max_{(v,w)\in E_{S'}} C^{\gamma(l)}(v,w) = \begin{cases} -\infty, \text{ if } T_S^{-1}(w) > T_S^{-1}(\gamma(l)) \text{ or } w = u \text{ or } w = \gamma(u) \\ L_S(\theta, v) + l(v,w) + L_{S\setminus(\gamma(u),\delta_S(u))}(w, \gamma(l)), \text{ if } T_S^{-1}(u) < T_S^{-1}(w) < T_S^{-1}(\gamma(u)) \\ L_S(\theta, v) + l(v,w) + L_S(w, \gamma(l)), \text{ if } T_S^{-1}(\gamma(u)) < T_S^{-1}(w) \leq T_S^{-1}(\gamma(l)) \end{cases} \tag{B.8}$$

$$L_{S'}(\gamma(l), \Lambda) = \max(l(\gamma(l), \gamma(\gamma(l))) + L_{S'}(\gamma(\gamma(l)), \Lambda), l(\gamma(l), u) + L_{S'}(u, \Lambda)) \tag{B.9}$$

$$= \max(l(\gamma(l), \gamma(\gamma(l))) + L_S(\gamma(\gamma(l)), \Lambda), l(\gamma(l), u) + L_{S'}(u, \Lambda)) \tag{B.10}$$

$$L_{S'}(u, \Lambda) = \max(l(u, \gamma(u)) + L_S(\gamma(u), \Lambda), l(u, \delta_S(\alpha(u))) + L_{S'}(\delta_S(\alpha(u)), \Lambda)) \tag{B.11}$$

$$L_{S'}(\delta_S(\alpha(u)), \Lambda) = \max(L_{S'}^{-\gamma(u)}(\delta_S(\alpha(u)), \Lambda), L_S(\delta_S(\alpha(u)), \gamma(u)) + L_S(\gamma(u), \Lambda)) \tag{B.12}$$

$$L_{S'}^{-\gamma(u)}(\delta_S(\alpha(u)), \Lambda) = \begin{cases} L_S(\delta_S(\alpha(u)), \Lambda) & \text{, if } T_S^{-1}(\delta_S(\alpha(u))) > T_S^{-1}(\gamma(u)) \\ L_S^{-\gamma(u)}(\delta_S(\alpha(u)), \Lambda), \text{ otherwise.} \end{cases} \tag{B.13}$$

$$L_{S'}^{-\gamma(l)}(\theta, \Lambda) = \max_{(v,w)\in E_{S'}} C^{-\gamma(l)}(v,w) = \begin{cases} L_S(\theta, \alpha(u)) + l(\alpha(u), u) + L_{S'}(u, \Lambda), \text{ if } w = u. \\ L_S(\theta, v) + l(v,w) + L_{S'}^{-\gamma(l)}(w, \Lambda), \text{ otherwise.} \end{cases} \tag{B.14}$$

$$L_{S'}^{-\gamma(l)}(w, \Lambda) = \begin{cases} L_S^{-\gamma(l)}(w, \Lambda) & \text{, if } T_S^{-1}(w) > T_S^{-1}(\gamma(u)) \\ \max\left(L_S(w, \gamma(u)) + L_S(\gamma(u), \Lambda), L_{S\setminus(\gamma(u),\delta_S(u))}^{-\gamma(l)}(w, \Lambda)\right) & \text{, if } T_S^{-1}(u) < T_S^{-1}(w) \leq T_S^{-1}(\gamma(u)) \end{cases} \tag{B.15}$$

Once we compute $L_{S'}^{\gamma(l)}(\theta, \Lambda), L_{S'}^{-\gamma(l)}(\theta, \Lambda)$, then the makespan of $S'$ i.e. $L_{S'}(\theta, \Lambda)$ is given by:

$$L_{S'}(\theta, \Lambda) = \max\left(L_{S'}^{\gamma(l)}(\theta, \Lambda), L_{S'}^{-\gamma(l)}(\theta, \Lambda)\right)$$

Further, since $E_{S'}$ is a $\theta - \Lambda$ cut, every $\theta - \Lambda$ path in $G(F \cup S')$ must contain exactly one of the arcs in $E_{S'}$. The quantity $L_{S'}^{\gamma(l)}(\theta, \Lambda)$ can be obtained by computing the longest $\theta - \Lambda$ path cost through each arc in $E_{S'}$ that also passes through $\gamma(l)$. If no $\theta - \Lambda$ path passing through $\gamma(l)$ and a cut arc exists, then the corresponding path cost through the cut arc will be $-\infty$. We then compute the maximum across all path costs computed through each cut arc to obtain $L_{S'}^{\gamma(l)}(\theta, \Lambda)$. We perform a similar procedure to compute $L_{S'}^{-\gamma(l)}(\theta, \Lambda)$.

**Computation of $L_{S'}^{\gamma(l)}(\theta, \Lambda)$**

We can write $L_{S'}^{\gamma(l)}(\theta, \Lambda) = L_{S'}(\theta, \gamma(l)) + L_{S'}(\gamma(l), \Lambda)$.

To compute $L_{S'}(\theta, \gamma(l))$, first observe that $\theta$ and $u$ both do not belong to the same partition of nodes created by cut $E_{S'}$ due to the presence of the arc $(\alpha(u), u)$. Further, since $(\gamma(l), u) \in S'$ but $(\gamma(l), u) \notin E_{S'}$, we can conclude that $\gamma(l)$ and $u$ both belong to the same partition of nodes created by $E_{S'}$. Hence, any $\theta - \gamma(l)$ path in $G(F \cup S')$ must pass through one of the arcs in $E_{S'}$. An expression for $L_{S'}(\theta, \gamma(l))$ in terms of the arcs $(v, w) \in E_{S'}$ is provided in Eqn (B.8). The validity of Eqn (B.8) is shown later in this section. The expression in Eqn (B.8) assumes that the quantity $L_{S\setminus\{(\gamma(u),\delta_S(u))\}}(w, \gamma(l))$ was pre-computed. In Section B.2, we provide details about efficiently computing this value and several others from Table B.7.

The computation of the longest $\gamma(l) - \Lambda$ path cost in $G(F \cup S')$ can be computed by considering the longest path through each successor of $\gamma(l)$ (i.e. $\gamma(\gamma(l))$, $u$) in $G(F \cup S')$ separately. Hence, we can write $L_{S'}(\gamma(l), \Lambda)$ as shown in Eqn (B.9). From Lemma 1, we know that the set of all $\gamma(\gamma(l)) - \Lambda$ paths is identical for both $G(F \cup S)$ and $G(F \cup S')$, and so we can simplify Eqn (B.9) by replacing $L_{S'}(\gamma(\gamma(l), \Lambda)$ by $L_S(\gamma(\gamma(l), \Lambda)$ to obtain Eqn (B.10). To compute the longest $u - \Lambda$ path cost in $G(F \cup S')$, the paths through each successor of $u$ (i.e. $\gamma(u)$, $\delta_S(\alpha(u))$) in $G(F \cup S')$ are separately considered, and the expression for the path cost is shown in Eqn (B.11). See validity of Eqn (B.11) is shown later in this section.

**Computation of $L_{S'}^{-\gamma(l)}(\theta, \Lambda)$**

We compute $L_{S'}^{-\gamma(l)}(\theta, \Lambda)$ by analyzing the longest path through each arc in the cut $E_{S'}$ which does not pass through $\gamma(l)$. The expressions required for the computation of $L_{S'}^{-\gamma(l)}(\theta, \Lambda)$ are provided in Eqns (B.14) and (B.15).

In Eqn (B.14) we consider 2 cases. In the first case, the cut arc used is $(\alpha(u), u)$, and in the second case, the cut arc used is any arc other than $(\alpha(u), u)$. The longest path cost passing through a cut arc other than $(\alpha(u), u)$ is provided in Eqn (B.15), and its validity is shown later in this section.

## B.2 Complexity of makespan computations for feasible *N4* neighbors obtained from a single critical block

For any critical block $cB = (u_1, u_2, \ldots, u_n, l)$ we analyze the aggregated complexity of computing the makespan of all feasible *N4* neighbors generated by forward moves. Without loss of generality, we assume that only $h$ of the $n$ forward moves resulted in feasible selections. Recall that the makespan computation costs can be decomposed into 2 parts, namely, computation of the cut $E_{S'}$ for each feasible selection generated by a forward move, and the computation of quantities mentioned in Table B.7.

The cut $E_{S'}$ defined in Eqn (B.5) is with respect to operation $u$. So corresponding to each $u \in \{u_1, \ldots, u_n\}$ for which the forward move results in a feasible selection, a different $\theta - \Lambda$ cut is required. $E_{S'}$ can be computed in $\mathcal{O}\left(\mathbf{M}\,\log_2(\mathbf{J}) + \mathbf{J}\,\log_2(\mathbf{M})\right)$ number of elementary operations via binary search, assuming the complexity of accessing $T_S^{-1}(\cdot)$ is $\mathcal{O}(1)$.

All expressions of the form $L_S(\theta, o)$ and $L_S(o, \Lambda)$ in Table B.7 are already available as inputs. Values for expressions like $L_S(\cdot, \gamma(l))$, $L_S^{\gamma(l)}(\cdot, \Lambda)$, $L_S^{-\gamma(l)}(\cdot, \Lambda)$ can be obtained by computing those values once for all nodes in the set $\{o \,|\, o \in \bar{O}, T_S^{-1}(u_1) \leq T_S^{-1}(o) \leq T_S^{-1}(\gamma(l))\}$. Computing those values for all nodes in the set can be performed in $\mathcal{O}(TL(cB))$ complexity, where $TL(cB) = T_S^{-1}(\gamma(l)) - T_S^{-1}(u_1)$ is the (topological) length of the critical block $cB$.

Values for longest path expressions in Table B.7 that do not pass through certain nodes or arcs (e.g., quantities such as $L^{-\gamma(l)}_{S\setminus(\gamma(u),\delta_S(u))}(\cdot, \Lambda)$, $L_{S\setminus(\gamma(u),\delta_S(u))}(\cdot, \gamma(l))$) depend on the operation $u$ that is moved. From Table B.7, it can be inferred that these quantities are only needed when the head of an arc in the cut belongs to the set $Y(u) = \{o | o \in \bar{O},\, T_S^{-1}(u) \leq T_S^{-1}(o) \leq T_S^{-1}(\gamma(u))\}$. We can compute those quantities for all nodes in $Y(u)$ upfront in $\mathcal{O}(T_S^{-1}(\gamma(u)) - T_S^{-1}(u))$ complexity, and just read the appropriate value on a need basis. For critical block $cB = (u_1, u_2, \ldots, u_n, l)$, the sets $Y(u_i)$ are pairwise disjoint since $T_S^{-1}(\gamma(u_i)) < T_S^{-1}(u_{i+1})$. The aggregated complexity for computing these quantities for all the $h$ feasible selections obtained by forward moves can be performed in $\mathcal{O}(TL(cB))$ computations.

Putting it all together, the aggregated complexity of makespan computations for all $h$ feasible neighbors is $\mathcal{O}\left(TL(cB) + (h\,(\mathbf{M}\log_2(\mathbf{J}) + \mathbf{J}\log_2(\mathbf{M})))\right)$. The significance of this result can be understood in comparison to the complexity of feasibility recovery procedures in Section 4.7. The complexity of recovering a feasible solution by applying the job insertion procedure is comparable to the aggregated complexity of computing the makespan of all the feasible *N4* neighbors of a solution.

## Proofs for equations in Table B.7

**Lemma 6.** *1. For any $o \in \tilde{O}$ s.t. $T_S^{-1}(o) > T_S^{-1}(\gamma(l))$, we have $L_{S'}(o, \Lambda) = L_S(o, \Lambda)$.*

*2. For any $o \in \tilde{O}$ s.t. $T_S^{-1}(o) < T_S^{-1}(u)$, we have $L_{S'}(\theta, o) = L_S(\theta, o)$.*

*Proof.* Immediate from Lemma 1. □

**Lemma 7.** *For all $(v, w) \in E_{S'} \setminus (\alpha(u), u)$ ($E_{S'}$ defined in Eqn (B.5)), no path from $w$ to $\gamma(\beta_S(u))$ exists in $G(F \cup S')$.*

*Proof.* It is clear from the definition of $E_{S'}$ that $\forall\,(v,w) \in E_{S'} \backslash (\alpha(u),u)$, we have $T_S^{-1}(w) > T_S^{-1}(u) > T_S^{-1}(\gamma(\beta_S(u)))$, and so no $w - \gamma(\beta_S(u))$ exists in $G(F \cup S)$. Further, notice that $R_S(w) \subseteq \{o \in \tilde{O} | o \overset{T_S}{\succeq} w\}$. So if there exists a $w - \gamma(\beta_S(u))$ path in $G(F \cup S')$, then some arc $(x,y)$ with $T_S^{-1}(x) > T_S^{-1}(u)$ and $T_S^{-1}(y) < T_S^{-1}(u)$ needs to have been added to obtain $S'$ from $S$ in order to create a $w - \gamma(\beta_S(u))$ path. However, no such $(x,y)$ was added to obtain $S'$ from $S$. Hence, no $w - \gamma(\beta_S(u))$ path exists in $G(F \cup S')$. □

**Proof for Eqn (B.8) i.e. computation of $L_{S'}(\theta, \gamma(l))$**

The quantity $L_{S'}(\theta, \gamma(l))$ can be computed by computing the longest path through each arc $(v,w) \in E_{S'}$, where $E_{S'}$ is defined in Eqn (B.5).

First observe that if $T_S^{-1}(w) > T_S^{-1}(\gamma(l))$ or $w = u$ or $w = \gamma(u)$, we first show that no path from $w$ to $\gamma(l)$ exists in $G(F \cup S')$. The conditions mentioned is the same as those mentioned in the first case in Eqn (B.8).

• If $T_S^{-1}(w) > T_S^{-1}(\gamma(l))$, from Lemma 1 it follows that $R_{S'}(w) = R_S(w)$, but $\gamma(l) \notin R_S(w)$ and so $\gamma(l) \notin R_{S'}(w)$.

• If $w = u$ or $w = \gamma(u)$, observe that since $u, \gamma(u) \overset{T_{S'}}{\succ} \gamma(l)$ due to arc $(\gamma(l), u) \in S'$, and so $\gamma(l) \notin R_{S'}(w)$.

So it is sufficient to only consider arcs $(v,w)$ in $E_{S'}$ s.t. $\gamma(l) \overset{T_S}{\succ} w$ and $w \neq u, \gamma(u)$ for computing $L_{S'}(\theta, \gamma(l))$. Further, for any arc $(v,w) \in E_{S'}$ satisfying $\gamma(l) \overset{T_S}{\succ} w$ and $w \neq u, \gamma(u)$, the reader can easily verify from the definition of $E_{S'}$ that $u \overset{T_S}{\succ} v$, and so we can apply Lemma 6 to claim that $L_{S'}(\theta, v) = L_S(\theta, v)$. We will use this result implicitly in the proof below.

We compute $L_{S'}(\theta, \gamma(l))$ by analyzing the following 2 cases:

• $T_S^{-1}(u) < T_S^{-1}(w) < T_S^{-1}(\gamma(u))$
Consider any path $p$ in $G(F \cup S')$ from $w$ to $\gamma(l)$. Suppose $p$ does not exist in $G(F \cup S)$, then it must be the case that $(\gamma(\beta_S(u)), \delta_S(u))$ occurs on $p$ [1]. However, the existence of the $w - \gamma(\beta_S(u))$ sub-path in $p$ will contradict Lemma 7. So $p$ only contains arcs that are also present in $G(F \cup S)$, and so $p$ is present in $G(F \cup S)$ as well.

If $p''$ is some $w - \gamma(l)$ path in $G(F \cup S)$ which does not contain the arc $(\gamma(u), \delta_S(u))$, we argue that $p''$ must also be present in $G(F \cup S')$. Suppose $p''$ does not occur in $G(F \cup S')$, then observe that the arc $(\gamma(\beta_S(u), u)$ must be present in $p''$ [1]. However since $w \overset{T_S}{\succ} \gamma(\beta_S(u))$ by assumption, no path from $w$ to $\gamma(\beta_S(u))$ exists in $G(F \cup S)$. hence $p''$ exists in $G(F \cup S')$ as well. From the discussions above, we can conclude that the set of all $w - \gamma(l)$ paths in

[1] Depending on the context, whenever arcs from the set $\{(\gamma(\beta_S(u)), u)\,, (\gamma(u), \delta_S(u)), (\gamma(l), \delta_S(l))\}$ in case of $S$ (arcs from the set $\{(\gamma(\beta_S(u)), \delta_S(u)), (\gamma(l), u), (\gamma(u), \delta_S(l))\}$ in case of $S'$) have been omitted from the discussion, then the reader should be able to infer that the omitted arcs could not have occurred on the path under consideration. The reader will be able to make that inference by noticing that the presence of any omitted arc on the path under consideration will trivially imply that $S$ or $S'$ (will be clear from context) is inconsistent

$G(F \cup S')$ is identical to the set of all $w - \gamma(l)$ paths in $G(F \cup S)$ where none of the paths in the latter set contain the arc $(\gamma(u), \delta_S(u))$. So we can claim that:

$$L_{S'}(\theta, \gamma(l)) = L_S(\theta, v) + l(v, w) + L_{S \setminus (\gamma(u), \delta_S(u))}(w, \gamma(l))$$

• If $T_S^{-1}(\gamma(u)) < T_S^{-1}(w) \leq T_S^{-1}(\gamma(l))$.
The cost corresponding to the longest $\theta - \gamma(l)$ path through arc $(v, w)$ in $G(F \cup S')$ is:

$$L_{S'}(\theta, \gamma(l)) = L_S(\theta, v) + l(v, w) + L_{S'}(w, \gamma(l))$$

We will show that $L_{S'}(w, \gamma(l))$ can be replaced by $L_S(w, \gamma(l))$. To show the result, it is sufficient if we prove that the set of all $w - \gamma(l)$ paths are identical for both $G(F \cup S)$ and $G(F \cup S')$.

Suppose $p$ is a $w - \gamma(l)$ path in $G(F \cup S)$, observe that $p$ cannot contain the arcs $(\gamma(\beta_S(u)), u)$, $(\gamma(u), \delta_S(u))$, since $w \overset{T_S}{\succ} \gamma(u), \gamma(\beta_S(u))$. Also, $(\gamma(l), \delta_S(l))$ clearly cannot be present in $p$. All arcs present in $p$ also occur in $G(F \cup S')$, and so $p$ is present in $G(F \cup S')$.

Suppose $p'$ is a $w - \gamma(l)$ path in $G(F \cup S')$, then observe that $p'$ cannot contain the arc $(\gamma(u), \delta_S(l))$ since $\gamma(u) \overset{T_{S'}}{\succ} \gamma(l)$. As $S'$ is consistent, obviously $(\gamma(l), u)$ cannot occur in $p'$ either. If $p'$ contains the arc $(\gamma(\beta_S(u)), \delta_S(u))$, the $w - \gamma(\beta_S(u))$ sub-path in $p'$ will contradict Lemma 7. So all arcs in $p'$ are present in $G(F \cup S')$ too.

**Lemma 8.** *The set of all paths from $\gamma(\gamma(u))$ to $\Lambda$ is identical for both $G(F \cup S)$ and $G(F \cup S')$.*

*Proof.* Assume the set of all paths from $\gamma(\gamma(u))$ to $\Lambda$ in $G(F \cup S)$ and $G(F \cup S')$ are not identical. Then there exists a path $p$ s.t. $p$ either occurs on $G(F \cup S)$ or $G(F \cup S')$ but not in both graphs simultaneously. $\gamma(\beta_S(u)), \gamma(u), \gamma(l)$ are the only nodes whose outgoing arcs differ in $G(F \cup S)$ and $G(F \cup S')$, and so at least one among them should have appeared in $p$. Among $\gamma(\beta_S(u)), \gamma(u), \gamma(l)$, whichever node occurs the earliest, observe that the sub-path from $\gamma(\gamma(u))$ to that node must be present in both $G(F \cup S)$ and $G(F \cup S')$. If $\gamma(l)$ occurs the earliest on $p$, then it would contradict the consistency of $S'$, since now there is a path from $u$ to $\gamma(l)$, but we know that $\gamma(l) \overset{T_{S'}}{\succ} u$ (due to arc $(\gamma(l), u) \in S'$). The reason why $\gamma(u)$ cannot appear in $p$ is obvious. If $\gamma(\beta_S(u))$ occurs the earliest on $p$, then we would violate the consistency of $S$ since $\gamma(\gamma(u)) \overset{T_S}{\succ} u \overset{T_S}{\succ} \gamma(\beta_S(u))$. □

**Lemma 9.** *For any $o \in \tilde{O}$, s.t. $T_S^{-1}(u) < T_S^{-1}(o) < T_S^{-1}(\gamma(u))$, the set of all paths from $o$ to $\gamma(u)$ is identical for both $G(F \cup S)$ and $G(F \cup S')$.*

*Proof.* Can be shown very similar to the proof of Lemma 8. □

**Lemma 10.** *The set of all paths from $\delta_S(\alpha(u))$ to $\gamma(u)$ are identical for both $G(F \cup S)$ and $G(F \cup S')$.*

*Proof.* To prove the Lemma, we consider 2 cases:

1. If $\delta_S(\alpha(u)) \overset{T_S}{\succ} \gamma(u)$

Observe that there is no path from $\delta_S(\alpha(u))$ to $\gamma(u)$ in $G(F \cup S)$, since $\delta_S(\alpha(u)) \overset{T_S}{\succ} \gamma(u)$ by assumption.
Assume there exists some path $p$ from $\delta_S(\alpha(u))$ to $\gamma(u)$ in $G(F \cup S')$. We know that $p$ cannot contain the arcs $(\gamma(\beta_S(u)), \delta_S(u)), (\gamma(l), u)$ since $\delta_S(\alpha(u)) \overset{T_{S'}}{\succ} u \overset{T_{S'}}{\succ} \gamma(\beta_S(u))$. Likewise, $p$ cannot contain the arc $(\gamma(u), \delta_S(l))$, due to consistency of $S'$. Hence, no path from $\delta_S(\alpha(u))$ to $\gamma(u)$ exists in $G(F \cup S')$, since we are left with no arcs in $G(F \cup S')$ that are absent from $G(F \cup S)$.

2. If $\gamma(u) \overset{T_S}{\succ} \delta_S(\alpha(u))$

Observe that $\delta_S(\alpha(u)) \overset{T_S}{\succ} u$, so we can apply Lemma 9 to claim that the set of paths from $\delta_S(\alpha(u))$ to $\gamma(u)$ are identical for both $G(F \cup S)$ and $G(F \cup S')$.

□

## Proof for Eqn (B.11) i.e. computation of $L_{S'}(u, \Lambda)$

We can decompose the set of all paths $u - \Lambda$ in $G(F \cup S')$ into 2 cases, paths that pass through $(u, \delta_S(\alpha(u)))$ and those that pass through $(u, \gamma(u))$. We compute the longest path for each of those 2 cases separately:

• Longest path through $(u, \gamma(u))$:
Any $u - \Lambda$ path through the arc $(u, \gamma(u))$ should either contain $\delta_S(l)$ or $\gamma(\gamma(u))$. So the cost corresponding to the longest $u - \Lambda$ path through arc $(u, \gamma(u))$ is $l(u, \gamma(u)) + L_{S'}(\gamma(u), \Lambda)$ where:

$$L_{S'}(\gamma(u), \Lambda) = \max(l(\gamma(u), \delta_S(l)) + L_{S'}(\delta_S(l), \Lambda), l(\gamma(u), \gamma(\gamma(u))) + L_{S'}(\gamma(\gamma(u)), \Lambda))$$

From Lemma 6, we know that $L_{S'}(\delta_S(l), \Lambda) = L_S(\delta_S(l), \Lambda)$.
From Lemma 8, we can conclude that $L_{S'}(\gamma(\gamma(u)), \Lambda) = L_S(\gamma(\gamma(u)), \Lambda)$. In other words, we can claim that:

$$L_{S'}(\gamma(u), \Lambda) = L_S(\gamma(u), \Lambda) \tag{B.16}$$

• Longest path through $(u, \delta_S(\alpha(u)))$:
To compute the longest $u - \Lambda$ path through the arc $(u, \delta_S(\alpha(u)))$ we consider 2 cases. In the first case, we consider the situation when the path also contains $\gamma(u)$, and in the second case, the path does not contain $\gamma(u)$.

– Passing through $\gamma(u)$:
The longest $u - \Lambda$ path containing the arc $(u, \delta_S(\alpha(u)))$ and node $\gamma(u)$ in $G(F \cup S')$ can be computed as:

$$l(u, \delta_S(\alpha(u))) + L_{S'}(\delta_S(\alpha(u)), \gamma(u)) + L_{S'}(\gamma(u), \Lambda)$$

We can apply Lemma 10, to conclude that $L_{S'}(\delta_S(\alpha(u)), \gamma(u)) = L_S(\delta_S(\alpha(u)), \gamma(u))$. To obtain $L_{S'}(\gamma(u), \Lambda)$, we simply use the value computed in Eqn (B.16).

– Not passing through $\gamma(u)$:
Consider any $u-\Lambda$ path $p$ in $G(F\cup S')$ containing the arc $(u,\delta_S(\alpha(u)))$ and not containing the node $\gamma(u)$. Since $u \overset{T_{S'}}{\succ} \gamma(\beta_S(u)),\gamma(l)$, none among $\gamma(\beta_S(u)),\gamma(l)$ can occur on $p$, and so we can conclude that $p$ passes through arcs present in both $G(F\cup S)$ and $G(F\cup S')$.

Any path $p'$ from $u$ to $\Lambda$ in $G(F\cup S)$ containing the arc $(u,\delta_S(\alpha(u)))$ and not containing the node $\gamma(u)$ must be present in $G(F\cup S')$. $p'$ does not contain $\gamma(\beta_S(u))$ since $\delta_S(\alpha(u)) \overset{T_S}{\succ} \gamma(\beta_S(u))$. We next show that $p'$ cannot also contain $\gamma(l)$. Suppose $p'$ contains $\gamma(l)$, then the $u-\gamma(l)$ sub-path in $p'$ implies there is a path from $u$ to $\gamma(l)$ not containing the arc $(\gamma(u),\delta_S(u))$ in $G(F\cup S)$. By Theorem 2, the existence of the $u-\gamma(l)$ sub-path would have then implied that $S'$ is not a consistent selection, contrary to our assumption. Hence, $p'$ cannot contain $\gamma(l)$. Hence, $p'$ only contains arcs present in both $G(F\cup S)$ and $G(F\cup S')$.

From the discussions above, we conclude that the set of all $u-\Lambda$ paths containing the arc $(u,\delta_S(\alpha(u)))$ and not passing through the node $\gamma(u)$ is identical for both $G(F\cup S')$ and $G(F\cup S)$. So we can compute the required $u-\Lambda$ path satisfying our desired properties from $G(F\cup S)$ directly. To compute this quantity, we consider 2 cases:

* If $T_S^{-1}(\delta_S(\alpha(u)))>T_S^{-1}(\gamma(u))$, then there can be no path from $\delta_S(\alpha(u))$ to $\gamma(u)$ in $G(F\cup S)$. The required path cost for this case is:

$$l(u,\delta_S(\alpha(u)))+L_S(\delta_S(\alpha(u)),\Lambda)$$

* Otherwise, we can compute the cost for the longest $\delta_S(\alpha(u))-\Lambda$ path not passing through $\gamma(u)$ in $G(F\cup S)$, and denote it by $L_S^{-\gamma(u)}(\delta_S(\alpha(u)),\Lambda)$. The required path cost for this case is:

$$l(u,\delta_S(\alpha(u)))+L_S^{-\gamma(u)}(\delta_S(\alpha(u)),\Lambda)$$

## Proof for Eqn (B.15) i.e. computation of the longest $\theta-\Lambda$ path in $G(F\cup S')$ that does not pass through $\gamma(l)$ and the arc $(\alpha(u),u)$

We are interested in computing the longest $\theta-\Lambda$ path through arcs in $E_{S'}\backslash(\alpha(u),u)$, and not passing through the node $\gamma(l)$. Like in the earlier proofs, we compute the longest path by computing the longest path through each arc $(v,w)$ in the set $E_{S'}\backslash(\alpha(u),u)$, and not passing through $\gamma(l)$. To compute the required longest path, we perform the computation by analyzing 2 cases. But first, the reader can easily verify from the definition of $E_{S'}$ that $u \overset{T_S}{\succ} v$, and so we can apply Lemma 6 to claim that $L_{S'}(\theta,v)=L_S(\theta,v)$. The 2 cases to consider are:

• If $T_S^{-1}(u)<T_S^{-1}(w)\leq T_S^{-1}(\gamma(u))$
We compute this quantity by considering 2 cases. In the first case, we compute the longest $w-\Lambda$ path which is present in $G(F\cup S')$ but necessarily absent in $G(F\cup S)$. In the second case, we compute the longest $w-\Lambda$ path present in both $G(F\cup S)$ and $G(F\cup S')$.

– Suppose $p$ is a $w - \Lambda$ path which does not pass through $\gamma(l)$ in $G(F \cup S')$. Since we assumed that $p$ does not occur on $G(F \cup S)$, then it must be the case $p$ contains the arc $(\gamma(u), \delta_S(l))$. Clearly $(\gamma(l), u)$ cannot occur on $p$, and by Lemma 7 the arc $(\gamma(\beta_S(u)), \delta_S(u))$ cannot occur on $p$ either. By Lemma 9, the sub-path from $w$ to $\gamma(u)$ must be present in $G(F \cup S)$. Hence, the longest $w - \Lambda$ path cost satisfying the required conditions is:

$$L_S(\theta, v) + l(v, w) + L_S(w, \gamma(u)) + L_{S'}(\gamma(u), \Lambda)$$

where $L_{S'}(\gamma(u), \Lambda)$ can be obtained from Eqn (B.16).

– Suppose $p$ is a $w - \Lambda$ path present in both $G(F \cup S')$ and $G(F \cup S)$ such that $p$ does not contain $\gamma(l)$, then note that $p$ necessarily does not contain the arc $(\gamma(u), \delta_S(u))$ since the arc is absent in $S'$. Any $w-\Lambda$ path in $G(F \cup S)$ also does not contain the arc $(\gamma(\beta_S(u)), u)$ since $w \overset{T_S}{\succ} \gamma(\beta_S(u))$. Hence, we can conclude that any $w - \Lambda$ path in $G(F \cup S)$ which does not pass through $\gamma(l)$ and $(\gamma(u), \delta_S(u))$ must be present in $G(F \cup S')$, since such a path essentially passes only through arcs that are present in both $G(F \cup S)$ and $G(F \cup S')$. The cost of the longest $w - \Lambda$ path satisfying the desired properties is equal to the quantity $L^{-\gamma(l)}_{S \setminus (\gamma(u), \delta_S(u))}(w, \Lambda)$. So the cost of the longest $\theta - \Lambda$ path satisfying the required conditions is:

$$L_S(\theta, v) + l(v, w) + L^{-\gamma(l)}_{S \setminus (\gamma(u), \delta_S(u))}(w, \Lambda)$$

• If $T_S^{-1}(w) > T_S^{-1}(\gamma(u))$

Notice that any $w - \Lambda$ path $p$ not passing through $\gamma(l)$ in $G(F \cup S')$ cannot also contain $\gamma(u), \gamma(\beta_S(u))$. Suppose $p$ does contain at least one among $\gamma(u), \gamma(\beta_S(u))$, then the sub-path from $w$ to whichever node among $\gamma(u), \gamma(\beta_S(u))$ occurs earliest in $p$ must be present in $G(F \cup S)$ also, which will however contradict our assumption that $T_S^{-1}(w) > T_S^{-1}(\gamma(u))$. Hence, the path $p$ exists in $G(F \cup S)$ as well.

Conversely, any path from $w$ to $\Lambda$ not passing through $\gamma(l)$ in $G(F \cup S)$, passes through arcs which are also present in $G(F \cup S')$, since such a path passes through nodes which occur after $\gamma(u)$ in $T_S$. Among nodes in $T_S$ which occur after $\gamma(u)$, only the outgoing arcs of $\gamma(l)$ differs in $G(F \cup S)$ and $G(F \cup S')$.

Hence, the longest $w - \Lambda$ path cost not passing through $\gamma(l)$ denoted by $L^{-\gamma(l)}_{S'}(w, \Lambda)$ is equal to $L^{-\gamma(l)}_{S}(w, \Lambda)$.

## B.3 Makespan computation for the backward move

Like in the case of the forward move, we work with minimal representations of feasible selections. Assume a consistent selection $S$ for a BJS instance, and let $cB = (g, u_1, \ldots, u_n)$ be a critical block in some critical path of $G(F \cup S)$. For any $u \in \{u_1, \ldots, u_n\}$, we are interested in computing the makespan for the neighbor generated by the backward move which makes operation $u$ the machine predecessor of operation $f$, where $f = g$ if the incoming arc into $g$ on the critical path is a job arc, and $f = \beta_S(g)$ otherwise. Denote the selection generated by the backward move for $u$ as $S'$ and assume that $S'$ is consistent. Portions of the graphs $G(F \cup S), G(F \cup S')$ that are affected by the move are shown in Figures 4.3c, 4.3d. Like for

the forward move, we begin by providing a $\theta - \Lambda$ cut for $G(F \cup S')$, which makes use of the topological ordering $T_S$ and $u$. The complexity of makespan computation of backward moves will be very similar to that of froward moves, and so we will skip analyzing the complexity separately.

**Proposition 9.** *$E_{S'}$ defined in Eqn (B.17), is a valid $\theta - \Lambda$ cut for $G(F \cup S')$.*

*Proof.* The proof can be shown along similar lines to that of Proposition 8, and essentially follows from the fact that no arcs of the form $(x, y)$, with $T_S^{-1}(x) > T_S^{-1}(\gamma(u))$ and $T_S^{-1}(y) < T_S^{-1}(\gamma(u))$ were added to $S$ to obtain $S'$. □

We decompose makespan computation for a backward move into 2 parts:

1. Cost of the longest path in $G(F \cup S')$ passing through $f$, denoted by $L^f_{S'}(\theta, \Lambda)$.
2. Cost of the longest path in $G(F \cup S')$ not passing through $f$, denoted by $L^{-f}_{S'}(\theta, \Lambda)$.

**Computation of $L^f_{S'}(\theta, \Lambda)$**

Observe that:

$$L^f_{S'}(\theta, \Lambda) = L_{S'}(\theta, f) + L_{S'}(f, \Lambda)$$

We begin with the computation of $L_{S'}(\theta, f)$. By considering the predecessors of $f$ in $G(F \cup S')$, $L_{S'}(\theta, f)$ can be written as shown in Eqn (B.19). By virtue of Lemma 3, we know that $\bar{P}_S(\alpha(f)) = \bar{P}_{S'}(\alpha(f))$, so we can replace $L_{S'}(\theta, \alpha(f))$ in Eqn (B.19) by $L_S(\theta, \alpha(f))$ to obtain Eqn (B.20). The expression for $L_{S'}(\theta, \gamma(u))$ in Eqn (B.20) is provided in Eqns (B.21) and (B.22). For computing $L_{S'}(f, \Lambda)$, we make use of the cut $E_{S'}$ defined in Eqn (B.17), the expression for $L_{S'}(f, \Lambda)$ is provided in Eqn (B.23). The proof of validity for Eqns (B.21)-(B.23) are provided later in this section.

**Computation of $L^{-f}_{S'}(\theta, \Lambda)$**

The computation of the longest path cost that does not pass through $f$, is performed by analyzing the longest path through each arc in the cut $E_{S'}$ that does not pass through $f$. The expression for $L^{-f}_{S'}(\theta, \Lambda)$ is provided in Eqns (B.24) and (B.25). The validity of the expressions are shown later in this section.

The makespan computation analysis for backward moves parallels the analysis for forward moves. It can be shown that the aggregated complexity of makespan computations for all $h$ feasible selections obtained by backward moves on critical block $cB$ is
$\mathcal{O}\left(TL(cB) + \left(h\left(\mathbf{M}\log_2(\mathbf{J}) + \mathbf{J}\log_2(\mathbf{M})\right)\right)\right)$, where $TL(cB) = T_S^{-1}(\gamma(u_n)) - T_S^{-1}(f)$.

## Proofs for equations in Table B.8

**Lemma 11.** *1. For $o \in \tilde{O}$ such that $T_S^{-1}(o) < T_S^{-1}(f)$, we have $L_{S'}(\theta, o) = L_S(\theta, o)$.*

*2. For $o \in \tilde{O}$ such that $T_S^{-1}(o) > T_S^{-1}(\gamma(u))$, we have $L_{S'}(o, \Lambda) = L_S(o, \Lambda)$.*

**Table B.8:** Quantities for makespan computation of feasible *N4* neighbors obtained by *N4* backward move

$$E_{S'} = \{(v,w)|(v,w) \in F \cup S,\, T_S^{-1}(v) < T_S^{-1}(\gamma(u)), T_S^{-1}(w) > T_S^{-1}(\gamma(u))\} \cup \{(\gamma(\beta_S(u)), \delta_S(u))\} \cup \{(\gamma(u), \gamma(\gamma(u)))\} \tag{B.17}$$

$$\textbf{Makespan}: L_{S'}(\theta,\Lambda) = \max\left(L^f_{S'}(\theta,\Lambda), L^{-f}_{S'}(\theta,\Lambda)\right) \tag{B.18}$$

$$L_{S'}(\theta,f) = \max(L_{S'}(\theta,\alpha(f)) + l(\alpha(f),f), L_{S'}(\theta,\gamma(u)) + l(\gamma(u),f)) \tag{B.19}$$

$$L_{S'}(\theta,f) = \max(L_S(\theta,\alpha(f)) + l(\alpha(f),f), L_{S'}(\theta,\gamma(u)) + l(\gamma(u),f)) \tag{B.20}$$

$$L_{S'}(\theta,\gamma(u)) = \max(L_{S\setminus\{(\gamma(\beta_S(f)),f),(\gamma(\beta_S(u)),u)\}}(\theta,\gamma(u)), c(u)) \tag{B.21}$$

$$c(u) = L_S(\theta,\gamma(\beta_S(f))) + l(\gamma(\beta_S(f)),u) + \max(l(u,\gamma(u))\,,\, l(u,\delta_S(\alpha(u))) + L_S(\delta_S(\alpha(u)),\gamma(u))) \tag{B.22}$$

$$L_{S'}(f,\Lambda) = \max_{(v,w)\in E_{S'}} C_1^f(v,w) = \begin{cases} L_S(f,v) + l(v,w) + L_S(w,\Lambda) & \text{, if } T_S^{-1}(f) \le T_S^{-1}(v) < T_S^{-1}(u) \\ L_{S\setminus(\gamma(\beta_S(u)),u)}(f,v) + l(v,w) + L_S(w,\Lambda) & \text{, if } T_S^{-1}(u) \le T_S^{-1}(v) < T_S^{-1}(\gamma(u)) \\ -\infty & \text{, otherwise.} \end{cases} \tag{B.23}$$

$$L^{-f}_{S'}(\theta,\Lambda) = \max_{(v,w)\in E_{S'}} C_2^{-f}(v,w) = \begin{cases} L_S(\theta,v) + l(v,w) + L_S(w,\Lambda) & \text{, if } T_S^{-1}(v) < T_S^{-1}(f) \\ L^{-f}_S(\theta,v) + l(v,w) + L_S(w,\Lambda) & \text{, if } T_S^{-1}(f) < T_S^{-1}(v) < T_S^{-1}(u) \\ C_3(u) & \text{, if } T_S^{-1}(u) \le T_S^{-1}(v) \le T_S^{-1}(\gamma(u)) \end{cases} \tag{B.24}$$

$$C_3(u) = \max(L_S(\theta,\gamma(\beta_S(f))) + l(\gamma(\beta_S(f)),u) + L_S(u,v) + l(v,w) + L_S(w,\Lambda), L^{-f}_{S\setminus(\gamma(\beta_S(u)),u)}(\theta,v) + l(v,w) + L_S(w,\Lambda)) \tag{B.25}$$

*Proof.* Immediate from Lemma 3. □

**Lemma 12.** *For any $o \in \tilde{O}$ s.t. $T_S^{-1}(u) \leq T_S^{-1}(o) < T_S^{-1}(\gamma(u))$, the set of all $o - \gamma(u)$ paths are identical for both $G(F \cup S)$ and $G(F \cup S')$.*

*Proof.* Let $p$ be any $o - \gamma(u)$ path in $G(F \cup S')$. Observe that since $f, \delta_S(u) \overset{T_{S'}}{\succ} \gamma(u)$, arcs $(\gamma(u), f), (\gamma(\beta_S(u)), \delta_S(u))$ cannot be present in $p$. Suppose $p$ contains the arc $(\gamma(\beta_S(f)), u)$, then observe that the sub-path from $o$ to $\gamma(\beta_S(f))$ must occur in $G(F \cup S)$ as well, since we are left with no arcs that occur in $G(F \cup S')$ but not in $G(F \cup S)$. However, no path from $o$ to $\gamma(\beta_S(f))$ exists in $G(F \cup S)$, since $o \overset{T_S}{\succ} u \overset{T_S}{\succ} \gamma(\beta_S(f))$. So all arcs in $p$ are present in $G(F \cup S)$ as well.

Conversely, suppose $p'$ is a $o - \gamma(u)$ path in $G(F \cup S)$. Since we know that $o \overset{T_S}{\succ} u \overset{T_S}{\succ} f$, we can immediately conclude that the arcs $(\gamma(\beta_S(f)), f), (\gamma(\beta_S(u)), u)$ cannot be present in $p'$. Further, since $\delta_S(u) \overset{T_S}{\succ} \gamma(u)$, the arc $(\gamma(u), \delta_S(u))$ cannot be present in $p'$. Hence, we can conclude that all arcs in $p'$ are also present in $G(F \cup S')$. □

**Lemma 13.** *The set of all paths from $\delta_S(\alpha(u))$ to $\gamma(u)$ are identical for both $G(F \cup S)$ and $G(F \cup S')$.*

*Proof.* To prove the Lemma, we consider 2 cases:

1. If $\delta_S(\alpha(u)) \overset{T_S}{\succ} \gamma(u)$

Observe that there is no path from $\delta_S(\alpha(u))$ to $\gamma(u)$ in $G(F \cup S)$, since $\delta_S(\alpha(u)) \overset{T_S}{\succ} \gamma(u)$ by assumption.
Assume there exists some $\delta_S(\alpha(u)) - \gamma(u)$ path $p$ in $G(F \cup S')$. We know that $p$ cannot contain the arc $(\gamma(\beta_S(f)), u)$ since $\delta_S(\alpha(u)) \overset{T_{S'}}{\succ} u$. Likewise, $p$ cannot contain the arcs $(\gamma(\beta_S(u)), \delta_S(u)), (\gamma(u), f)$, since $\delta_S(u), f \overset{T_{S'}}{\succ} \gamma(u)$. Hence, no path from $\delta_S(\alpha(u))$ to $\gamma(u)$ exists in $G(F \cup S')$, since we are left with no arcs present in $G(F \cup S')$ which are absent from $G(F \cup S)$, and $G(F \cup S)$ has no $\delta_S(\alpha(u)) - \gamma(u)$ path.

2. If $\gamma(u) \overset{T_S}{\succ} \delta_S(\alpha(u))$

Observe that $\delta_S(\alpha(u)) \overset{T_S}{\succ} u$, so we can apply Lemma 12 to claim that the set of $\delta_S(\alpha(u)) - \gamma(u)$ paths are identical for both $G(F \cup S)$ and $G(F \cup S')$. □

**Lemma 14.** *For $o \in \tilde{O}$ s.t. $T_S^{-1}(f) < T_S^{-1}(o) < T_S^{-1}(u)$, the set of all $f - o$ paths are identical for both $G(F \cup S)$ and $G(F \cup S')$.*

*Proof.* Can be shown very similar to the proof of Lemma 12. □

**Proof for Eqn (B.21) i.e. computation of $L_{S'}(\theta, \gamma(u))$**

To compute $L_{S'}(\theta, \gamma(u))$, we will consider 2 cases. In the first case, we compute the longest $\theta - \gamma(u)$ path which is present in $G(F \cup S')$ but necessarily absent in $G(F \cup S)$. In the second case, we compute the longest $\theta - \gamma(u)$ path present in both $G(F \cup S)$ and $G(F \cup S')$.

• Suppose $p'$ is a $\theta - \gamma(u)$ path in $G(F \cup S')$ not present in $G(F \cup S)$, then it must be the case that $p'$ contains the arc $(\gamma(\beta_S(f)), u)$. Obviously $p'$ cannot contain the arcs $(\gamma(u), f)$, $(\gamma(\beta_S(u)), \delta_S(u))$, since $f, \delta_S(u) \overset{T_{S'}}{\succ} \gamma(u)$. So the cost of the longest $\theta - \gamma(u)$ path in $G(F \cup S')$ not present in $G(F \cup S)$ is:

$$L_{S'}(\theta, \gamma(\beta_S(f))) + l(\gamma(\beta_S(f)), u) + \\ \max(l(u, \gamma(u)), l(u, \delta_S(\alpha(u))) + L_{S'}(\delta_S(\alpha(u)), \gamma(u)) \tag{B.26}$$

Eqn (B.26) can be further simplified as follows. By Lemma 11 we have $L_{S'}(\theta, \gamma(\beta_S(f))) = L_S(\theta, \gamma(\beta_S(f)))$. By Lemma 13 we have $L_{S'}(\delta_S(\alpha(u)), \gamma(u)) = L_S(\delta_S(\alpha(u)), \gamma(u))$.

• Consider any $\theta - \gamma(u)$ path $p$ present in both $G(F \cup S')$ and $G(F \cup S)$. Then, it must be the case that $p$ does not contain the arcs $(\gamma(\beta_S(f)), f), (\gamma(\beta_S(u), u))$, since those arcs are not present in $S'$. Observe that any $\theta - \gamma(u)$ path in $G(F \cup S)$ not passing through the arcs $(\gamma(\beta_S(f)), f), (\gamma(\beta_S(u), u))$ must be present in $G(F \cup S')$, since all the arcs on the $\theta - \gamma(u)$ path are also present in $G(F \cup S')$. Note that the arc $(\gamma(u), \delta_S(u))$ cannot be present in any $\theta - \gamma(u)$ path in $G(F \cup S)$. So the cost of the longest $\theta - \gamma(u)$ path, where the path is present in both $G(F \cup S')$ and $G(F \cup S)$, is equal to the quantity $L_{S \setminus \{(\gamma(\beta_S(f)), f), (\gamma(\beta_S(u)), u)\}}(\theta, \gamma(u))$ .

**Proof for Eqn (B.23) i.e. computation of $L_{S'}(f, \Lambda)$**

Observe that $\forall (v, w) \in E_{S'}$ we have $T_S^{-1}(w) > T_S^{-1}(\gamma(u))$, and so $L_{S'}(w, \Lambda) = L_S(w, \Lambda)$, by Lemma 11. We will implicitly use this fact in the proof below.

It can be easily be shown that the cut $E_{S'}$ (defined in Eqn (B.17)) separates the nodes $f, \Lambda$. To compute $L_{S'}(f, \Lambda)$, we compute the longest path from $f$ to $\Lambda$ through each arc in the cut $E_{S'}$. For each arc $(v, w) \in E_{S'}$, we compute the longest path through $(v, w)$ by analyzing a few cases as shown below:

• If $v = f$
It is not hard to show that the only possibility for $w$ is $\delta_S(\alpha(f))$. The longest path cost in this case is given by:

$$l(v, w) + L_S(w, \Lambda)$$

• If $T_S^{-1}(f) < T_S^{-1}(v) < T_S^{-1}(u)$
From Lemma 14, we can see that the set of all paths from $f$ to $v$ must be identical for both $G(F \cup S)$ and $G(F \cup S')$. Hence, the longest $f - \Lambda$ path cost through $(v, w)$ is given by:

$$L_S(f, v) + l(v, w) + L_S(w, \Lambda)$$

• If $T_S^{-1}(u) \leq T_S^{-1}(v) < T_S^{-1}(\gamma(u))$
Suppose there is a path $p$ from $f$ to $v$ in $G(F \cup S')$ s.t. $p$ is absent in $G(F \cup S)$, then it must be the case that arc $(\gamma(\beta_S(u)), \delta_S(u))$ occurs in $p$ [2]. From Lemma 3, we know that $R_{S'}(\delta_S(u)) = R_S(\delta_S(u))$, but $f \notin R_S(\delta_S(u))$ since $\delta_S(u) \overset{T_S}{\succ} f$. Consequently, all $f - v$ paths in $G(F \cup S')$ are also present in $G(F \cup S)$.

Let $p'$ be any $f-v$ path present in both $G(F \cup S)$ and $G(F \cup S')$, then it must be the case that $p$ does not contain the arc $(\gamma(\beta_S(u)), u)$. Also, observe that the arcs $(\gamma(\beta_S(f)), f), (\gamma(u), \delta_S(u))$ could not have occurred on any $f - v$ path in $G(F \cup S)$, otherwise we will trivially contradict the consistency of $S$. Hence, we can conclude that the set of all $f - v$ paths in $G(F \cup S)$ that do not pass through the arc $(\gamma(\beta_S(u)), u)$ must be present in $G(F \cup S')$, as these paths only contain arcs present in $G(F \cup S')$ also.

From the discussions above, the cost of the longest $f - \Lambda$ path through arc $(v, w)$ can be computed from $G(F \cup S)$ as:

$$L_{S \setminus (\gamma(\beta_S(u)), u)}(f, v) + l(v, w) + L_S(w, \Lambda)$$

• If $v = \gamma(u)$ or $T_S^{-1}(v) < T_S^{-1}(f)$ or $T_S^{-1}(v) > T_S^{-1}(\gamma(u))$
In each of the 3 cases, we claim that there is no $f - v$ path in $G(F \cup S')$.
Observe that if $v = \gamma(u)$, then the presence of a $f - v$ path in $G(F \cup S')$ trivially contradicts the fact that $f \overset{T_{S'}}{\succ} \gamma(u)$ due to arc $(\gamma(u), f) \in S'$.
If $T_S^{-1}(v) < T_S^{-1}(f)$, then observe from Lemma 3 that $\bar{R}_{S'}(v) = \bar{R}_S(v)$. However since $f \overset{T_S}{\succ} v$ by assumption, we know that $f \notin \bar{R}_S(v)$, hence $f \notin \bar{R}_{S'}(v)$.
Finally from the definition of $E_{S'}$ in Eqn (B.17), we can verify that $T_S^{-1}(v) \leq T_S^{-1}(\gamma(u))$.

**Proof of Eqn (B.24) i.e. computation of $L_{S'}^{-f}(\theta, \Lambda)$**

To compute $L_{S'}^{-f}(\theta, \Lambda)$, we compute the longest path passing through each arc in the cut $E_{S'}$ which does not pass through $f$. Observe that $\forall (v, w) \in E_{S'}$ we have $T_S^{-1}(w) > T_S^{-1}(\gamma(u))$, and so $L_{S'}(w, \Lambda) = L_S(w, \Lambda)$, by Lemma 11. We will use this fact implicitly in the proof below. For each $(v, w) \in E_{S'}$, we compute the cost of the longest path through $(v, w)$ by considering the following cases:

• If $T_S^{-1}(v) < T_S^{-1}(f)$
From Lemma 11, we know that $L_{S'}(\theta, v) = L_S(\theta, v)$, and $L_{S'}(w, \Lambda) = L_S(w, \Lambda)$. The longest path through $(v, w)$ is:

$$L_S(\theta, v) + l(v, w) + L_S(w, \Lambda)$$

[2] Depending on the context, whenever arcs from the set $\{(\gamma(\beta_S(f)), f), (\gamma(\beta_S(u)), u), (\gamma(u), \delta_S(u))\}$ in case of $S$ (arcs from the set $\{(\gamma(\beta_S(f)), u), (\gamma(u), f), (\gamma(\beta_S(u)), \delta_S(u))\}$ in case of $S'$) have been omitted from the discussion, then the reader should be able to infer that the omitted arcs could not have occurred on the path under consideration. The reader will be able to make that inference by noticing that the presence of any omitted arc on the path under consideration will trivially imply that $S$ or $S'$ (will be clear from context) is inconsistent

• If $T_S^{-1}(f) < T_S^{-1}(v) < T_S^{-1}(u)$
Consider a $\theta - v$ path $p$ in $G(F \cup S')$ which does not pass through $f$. Obviously, $p$ cannot contain the arc $(\gamma(u), f)$. Arc $(\gamma(\beta_S(u)), \delta_S(u))$ cannot also be present in $p$, since, $\delta_S(u)$ and $v$ lie in different partitions created by the cut $E_{S'}$. Suppose $p$ did not exist in $G(F \cup S)$, then it must be the case that $p$ contains the arc $(\gamma(\beta_S(f)), u)$. Hence, it implies that there must exist a sub-path $p'$ from $u$ to $v$ in $p$, but $p'$ must also exist in $G(F \cup S)$ since all arcs we are left with in $G(F \cup S')$ for $p$ also occur in $G(F \cup S)$. However, $p'$ contradicts our assumption that $T_S^{-1}(v) < T_S^{-1}(u)$.

Now consider any $\theta - v$ path in $G(F \cup S)$ which does not pass through $f$. It can easily be shown that the path cannot contain the arcs $(\gamma(\beta_S(f)), f), (\gamma(\beta_S(u)), u)$ and $(\gamma(u), \delta_S(u))$. Hence the path only contains arcs that are also present in $G(F \cup S')$.

From the discussions above, we conclude that the set of all $\theta - \Lambda$ paths through $(v, w)$ not passing through $f$ is identical for both $G(F \cup S)$ and $G(F \cup S')$. Hence, the longest path through $(v, w)$ not passing through $f$ is given by:

$$L_S^{-f}(\theta, v) + l(v, w) + L_S(w, \Lambda)$$

• If $T_S^{-1}(u) \leq T_S^{-1}(v) \leq T_S^{-1}(\gamma(u))$
We compute the longest $\theta - \Lambda$ path through $(v, w)$ and not passing through $f$ by considering 2 cases. Consider any $\theta - v$ path $p$ in $G(F \cup S')$ not containing $f$, then:

– Suppose $p$ does not occur in $G(F \cup S)$. Observe that arc $(\gamma(u), f)$ cannot be present in $p$. Further $v$ and $\delta_S(u)$ lie on different partitions of the cut $E_{S'}$, so a path from $\delta_S(u)$ to $v$ in $G(F \cup S')$ does not exist, which thereby implies that arc $(\gamma(\beta_S(u)), \delta_S(u))$ is absent from $p$. Consequently, it must be the case that $p$ contains the arc $(\gamma(\beta_S(f)), u)$. So the cost of the longest $\theta - \Lambda$ path through $(v, w)$ not passing through $f$ is:

$$L_S(\theta, \gamma(\beta_S(f))) + l(\gamma(\beta_S(f)), u) + L_{S'}^{-f}(u, v) + l(v, w) + L_S(w, \Lambda) \tag{B.27}$$

We can replace $L_{S'}^{-f}(u, v)$ by $L_S(u, v)$ in Eqn (B.27) whose justification we provide below. Observe that all $u - v$ paths in $G(F \cup S')$ that do not contain $f$ essentially only contain arcs that are present in both $G(F \cup S')$ and $G(F \cup S)$. Conversely, any $u - v$ path in $G(F \cup S)$ cannot contain the arc $(\gamma(\beta_S(f)), f)$ since $u \overset{T_S}{\succ} f$, and neither the arc $(\gamma(u), \delta_S(u))$ since $\delta_S(u) \overset{T_S}{\succ} v$ [2].

Hence, the cost corresponding to the longest $\theta - \Lambda$ path through $(v, w)$ in $G(F \cup S')$ not passing through $f$ s.t. the path is absent from $G(F \cup S)$ is given by:

$$L_S(\theta, \gamma(\beta_S(f))) + l(\gamma(\beta_S(f)), u) + L_S(u, v) + l(v, w) + L_S(w, \Lambda)$$

– Suppose $p$ also occurs in $G(F \cup S)$, then obviously $p$ cannot contain the arc $(\gamma(\beta_S(u)), u)$. Observe that any $\theta - v$ path in $G(F \cup S)$ cannot contain the arc $(\gamma(u), \delta_S(u))$, since $\delta_S(u) \overset{T_S}{\succ} v$. Further, any $\theta - v$ path in $G(F \cup S)$ which does not contain the arc

$(\gamma(\beta_S(u)), u)$ and node $f$ must also be present in $G(F \cup S')$, since all the arcs in the $\theta - v$ path is also present in $G(F \cup S')$. Hence, the cost corresponding to the longest $\theta - \Lambda$ path through $(v, w)$ and not passing through $f$, but present in both $G(F \cup S)$ and $G(F \cup S')$ can be obtained from $G(F \cup S)$, and is given by:

$$L^{-f}_{S\setminus(\gamma(\beta_S(u)),u)}(\theta, v) + l(v, w) + L_S(w, \Lambda)$$

- $T_S^{-1}(v) > T_S^{-1}(\gamma(u))$

No arc $(v, w)$ with $T_S^{-1}(v) > T_S^{-1}(\gamma(u))$ exists in $E_{S'}$.

## B.4 Additional Proofs

**Lemma** 5. Given job $\mathcal{J}$ and a feasible schedule $Sch$ to the BJS problem instance, let $Q$ denote the set of all jobs that have a service interval strictly contained within the service interval of $\mathcal{J}$, then $|Q| \leq (\mathbf{M}-1)^2$.

*Proof.* W.l.o.g let us assume that the technological order for job $\mathcal{J}$ is $1, 2 \ldots, \mathbf{M}$. Let us denote the set of all jobs in our problem by $Jo$. Given a job $\bar{\mathcal{J}} \in Jo \backslash \mathcal{J}$ and a feasible schedule $Sch$ to the BJS problem instance, then define the following indicator function:

$$\mathbb{1}(i, \bar{\mathcal{J}}) = \begin{cases} 1, & \text{if } \mathcal{J} \text{ transitions from machine } i \text{ to } i+1 \text{ in the open interval } (St_{\bar{\mathcal{J}}}(Sch), Co_{\bar{\mathcal{J}}}(Sch)) \\ 0, & \text{otherwise} \end{cases} \tag{B.28}$$

If $\bar{\mathcal{J}} \in Q$, then observe that had $\mathcal{J}$ remained stationary on any one machine $m$ during the entire service interval of $\bar{\mathcal{J}}$, then $\bar{\mathcal{J}}$ could not have been serviced by $m$ earlier than the completion time of $\mathcal{J}$, so $\mathcal{J}$ must have transitioned at least one machine during the service interval of $\bar{\mathcal{J}}$. So, a trivial upper bound for the cardinality of $Q$ can be obtained by counting the number of jobs during whose service interval $\mathcal{J}$ transitioned at least one machine. In other words,

$$|Q| \leq \sum_{i=1}^{\mathbf{M}-1} \sum_{\bar{\mathcal{J}} \in Jo \backslash \mathcal{J}} \mathbb{1}(i, \bar{\mathcal{J}}) \tag{B.29}$$

Notice, that at the time instant in which job $\mathcal{J}$ transitions from machine $i$ to $i+1$, there can be at most $\mathbf{M}-1$ other jobs occupying other machines. So $\sum_{\bar{\mathcal{J}} \in Jo \backslash \mathcal{J}} \mathbb{1}(i, \bar{\mathcal{J}}) \leq \mathbf{M}-1$, $\forall i \in \{1, \ldots, \mathbf{M}-1\}$, and hence $|Q| \leq (\mathbf{M}-1)^2$. □

### Example showing the tightness of the bound on $|Q|$ in Lemma 5

W.l.o.g assume that the sequence of machines that job $\mathcal{J}$ traverses is in the order $1, 2, \ldots, \mathbf{M}$. We construct a set of jobs $Q_1$ with cardinality $\mathbf{M}-2$, such that the last machine required by every job in $Q_1$ is machine 1. We can set up a schedule $Sch$ such that, the start time of any job in $Q_1$ is later than $St_{\mathcal{J}}(Sch)$. Further, we can set up $Sch$ such that while $\mathcal{J}$ is being serviced by machine 1, the jobs in $Q_1$ can complete being serviced by all machines other than machine 1, and are waiting to be serviced by machine 1. We can force the jobs in $Q_1$ to wait by making the job $\mathcal{J}$ occupy machine 1 sufficiently long, thus blocking machine 1 from being accessed by jobs in $Q_1$. When job $\mathcal{J}$ transitions to machine 2, all jobs in $Q_1$ can then transition to machine 1 one after the other, and exit the system (i.e. no longer needs to occupy any machine), all this while job $\mathcal{J}$ occupies machine 2. For the no swap BJS case, the cardinality of $Q_1$ cannot exceed $\mathbf{M}-2$, because otherwise the only way $\mathcal{J}$ can advance to machine 2 will require swapping jobs between machines. Now while job $\mathcal{J}$ is at machine 2, we can set up yet another set of jobs $Q_2$ with cardinality $\mathbf{M}-2$, where the last machine on each job in $Q_2$ is machine 2, so $Q_1$ and $Q_2$ have no jobs in common. Like in the earlier case, jobs in $Q_2$ can exit the system only after $\mathcal{J}$ transitions to machine 3. We can set up $Sch$ for jobs in $Q_2$ exactly analogous to the way it was set up for jobs in $Q_1$. By repeating this

argument for each of the $\mathbf{M}-1$ machine to machine transitions for job $\mathcal{J}$, we conclude that $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_{\mathbf{M}-1}$, so $|Q| = (\mathbf{M}-1)(\mathbf{M}-2)$.

**Theorem** 5. A job's service interval can overlap with the service interval of at most $\mathbf{M}^2 - 1$ other jobs.

*Proof.* Let us denote the set of all jobs in our problem by $Jo$. Given any feasible schedule $Sch$, we will partition jobs in $Jo$ whose service interval overlaps with that of $\mathcal{J}$, as follows:

$$W_1 = \{\bar{\mathcal{J}} \in Jo \backslash \mathcal{J} | St_{\bar{\mathcal{J}}}(Sch) \leq St_{\mathcal{J}}(Sch) < Co_{\bar{\mathcal{J}}}(Sch) \text{ and } Co_{\bar{\mathcal{J}}}(Sch) \leq Co_{\mathcal{J}}(Sch)\}$$
$$W_2 = \{\bar{\mathcal{J}} \in Jo \backslash \mathcal{J} | St_{\bar{\mathcal{J}}}(Sch) > St_{\mathcal{J}}(Sch) \text{ and } Co_{\bar{\mathcal{J}}}(Sch) < Co_{\mathcal{J}}(Sch)\}$$
$$W_3 = \{\bar{\mathcal{J}} \in Jo \backslash \mathcal{J} | Co_{\mathcal{J}}(Sch) > St_{\bar{\mathcal{J}}}(Sch) \geq St_{\mathcal{J}}(Sch) \text{ and } Co_{\bar{\mathcal{J}}}(Sch) \geq Co_{\mathcal{J}}(Sch)\}$$
$$W_4 = \{\bar{\mathcal{J}} \in Jo \backslash \mathcal{J} | St_{\bar{\mathcal{J}}}(Sch) \leq St_{\mathcal{J}}(Sch) \text{ and } Co_{\bar{\mathcal{J}}}(Sch) \geq Co_{\mathcal{J}}(Sch)\}$$

From Lemma 5 we know that $|W_2| \leq (\mathbf{M}-1)^2$. Notice that the set of jobs in $W_1 \cup W_4$ are jobs other than $\mathcal{J}$, that must have been occupying some machine at time $St_{\mathcal{J}}(Sch)$, so $|W_1 \cup W_4| \leq \mathbf{M}-1$. Similarly, the jobs in the set $W_3 \cup W_4$ are the jobs (other than $\mathcal{J}$) that must have been occupying some machine at time $Co_{\mathcal{J}}(Sch)$, so $|W_3 \cup W_4| \leq \mathbf{M}-1$. So by union bound, we have:

$$|W_1 \cup W_2 \cup W_3 \cup W_4| \leq |W_1 \cup W_4| + |W_2| + |W_3 \cup W_4| \leq (\mathbf{M}-1)^2 + 2(\mathbf{M}-1) = \mathbf{M}^2 - 1 \ \square$$

**Proposition 10.** *Given any feasible selection $S$ and a set of jobs $Q$ s.t. $|Q| > \mathbf{M}$, we can identify a pair of jobs $\mathcal{J}_1, \mathcal{J}_2 \in Q$ s.t. there is a path in $G(F \cup S)$ from the last node belonging to job $\mathcal{J}_1$ (i.e. dummy operation of job $\mathcal{J}_1$) to the first node (i.e. first operation) belonging to $\mathcal{J}_2$.*

*Proof.* Let $Sch$ denote the start times computed for the nodes in $G(F \cup S)$ using the longest path algorithm. Note that we can always identify a job $\mathcal{J}_2 \in Q$ using selection $S$ s.t. the machine $m$ required to service the first operation $o$ in $\mathcal{J}_2$, services $o$ only after servicing all the operations belonging to the other jobs in $Q \backslash \mathcal{J}_2$ that also require $m$. We claim that there is some job $\mathcal{J}_1 \in Q \backslash \mathcal{J}_2$ s.t. there is a path from the last node belonging to job $\mathcal{J}_1$ to the first node belonging to $\mathcal{J}_2$. We will prove this fact by contradiction. Assume that no such $\mathcal{J}_1$ exists. Suppose we increase the processing duration of the last operation (not to be confused with the dummy operation) in the technological order of every job in $Q \backslash \mathcal{J}_2$, and compute a schedule $Sch'$ for selection $S$ with the new processing times using the longest path algorithm, then the increased processing duration values should have no effect on the start time of $\mathcal{J}_2$. In other words, we must have $St_{\mathcal{J}_2}(Sch') = St_{\mathcal{J}_2}(Sch)$. For some suitably large values of processing duration, we can ensure $Co_{\mathcal{J}}(Sch') > St_{\mathcal{J}_2}(Sch'), \forall \mathcal{J} \in Q \backslash \mathcal{J}_2$. It follows directly from our choice of $\mathcal{J}_2$ that $St_{\mathcal{J}}(Sch') < St_{\mathcal{J}_2}(Sch') \, \forall \, \mathcal{J} \in Q \backslash \mathcal{J}_2$. Hence, in schedule $Sch'$, we have managed to show that the service intervals of all the jobs in $Q$ strictly contains the time instance $St_{\mathcal{J}_2}(Sch') + \epsilon$, for some arbitrarily small $\epsilon > 0$. This contradicts the fact that in any feasible schedule at most $\mathbf{M}$ jobs can simultaneously be in state 3. $\square$

**Proposition 11.** *Given the predecessor buffer $f$ for selection $S_{\bar{\mathcal{J}}}$ (i.e. selection obtained by removing all arc pairs associated to job $\bar{\mathcal{J}}$ from a feasible selection $S$), operations belonging*

*to job $\mathcal{J}$ can appear in the predecessor maps for nodes in $\bar{O}_{\bar{\mathcal{J}}}$ belonging to at most $\mathbf{M}^2+\mathbf{M}-1$ jobs other than $\mathcal{J}$, where $\bar{O}_{\bar{\mathcal{J}}}$ is the set of nodes in $\bar{O}$ other than those belonging to job $\bar{\mathcal{J}}$.*

*Proof.* In buffer $f$, observe that operations belonging to job $\mathcal{J}$ can appear in the predecessor maps for nodes belonging to jobs in the sets $W_{\mathcal{J}}^{S}\backslash\{\bar{\mathcal{J}}\}$ and $W_{\mathcal{J}}^{S+}\backslash\{\bar{\mathcal{J}}\}$. Within the set $W_{\mathcal{J}}^{S}\backslash\{\bar{\mathcal{J}}\}$ (resp. $W_{\mathcal{J}}^{S+}\backslash\{\bar{\mathcal{J}}\}$) consider the set of jobs which contains an operation belonging to $\mathcal{J}$ in the predecessor map for at least one of its nodes and denote those set of jobs by $Q_1$ (resp. $Q_2$). We will compute an upper bound on $|Q|$ where $Q = Q_1 \cup Q_2$.

We claim that $|Q_2|$ cannot exceed $\mathbf{M}$. If suppose $|Q_2| > \mathbf{M}$, then we can identify a pair of jobs $\mathcal{J}_1, \mathcal{J}_2 \in Q_2$ satisfying the condition in Proposition 10. From the definition of $W_{\mathcal{J}}^{S+}$ we know that $\mathcal{J}_1$ is serviced later than $\mathcal{J}$ on every machine. Combining this fact with the definition in Eqn (4.5), the existence of the path (as mentioned in Proposition 10) from $\mathcal{J}_1$ to $\mathcal{J}_2$ implies that an operation belonging to $\mathcal{J}$ could not have appeared in predecessor maps for nodes belonging to $\mathcal{J}_2$. Hence, we can safely conclude that $|Q_2|$ cannot exceed $\mathbf{M}$. From Corollary 1 we know that $|Q_1| \leq \mathbf{M}^2 - 1$, hence $|Q| \leq \mathbf{M}^2 + \mathbf{M} - 1$. □

## B.5 Additional Tables

**Table B.9:** Missing results from Table 4.1 for 5 and 20 minutes. We followed the same conventions as we did for Table 4.1, where numbers in bold font indicate the makespan of the best solution for that instance, and underlined instances are those for which a new best solution is reported in this paper.

| Inst. | Size | Previous best known | N5 with JIFR-1 | | | | | | N4 with JIFR-1 & 2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 300 sec | | 1200 sec | | 1800 sec | | 300 sec | | 1200 sec | | 1800 sec | |
| | | | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg |
| LA01 | 10 × 5 | **881**[a,b,c] | **881** | 881 | 881 | 881 | 881 | 881 | **881** | 881 | 881 | 881 | 881 | 881 |
| LA02 | 10 × 5 | **900**[a,b,c] | **900** | 900 | 900 | 900 | 900 | 900 | **900** | 900 | 900 | 900 | 900 | 900 |
| LA03 | 10 × 5 | **808**[a,b,c] | **808** | 808 | 808 | 808 | 808 | 808 | **808** | 808 | 808 | 808 | 808 | 808 |
| LA04 | 10 × 5 | **859**[a,b,c] | **859** | 859 | 859 | 859 | 859 | 859 | **859** | 859 | 859 | 859 | 859 | 859 |
| LA05 | 10 × 5 | **732**[a,b,c] | **732** | 732 | 732 | 732 | 732 | 732 | **732** | 732 | 732 | 732 | 732 | 732 |
| LA06 | 15 × 5 | 1199[c] | **1194** | 1195.8 | 1194 | 1194.5 | 1194 | 1194.4 | **1194** | 1197.9 | 1194 | 1194.3 | 1194 | 1194.2 |
| LA07 | 15 × 5 | 1130[c] | **1127** | 1127.6 | 1127 | 1127 | 1127 | 1127 | **1127** | 1127.4 | 1127 | 1127 | 1127 | 1127 |
| LA08 | 15 × 5 | **1173**[c] | **1173** | 1179.2 | 1173 | 1173 | 1173 | 1173 | **1173** | 1177.6 | 1173 | 1173 | 1173 | 1173 |
| LA09 | 15 × 5 | 1311[b,c] | **1305** | 1306 | 1305 | 1305 | 1305 | 1305 | **1305** | 1306.9 | 1305 | 1305.1 | 1305 | 1305.1 |
| LA10 | 15 × 5 | 1222[c] | **1218** | 1221.6 | 1218 | 1220.4 | 1218 | 1220 | **1218** | 1221 | 1218 | 1220.8 | 1218 | 1220 |
| LA11 | 20 × 5 | 1591[c] | 1522 | 1562 | **1501** | 1542.2 | 1501 | 1538.7 | 1522 | 1561.6 | 1522 | 1554.5 | 1522 | 1543.6 |
| LA12 | 20 × 5 | 1399[c] | 1385 | 1394.5 | **1353** | 1376 | 1353 | 1375.9 | 1374 | 1399.8 | **1353** | 1388.2 | 1353 | 1383.7 |
| LA13 | 20 × 5 | 1538[c] | 1518 | 1541.1 | **1508** | 1518.7 | 1508 | 1518.7 | **1508** | 1537.9 | 1508 | 1525.1 | 1508 | 1523.4 |
| LA14 | 20 × 5 | **1544**[c] | 1582 | 1593.1 | 1562 | 1576.8 | 1557 | 1571.6 | 1548 | 1587.4 | 1548 | 1570 | 1548 | 1564.9 |
| LA15 | 20 × 5 | 1616[c] | **1565** | 1605.5 | 1565 | 1590.1 | 1565 | 1586.4 | 1587 | 1601 | 1578 | 1592.1 | 1578 | 1591.2 |
| LA16 | 10 × 10 | **1148**[a,b,c] | **1148** | 1148 | 1148 | 1148 | 1148 | 1148 | **1148** | 1148 | 1148 | 1148 | 1148 | 1148 |
| LA17 | 10 × 10 | **968**[a,b,c] | **968** | 968 | 968 | 968 | 968 | 968 | **968** | 968 | 968 | 968 | 968 | 968 |
| LA18 | 10 × 10 | **1077**[a,b,c] | **1077** | 1078.8 | 1077 | 1077 | 1077 | 1077 | **1077** | 1078.8 | 1077 | 1077 | 1077 | 1077 |
| LA19 | 10 × 10 | **1102**[a,b,c] | **1102** | 1102.4 | 1102 | 1102 | 1102 | 1102 | **1102** | 1104.3 | 1102 | 1102 | 1102 | 1102 |
| LA20 | 10 × 10 | **1118**[a,b,c] | **1118** | 1118 | 1118 | 1118 | 1118 | 1118 | **1118** | 1118 | 1118 | 1118 | 1118 | 1118 |
| LA21 | 15 × 10 | 1501[c] | **1483** | 1521.4 | 1483 | 1506.5 | 1483 | 1499 | 1507 | 1527.1 | **1483** | 1506.6 | 1483 | 1505 |
| LA22 | 15 × 10 | 1368[c] | **1328** | 1363.5 | 1328 | 1343.5 | 1328 | 1341 | **1328** | 1368.8 | 1328 | 1355.4 | 1328 | 1342.9 |
| LA23 | 15 × 10 | 1534[c] | **1475** | 1517.2 | 1475 | 1486.9 | 1475 | 1483.9 | **1475** | 1518.8 | 1475 | 1503.4 | 1475 | 1497.4 |
| LA24 | 15 × 10 | 1447[c] | **1402** | 1450.4 | 1402 | 1424.7 | 1402 | 1408.1 | **1402** | 1437.9 | 1402 | 1429.8 | 1402 | 1419.4 |
| LA25 | 15 × 10 | 1453[c] | 1418 | 1433.9 | 1409 | 1419.8 | 1409 | 1418.3 | 1414 | 1434.7 | **1406** | 1421.3 | 1406 | 1420.9 |
| LA26 | 20 × 10 | 1968[c] | 1906 | 1962.9 | 1885 | 1937.3 | 1885 | 1928.2 | **1870** | 1948.2 | 1870 | 1925.2 | 1870 | 1920.3 |
| LA27 | 20 × 10 | 2047[c] | 2006 | 2054.4 | **1933** | 2014 | 1933 | 2007.4 | 1992 | 2060.2 | 1987 | 2031.8 | 1987 | 2028 |
| LA28 | 20 × 10 | 2046[c] | 1973 | 2001.1 | 1938 | 1984.4 | **1937** | 1972.8 | 1972 | 2026.7 | 1971 | 2001.1 | 1971 | 1989.8 |
| LA29 | 20 × 10 | 1857[c] | 1819 | 1886.5 | 1817 | 1851.9 | **1764** | 1825.9 | 1808 | 1873.2 | 1808 | 1862.8 | 1808 | 1850.3 |
| LA30 | 20 × 10 | 2033[c] | 1944 | 1996.3 | **1939** | 1967 | 1939 | 1965.3 | 1974 | 2022.6 | 1953 | 2001.6 | 1953 | 1987.8 |
| LA31 | 30 × 10 | 2866[c] | 2746 | 2790.9 | 2746 | 2784.5 | 2746 | 2780.8 | 2791 | 2851 | **2714** | 2817.3 | 2714 | 2811.6 |
| LA32 | 30 × 10 | 3040[c] | 2982 | 3038.2 | 2982 | 3038.2 | 2982 | 3029.2 | 2962 | 3024.6 | **2928** | 2997.7 | 2928 | 2995.2 |
| LA33 | 30 × 10 | 2803[c] | **2717** | 2797.1 | 2717 | 2783.2 | 2717 | 2770.6 | 2736 | 2774.1 | 2732 | 2765.7 | 2732 | 2765.7 |
| LA34 | 30 × 10 | 2862[c] | 2800 | 2848 | **2769** | 2831.9 | 2769 | 2831.9 | 2812 | 2866.4 | 2812 | 2856.8 | 2812 | 2855.6 |
| LA35 | 30 × 10 | 2871[c] | 2802 | 2861.8 | 2762 | 2850.4 | 2762 | 2849.8 | 2860 | 2901 | 2802 | 2875.7 | **2757** | 2855.2 |
| LA36 | 15 × 15 | 1757[c] | 1708 | 1782.2 | 1697 | 1744.3 | **1683** | 1731.3 | 1701 | 1771.4 | 1700 | 1746.5 | 1700 | 1742.8 |
| LA37 | 15 × 15 | 1871[c] | 1862 | 1903.8 | **1856** | 1885.9 | 1856 | 1879.7 | 1881 | 1915.4 | 1881 | 1912.8 | 1867 | 1907.9 |
| LA38 | 15 × 15 | 1716[c] | **1665** | 1725.2 | 1665 | 1709.1 | 1665 | 1697.4 | 1692 | 1732.2 | 1670 | 1717.3 | 1670 | 1707.6 |
| LA39 | 15 × 15 | 1750[c] | 1734 | 1771.1 | 1728 | 1754.3 | 1728 | 1745.2 | 1725 | 1758.3 | **1720** | 1749.8 | 1720 | 1741.1 |
| LA40 | 15 × 15 | 1742[c] | **1712** | 1760.7 | 1712 | 1746.6 | 1712 | 1743.3 | 1747 | 1779.4 | 1747 | 1768.9 | 1735 | 1760.9 |

a = Mati et al. [Mati and Xie, 2011], b = Pranzo and Pacciarelli [Pranzo and Pacciarelli, 2016], c = Dabah et al. [Dabah *et al.*, 2019]

**Table B.10:** Results for TA Instances with *N5* and JIFR - 1 & 2.

| Instance | Size | 60 sec | | 300 sec | | 600 sec | | 1200 sec | | 1800 sec | | Avg # Iter |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg | (1800 sec) |
| TA01 | 15 × 15 | 1720 | 1769.4 | 1720 | 1761.2 | 1720 | 1761.2 | 1705 | 1720.8 | 1705 | 1720.8 | 132493 |
| TA02 | 15 × 15 | 1679 | 1713.6 | 1678 | 1700 | 1678 | 1700 | 1665 | 1686.6 | 1665 | 1684.8 | 125637 |
| TA03 | 15 × 15 | 1705 | 1750.4 | 1699 | 1715 | 1699 | 1715 | 1655 | 1676.6 | 1655 | 1675 | 128002.2 |
| TA04 | 15 × 15 | 1669 | 1682.6 | 1651 | 1673.4 | 1617 | 1659.6 | 1617 | 1638.8 | 1617 | 1627 | 131116 |
| TA05 | 15 × 15 | 1692 | 1729 | 1679 | 1712.2 | 1679 | 1712.2 | 1679 | 1704 | 1679 | 1696.4 | 124046.8 |
| TA06 | 15 × 15 | 1720 | 1754.4 | 1720 | 1745 | 1707 | 1727.8 | 1707 | 1727.6 | 1691 | 1721.8 | 129908.8 |
| TA07 | 15 × 15 | 1720 | 1780.6 | 1720 | 1768 | 1720 | 1753.6 | 1720 | 1750.8 | 1707 | 1728.4 | 126565 |
| TA08 | 15 × 15 | 1719 | 1756.6 | 1701 | 1723.8 | 1701 | 1723.8 | 1701 | 1716.2 | 1701 | 1716.2 | 130917 |
| TA09 | 15 × 15 | 1763 | 1814.2 | 1763 | 1797 | 1763 | 1797 | 1763 | 1779.8 | 1731 | 1765.8 | 129198.4 |
| TA10 | 15 × 15 | 1705 | 1762.4 | 1705 | 1728.6 | 1705 | 1728.6 | 1687 | 1714.2 | 1687 | 1709 | 128860.4 |
| TA11 | 20 × 15 | 2041 | 2127.6 | 2009 | 2084 | 2009 | 2078.4 | 2009 | 2078.4 | 2009 | 2073.2 | 65770.2 |
| TA12 | 20 × 15 | 2229 | 2276.6 | 2156 | 2228.8 | 2156 | 2228.8 | 2156 | 2214.6 | 2156 | 2214.6 | 61429.6 |
| TA13 | 20 × 15 | 2055 | 2134.8 | 2010 | 2099 | 2010 | 2099 | 2010 | 2094.2 | 2010 | 2083 | 66094.8 |
| TA14 | 20 × 15 | 2110 | 2140.2 | 2070 | 2098.4 | 2070 | 2098.4 | 2070 | 2098.4 | 2070 | 2097.4 | 66627.2 |
| TA15 | 20 × 15 | 2087 | 2152.8 | 2031 | 2106.6 | 2031 | 2106.6 | 2031 | 2106.6 | 2031 | 2087.6 | 67208 |
| TA16 | 20 × 15 | 2173 | 2236 | 2168 | 2225.6 | 2168 | 2225.6 | 2168 | 2205.4 | 2168 | 2205.4 | 62628 |
| TA17 | 20 × 15 | 2264 | 2296 | 2237 | 2269.2 | 2229 | 2261.8 | 2229 | 2261.8 | 2229 | 2261.8 | 63250.2 |
| TA18 | 20 × 15 | 2160 | 2215.8 | 2136 | 2157 | 2136 | 2157 | 2136 | 2144.8 | 2125 | 2142.2 | 65171.6 |
| TA19 | 20 × 15 | 2148 | 2190.6 | 2076 | 2129.4 | 2076 | 2129.4 | 2076 | 2129.4 | 2053 | 2108 | 62692.2 |
| TA20 | 20 × 15 | 2138 | 2238.6 | 2125 | 2183.6 | 2125 | 2167.8 | 2125 | 2152 | 2125 | 2152 | 64146.6 |
| TA21 | 20 × 20 | 2601 | 2637 | 2456 | 2520.4 | 2456 | 2517 | 2439 | 2509.6 | 2439 | 2509.6 | 42527.2 |
| TA22 | 20 × 20 | 2485 | 2536.2 | 2402 | 2441.4 | 2402 | 2437 | 2402 | 2437 | 2402 | 2437 | 39014 |
| TA23 | 20 × 20 | 2397 | 2492.6 | 2358 | 2403 | 2348 | 2396.8 | 2348 | 2396.8 | 2348 | 2396.8 | 38316.6 |
| TA24 | 20 × 20 | 2503 | 2545 | 2462 | 2492 | 2431 | 2484.2 | 2431 | 2484.2 | 2431 | 2484.2 | 40505.8 |
| TA25 | 20 × 20 | 2436 | 2487.8 | 2350 | 2406.8 | 2329 | 2394.4 | 2329 | 2394.4 | 2329 | 2394.4 | 39618.4 |
| TA26 | 20 × 20 | 2564 | 2637.2 | 2467 | 2545.6 | 2467 | 2544.6 | 2467 | 2529.2 | 2467 | 2529.2 | 38380.6 |
| TA27 | 20 × 20 | 2618 | 2667.2 | 2529 | 2582.6 | 2529 | 2577.4 | 2529 | 2577.4 | 2529 | 2577.4 | 41745.6 |
| TA28 | 20 × 20 | 2500 | 2545.4 | 2445 | 2479 | 2434 | 2471.6 | 2434 | 2471.6 | 2434 | 2471.6 | 40721.4 |
| TA29 | 20 × 20 | 2546 | 2615.2 | 2506 | 2548.8 | 2489 | 2537.6 | 2489 | 2534.2 | 2489 | 2534.2 | 40344.2 |
| TA30 | 20 × 20 | 2483 | 2540.8 | 2427 | 2466.8 | 2427 | 2465.8 | 2374 | 2445 | 2374 | 2443 | 42781.6 |
| TA31 | 30 × 15 | 3303 | 3358.8 | 3209 | 3250.4 | 3157 | 3189 | 3123 | 3177.4 | 3123 | 3177.4 | 24551.8 |
| TA32 | 30 × 15 | 3313 | 3395.4 | 3193 | 3257.6 | 3184 | 3249.4 | 3159 | 3237.8 | 3159 | 3237.8 | 24714 |
| TA33 | 30 × 15 | 3446 | 3501.6 | 3266 | 3385.6 | 3266 | 3362.6 | 3234 | 3324.4 | 3234 | 3289 | 23910 |
| TA34 | 30 × 15 | 3391 | 3474.8 | 3335 | 3379 | 3221 | 3285.2 | 3221 | 3274.4 | 3221 | 3274.4 | 23571.4 |
| TA35 | 30 × 15 | 3259 | 3334.2 | 3106 | 3184.6 | 3106 | 3160.6 | 3106 | 3153.8 | 3106 | 3153.8 | 24200.8 |
| TA36 | 30 × 15 | 3301 | 3387.8 | 3210 | 3306.8 | 3169 | 3270.6 | 3146 | 3247.6 | 3146 | 3247.6 | 23884.6 |
| TA37 | 30 × 15 | 3396 | 3478.2 | 3293 | 3378 | 3223 | 3324.8 | 3223 | 3318.4 | 3223 | 3318.4 | 24077.2 |
| TA38 | 30 × 15 | 3241 | 3263.2 | 3094 | 3163.6 | 3071 | 3121.4 | 3071 | 3105.8 | 3071 | 3105.8 | 23838.8 |
| TA39 | 30 × 15 | 3092 | 3159 | 3038 | 3074.4 | 3002 | 3036.2 | 3002 | 3034 | 3002 | 3034 | 26054.8 |
| TA40 | 30 × 15 | 3122 | 3270 | 3098 | 3146.2 | 3094 | 3117.4 | 3026 | 3061.2 | 3026 | 3061.2 | 24870.8 |

**Table B.10:** Results for TA Instances continued

| Instance | Size | 60 sec | | 300 sec | | 600 sec | | 1200 sec | | 1800 sec | | Avg # Iter |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg | (1800 sec) |
| TA41 | 30 × 20 | 3730 | 3890.4 | 3667 | 3714.6 | 3596 | 3638.2 | 3541 | 3602.4 | 3516 | 3593.4 | 13081.8 |
| TA42 | 30 × 20 | 3656 | 3745.6 | 3478 | 3560.6 | 3478 | 3535.8 | 3476 | 3505.2 | 3474 | 3498.8 | 14202.4 |
| TA43 | 30 × 20 | 3549 | 3618.8 | 3362 | 3506 | 3357 | 3460 | 3345 | 3419.6 | 3345 | 3419.6 | 14978.2 |
| TA44 | 30 × 20 | 3752 | 3805 | 3575 | 3612.2 | 3516 | 3593 | 3516 | 3562.8 | 3516 | 3562.8 | 14513.8 |
| TA45 | 30 × 20 | 3682 | 3888.2 | 3519 | 3651 | 3492 | 3578.6 | 3472 | 3547.4 | 3472 | 3547.4 | 14116.6 |
| TA46 | 30 × 20 | 3704 | 3867.8 | 3637 | 3665.8 | 3562 | 3610.2 | 3451 | 3547.8 | 3444 | 3546.4 | 14486.6 |
| TA47 | 30 × 20 | 3723 | 3776 | 3526 | 3591.6 | 3427 | 3531 | 3396 | 3508.4 | 3396 | 3486.4 | 15517.4 |
| TA48 | 30 × 20 | 3651 | 3773.6 | 3487 | 3563.8 | 3445 | 3513.4 | 3428 | 3491.4 | 3427 | 3491.2 | 14053 |
| TA49 | 30 × 20 | 3534 | 3694.2 | 3435 | 3527.6 | 3394 | 3480.8 | 3394 | 3458 | 3394 | 3441.2 | 14766.6 |
| TA50 | 30 × 20 | 3653 | 3834.2 | 3532 | 3656.6 | 3519 | 3617.6 | 3514 | 3562.4 | 3514 | 3560.8 | 13984 |
| TA51 | 50 × 15 | 5531 | 5689.4 | 5239 | 5374.8 | 5216 | 5297.2 | 5175 | 5246.6 | 5079 | 5213.8 | 8273.2 |
| TA52 | 50 × 15 | 5611 | 5703.8 | 5322 | 5382.4 | 5254 | 5298 | 5151 | 5238 | 5148 | 5228.4 | 8236.8 |
| TA53 | 50 × 15 | 5211 | 5515.4 | 5160 | 5280.6 | 5143 | 5242.2 | 5069 | 5132.2 | 5041 | 5113.6 | 8114.6 |
| TA54 | 50 × 15 | 5375 | 5540.4 | 5237 | 5349 | 5136 | 5233 | 5115 | 5188.2 | 5074 | 5157.4 | 8376 |
| TA55 | 50 × 15 | 5299 | 5577.4 | 5081 | 5212.8 | 5042 | 5157.4 | 4986 | 5118.6 | 4972 | 5080.4 | 8218.2 |
| TA56 | 50 × 15 | 5428 | 5666 | 5295 | 5397.6 | 5194 | 5316 | 5170 | 5288.4 | 5113 | 5233.6 | 8182 |
| TA57 | 50 × 15 | 5639 | 5731.6 | 5415 | 5509.6 | 5411 | 5454.8 | 5187 | 5306.8 | 5187 | 5301.4 | 8423.8 |
| TA58 | 50 × 15 | 5698 | 5833 | 5368 | 5557.8 | 5368 | 5529.8 | 5171 | 5418.6 | 5128 | 5397.8 | 7978.8 |
| TA59 | 50 × 15 | 5355 | 5488.4 | 5182 | 5269.8 | 5071 | 5211.6 | 4974 | 5140 | 4974 | 5108.6 | 8301.6 |
| TA60 | 50 × 15 | 5663 | 5757 | 5397 | 5450 | 5326 | 5371.2 | 5145 | 5244 | 5119 | 5198 | 8879 |
| TA61 | 50 × 20 | 6426 | 6542.6 | 6046 | 6181.8 | 5846 | 6013.2 | 5774 | 5942.4 | 5756 | 5918.2 | 5665 |
| TA62 | 50 × 20 | 6703 | 6788.8 | 6298 | 6356.4 | 6042 | 6217.2 | 5948 | 6076.6 | 5890 | 6021.4 | 5526 |
| TA63 | 50 × 20 | 6328 | 6441.4 | 5901 | 5995.2 | 5671 | 5843.4 | 5533 | 5704 | 5530 | 5646 | 5918.2 |
| TA64 | 50 × 20 | 6222 | 6320.6 | 5754 | 5909 | 5716 | 5773.6 | 5532 | 5605 | 5523 | 5576.4 | 5718 |
| TA65 | 50 × 20 | 6450 | 6512 | 5853 | 5954.6 | 5763 | 5827 | 5582 | 5718.2 | 5558 | 5675.2 | 5455.2 |
| TA66 | 50 × 20 | 6383 | 6519.6 | 5949 | 6038.6 | 5879 | 5926.8 | 5761 | 5841.8 | 5723 | 5816.4 | 5517.4 |
| TA67 | 50 × 20 | 6422 | 6567.6 | 5805 | 5943.2 | 5800 | 5894.2 | 5748 | 5798 | 5715 | 5745.4 | 5752 |
| TA68 | 50 × 20 | 6102 | 6356.4 | 5842 | 6018.2 | 5842 | 5978.8 | 5803 | 5855.4 | 5740 | 5804.2 | 5275.4 |
| TA69 | 50 × 20 | 6658 | 6699.6 | 6087 | 6126.2 | 5979 | 6062.2 | 5844 | 5934 | 5810 | 5907 | 5914.2 |
| TA70 | 50 × 20 | 6599 | 6764 | 5988 | 6174.8 | 5982 | 6036.8 | 5799 | 5912.8 | 5799 | 5882.6 | 6062 |
| TA71 | 100 × 20 | 16560 | 17426.6 | 13372 | 13485.4 | 12744 | 12882.6 | 12231 | 12568.4 | 12153 | 12369.4 | 2275.6 |
| TA72 | 100 × 20 | 15445 | 16225.8 | 12901 | 13076.4 | 12182 | 12387.2 | 11688 | 11907.4 | 11444 | 11745.6 | 2437.8 |
| TA73 | 100 × 20 | 16444 | 17370.4 | 13375 | 13533.4 | 12602 | 12792.8 | 11991 | 12269.4 | 11766 | 12078.6 | 2391.6 |
| TA74 | 100 × 20 | 16266 | 16963.6 | 13354 | 13526.4 | 12702 | 12791.6 | 12038 | 12285.2 | 11897 | 12044.8 | 2569.8 |
| TA75 | 100 × 20 | 16189 | 17127.6 | 13288 | 13497.6 | 12451 | 12729.8 | 12002 | 12181.8 | 11476 | 11911.4 | 2287.6 |
| TA76 | 100 × 20 | 15532 | 16578 | 13081 | 13447.8 | 12543 | 12865.8 | 12067 | 12388.2 | 12067 | 12223.8 | 2352 |
| TA77 | 100 × 20 | 16369 | 17674.8 | 13436 | 13640.4 | 12813 | 13037.6 | 12457 | 12543.8 | 12265 | 12412.2 | 2383.4 |
| TA78 | 100 × 20 | 16649 | 17007.8 | 13198 | 13506.4 | 12596 | 12756 | 11793 | 12074.6 | 11697 | 11898.6 | 2329.8 |
| TA79 | 100 × 20 | 15834 | 17145.8 | 13243 | 13439 | 12508 | 12820.6 | 11999 | 12281 | 11870 | 12118.4 | 2236.6 |
| TA80 | 100 × 20 | 15227 | 16186.4 | 12640 | 12943.6 | 12042 | 12310.2 | 11654 | 11869.8 | 11405 | 11729 | 2253.4 |

# C Scheduling for multi-robot routing with blocking and enabling constraints

## C.1 Additional Proofs

**Theorem** 6. Determining whether SSp permits a feasible schedule is NP-Complete.

*Proof. (Sketch)* The deadlock avoidance (DA) problem as just stated was shown to be NP-Complete in Theorem 2 of Araki *et al.* [1977]. The DA problem under those restrictions can be transformed into an instance of SSp, by viewing processes in DA as a set of robot tours. Resource requirements for each vertex in a process of DA can be easily computed. If resource requirements for a pair of vertices belonging to different processes overlap, it can be viewed as a CC between that pair of vertices in the context of SSp. This view is valid since the DA problem is restricted to contain only a single unit of each resource type, and introducing a CC thereby prevents the service duration of the conflicting pair of nodes to overlap in any feasible solution of the SSp instance constructed. The processing times of the nodes in the SSp instance can be set to any arbitrary positive number, without affecting the feasibility of the instance.

It remains to show how one can transform a feasible solution of the DA instance (if one exists) to a feasible solution of the SSp instance constructed, and vice-versa. Since a feasible solution to the DA instance is by definition described as a STP, we can then use the procedure in ($\Leftarrow$) direction of proposition 4 to construct a schedule valid for SSp.

Likewise, by using the procedure in ($\Rightarrow$) direction of proposition 4, we can obtain a STP $P$ from a feasible schedule of the SSp instance. However, the STP $P$ may contain state transitions where a swap may have occurred to overcome a deadlock situation similar to the situation in Figure 5.1d. It can be shown that for swaps to occur, the vertices occurring on the state immediately after the swap (e.g. nodes $V12, V22$ in Figure 5.1d) would at-least have to issue both an allocate and deallocate request. Issuing both requests at a single vertex will violate the “single parameter" restriction of the DA problem instance. So $P$ can be shown to be free of swap moves, and so is a valid STP for the DA instance. □

**Theorem** 7. Given a SSp instance and time bound $B$, determining whether there exists a feasible schedule with makespan at most $B$ for the given SSp instance is strongly NP-Complete.

*Proof.* Given a schedule for the SSp instance, determining whether the schedule is feasible, and it's makespan is at most $B$ can be easily established in polynomial time. So the problem stated in Theorem 7 is in NP. We next show that SMRTD is polynomial time reducible to the problem stated in Theorem 7. Given an instance of the SMRTD problem, we construct an SSp instance as follows. For each task $t_i \in T$ of the machine scheduling problem, we introduce a robot $r_i$ in the SSp instance and define a robot tour $Seq_{r_i} = (o^{r_i}, v_i^1, v_i^2, d^{r_i})$. The processing time values for each node in the tour is set as follows, $p^{r_i}_{o^{r_i}} = r(t_i)$, $p^{r_i}_{v_i^1} = l(t_i)$ and $p^{r}_{v_i^2} = B - d(t_i)$, where

$$B = 1 + \left( \max_{i \in [1,M]} d(t_i) \right) \tag{C.30}$$

and $\boldsymbol{\mathcal{E}}_v = \emptyset, \forall v \in \{o^{r_i}, v_i^1, v_i^2, d^{r_i}\}$. Notice, the processing time defined above implies that in any feasible schedule for the SSp instance, the start time for node $v_i^1$ cannot be earlier than the release time of task $t_i$. Further, if servicing node $v_i^2$ is started later than $d(t_i)$, the corresponding schedule's makespan will exceed $B$, thereby imposing a deadline on the completion time of the node $v_i^1$. Define collision constraints $\mathbf{C}$ for the SSp instance as:

$$\mathbf{C} = \{(v_i^1, r_i, v_j^1, r_j) | i, j \in [1, M], i < j\} \tag{C.31}$$

Notice that CCs in equation (C.31) are only applied to nodes in the set $V^1 = \{v_i^1\}_{i=1}^M$, thereby preventing concurrent execution of any pair of nodes in $V^1$. These CCs emulate the behavior that during servicing of any node in $V^1$, a shared resource (machine) needs to be blocked. It is straightforward to verify that there exists a feasible solution to the SSp instance constructed with makespan at most $B$ iff the SMRTD scheduling problem admits a feasible solution. □

**Proposition** 5 For state $s_{curr}$, if for each scc $c \in SCC$ and each node $u \in U_c$ (refer Equation (5.8)) we have $|K(u)| \leq 1$ (refer Equation (5.7)), then $CF_{s_{curr}}^{Ps}$ has at most $M$ minimal solutions.

*Proof.* Under the conditions mentioned in proposition 5 we first provide a simple procedure for enumerating the minimal models of $CF_{s_{curr}}^{Ps}$ and then argue that the number of such minimal models generated by the procedure does not exceed $M$. Briefly, the enumeration procedure omits the collision related clauses (see Equation (5.10)) in $CF_{s_{curr}}^{Ps}$ and outputs the minimal models feasible for the remaining clauses. The minimal models obtained from the previous step are then checked whether any collision related clause is violated. Our procedure essentially builds a directed graph to capture the dependencies between maximal sccs in $G_{Ps}[B_{s_{curr}}]$ and then retrieves the minimal models from the dependency graph. The description below interlaces the procedure with the reasons for those steps being performed. The procedural steps are highlighted in bold italics.

1. ***We build the dependency graph*** $G_{dep}$ ***as follows. Corresponding to each maximal scc of*** $G_{Ps}[B_{s_{curr}}]$***, we introduce a vertex in*** $G_{dep}$**.** If $e$ is a maximal scc of $G_{Ps}[B_{s_{curr}}]$, then slightly abusing notation we will denote the vertex corresponding to scc $e$ in $G_{dep}$ by $v_e$.

2. Under the condition mentioned in the proposition that $|K(u)| \leq 1$, every clause that was generated in $CF_{s_{curr}}^{Ps}$ from Equations (5.5), (5.9) and (5.11) are either of the form $(y_c \rightarrow y_d)$ or $(y_c \rightarrow 0)$. ***Corresponding to each clause of the form*** $(y_c \rightarrow y_d)$ ***in*** $CF_{s_{curr}}^{Ps}$ ***we introduce an arc*** $(v_d, v_c)$ ***in*** $G_{dep}$, to represent the dependency of scc $c$ on scc $d$.

3. After introducing those arcs in the previous step, we then remove some vertices and arcs from $G_{dep}$ as follows. ***For each clause of the form*** $(y_c \rightarrow 0)$ ***in*** $CF_{s_{curr}}^{Ps}$***, we remove vertex*** $v_c$ ***and all the other vertices that are reachable from*** $v_c$ ***by traversing arcs in*** $G_{dep}$ ***from tail to head.*** While nodes comprising scc $c$ cannot appear in a successor of $s_{curr}$ owing to the clause $(y_c \rightarrow 0)$, sccs reachable from $c$ can also be ignored owing to their dependency on $c$ by transitivity. Consequently we may remove $v_c$ and vertices dependent on $v_c$ from further consideration in the enumeration process.

4. We next describe a procedure to enumerate the minimal models of $CF^{Ps}_{s_{curr}}$ using the maximal sccs of $G_{dep}$. ***Compute the maximal sccs of*** $G_{dep}$**.** Observe that each maximal scc in $G_{dep}$ comprises of maximal sccs of $G_{Ps}[B_{s_{curr}}]$. Further, if $W_1, W_2$ are maximal sccs of $G_{dep}$, such that, there exists a path from a vertex in $W_1$ to a vertex in $W_2$, then vertices in $W_2$ depend on $W_1$. It implies that if a member of $W_2$ is present in a feasible solution to $CF^{Ps}_{s_{curr}}$ then all members of $W_1$ must also necessarily be present in that solution, but not necessary for the other way around. Hence to enumerate minimal models of $CF^{Ps}_{s_{curr}}$, it is sufficient to consider only those maximal sccs of $G_{dep}$ that do not have an incoming arc from any other maximal scc of $G_{dep}$[3], since occurrence of these sccs in a feasible solution is not dependent on the occurrence other sccs.

5. Let us denote the maximal sccs of $G_{dep}$ that do not have an incoming arc from any other maximal sccs of $G_{dep}$ by $SCC_{dep}$. If there exists a pair of maximal sccs of $G_{Ps}[B_{s_{curr}}]$, say $c$ and $d$, such that $CF^{Ps}_{s_{curr}}$ contains a collision related clause $(y_c \rightarrow (1 - y_d))$ while vertices $v_c$ and $v_d$ both belong to the same maximal scc of $G_{dep}$, then the STP to $s_{curr}$ cannot possibly be extended to a complete STP.

6. ***If the condition in the previous step does not occur, then from each member (maximal scc) of*** $SCC_{dep}$ ***we generate a minimal model of*** $CF^{Ps}_{s_{curr}}$**.** If $W$ is a member of $SCC_{dep}$, then a minimal model of $CF^{Ps}_{s_{curr}}$ is derived by setting the $y_.$ variables corresponding to just those maximal sccs of $G_{Ps}[B_{s_{curr}}]$ that are present in $W$ to 1. Since the number of maximal sccs of $G_{dep}$ is bounded by $M$, it follows that the number of minimal models of $CF^{Ps}_{s_{curr}}$ is $\leq M$.

□

## C.2 Example for a case where the number of minimal models for $CF^{Ps}_{s_{curr}}$ in Equation (5.12) is exponential in $M$.

Assume the SSp instance contains no CCs, the number of robots $M$ is even and the consistent positional selection $Ps$ has no arc with it's head to a node in $B_{s_{curr}}$. For notational convenience, let $v^i = s_{curr}[i]$ and let $w^i = next_{v^i}$. The ECs for each $w^i$, $i \in [1, M]$ is defined as follows:

$$\boldsymbol{\mathcal{E}}_{w^i} = \begin{cases} \{v^{M-1}, v^M\} & \text{if } i = 1 \\ \{v^{(i-2)}, v^{(i-3)}\} & \text{if } i \text{ is odd and } > 1 \\ \boldsymbol{\mathcal{E}}_{w^{i-1}} & \text{if } i \text{ is even} \end{cases} \tag{C.32}$$

Under those assumptions, we can safely assume that $CF^{Ps}_{s_{curr}}$ only contains clauses for the enabling constraints (Equation (5.9)) and of course the clause to select at least one node in $B_{s_{curr}}$ to appear in the successor. Also observe that each node in $B_{s_{curr}}$ corresponds to a maximal scc in $G_{Ps}[B_{s_{curr}}]$. So instead of stating which maximal sccs occur in the minimal solution, we will instead mention which nodes of $B_{s_{curr}}$ occur in each minimal solution.

For $M = 8$ case, we illustrate the enablers provided in Equation (C.32) graphically in Figure C.9. For e.g. if $w^1$ is to appear in $s_{next}$ (minimal successor of $s_{curr}$), then from Equation

[3] Any directed graph must contain at least one such maximal scc

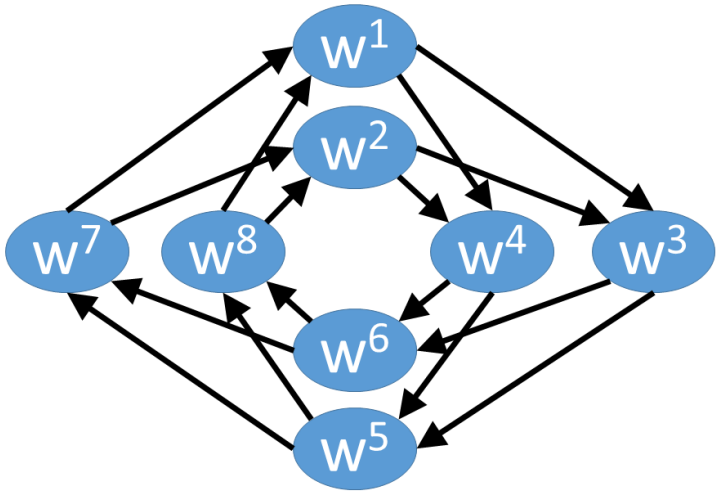



**Figure C.9:** Example for exponential branching.

(C.32) notice that at least one among $w^7, w^8$ must also appear $s_{next}$. We represented this requirement using arcs $(w^8, w^1), (w^7, w^1)$ in Figure C.9. Similarly the enabling requirements for all other nodes are also represented using arcs in Figure C.9.

We next enumerate the minimal models of $CF^{Ps}_{s_{curr}}$. We claim that nodes occurring on any chordless cycle in the graph shown in Figure C.9 corresponds to a minimal model. To see this, observe that the nodes occurring on any cycle of the graph shown in Figure C.9 corresponds to a feasible model for $CF^{Ps}_{s_{curr}}$, as every node in the cycle is enabled by its predecessor on the cycle. If the cycle contained a chord, then the solution is not minimal since we can form a shorter cycle and the nodes belonging to the shorter cycle also correspond to a feasible model of $CF^{Ps}_{s_{curr}}$. We can continue short cutting the cycle until we obtain a chordless cycle. There are at least $2^{\frac{M}{2}}$ chordless cycles (Lavrov [2018]) in the graph shown in Figure C.9 and corresponding to each of those cycles we can obtain a minimal successor.