# MULTI-AGENT ROUTING IN SHARED GUIDEPATH NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Stephen Greyson Daugherty

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology

December 2017

# MULTI-AGENT ROUTING IN SHARED GUIDEPATH NETWORKS

Approved by:

Date Approved: August 23, 2017

Only those who will risk going too far can possibly find out how far one can go.

*T. S. Eliot*

To my parents, my wife, and my parachute rigger.

## ACKNOWLEDGEMENTS

I would like to thank the Georgia Tech Research Institute (GTRI) for its support of the research contained in this thesis as an Internal Research and Development (IRAD) project. I would like to thank Greg Mohler in particular, whose support and advocacy played an enormous role in making this research possible.

I would further like to thank my academic advisor, Spyros Reveliotis, for his advice, his insights, his patience, and his constructive criticism. I consider myself extremely fortunate to have worked with him.

Last but not least, I would like to thank the superlative academic instructors whom I have had the privilege to learn from within Georgia Tech's H. Milton Stewart School of Industrial and Systems Engineering, including (but not limited to): David Goldberg, Eva Lee, Craig Tovey, Alan Erera, Nicoleta Serban, and Spyros Reveliotis.

# TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1
# INTRODUCTION AND BACKGROUND

## 1.1 Problem Description and Motivation

The research program presented in this document addresses the problem of routing a number of traveling agents along the edges of a connected graph, which is referred to as a "guidepath network." Every agent has a designated destination to which it is being routed. The objective of this research is to find ways to minimize the time required to route all agents in the system from their "current" positions to their respective destinations, simultaneously. Since the considered guidepath network is shared by the entire set of agents, the routes these agents follow will be subject to certain congestion, or "coupling", constraints that prevent any agent's route from causing conflict with any other agent's route. Thus, this research is aimed at providing efficient, conflict-free, multi-agent routes between arbitrary origins and destinations within a highly restricted transportation environment.

One "real-world" application that motivates this research is that of flexibly automated, unit-load, zone-controlled material handling (MH) systems implemented in various production and distribution settings [24]. In this case, the vehicles (represented as "agents" in the problem formulation below) are restricted to a well-defined network (i.e., a "guidepath network") whose structure may follow directly from the physical layout of the MH system, as in the case of crane or gantry systems used in ports or in heavy-industrial manufacturing plants, or of the monorail systems used in modern semiconductor fabs. Alternatively, the network may be virtual, as in the case of automated guided vehicles (AGVs); in this case, vehicles are restricted to certain sections of a production or distribution layout to avoid collisions with workers and other machines.

The guidepath network traversed by a set of AGVs (or other MH systems) can be divided into “zones” designed to prevent collisions by allowing only one vehicle to reside in a zone at any given moment. These zones induce a natural discretization of the vehicle trips wherein a route for any given vehicle can be described by a sequence of adjacent zones that define the agent’s discrete location at each time step.

Another problem instance that motivates this work (and that maps neatly into the discretization just described) is that of coordinating the movement of quantum bits (known as “qubits”) inside the central processing unit of a quantum computer [39]. Much like AGVs, qubits must travel between storage locations and “gate” locations, where they can interact with one another or be subjected to phase-change operations. And much like AGVs, qubits move along a specific guidepath network. The exact physical implementation may vary (from ion-trap to quantum-dot configurations), but the underlying routing problem is essentially independent of the type of quantum computer.

The above contexts share certain operational characteristics–namely, restrictions against agent co-residency at zones on the guidepath network that do not serve either as source or destination locations, along with a “no-swap” rule that prevents agents from switching zones between one time step and the next. This last restriction ensures, among other things, that the model will not expect two AGVs traveling in opposite directions down a narrow aisle to switch places instantaneously.

There are also some differences between the AGV context and the quantum-computing one. For instance, it is often assumed that AGVs do not reverse direction within an aisle, but this behavior is acceptable for qubits, and it may, indeed, be necessary or helpful in order to solve the problem addressed in this work. Another consideration is that AGV systems are often assumed to have no destination locations that can also be used as transit locations by other vehicles. In the quantum-computing context, however, this may be very common. This possibility offers a scheduler extra freedom when routing agents to their respective destinations, but it also implies a number of difficult constraints; given that some zones of the guidepath network that are used for transit can also serve as destinations for various agents, it is imperative that no set of agents residing in their destination locations form a cut of the guidepath network that separates an agent still in transit from its own destination.

The bottom line for both applications is that agents, whether they be qubits or AGVs, must travel efficiently between various origin and destination locations while avoiding certain detrimental or forbidden behaviors. In particular, multiple agents cannot occupy the same zone at the same time during transit, and two agents cannot swap from neighboring zones in the guidepath network. The system must also prevent the formation of "deadlocks"–i.e., states where multiple agents block each other's advancement within the guidepath network–that might arise due to the previous two restrictions. The model (and the algorithms) developed for this research are designed with sufficient generality to cover both types of problem instances and are even adaptable to other contexts (e.g., flexible job-shop scheduling) that have seemingly different system behaviors or constraints than the ones discussed here.

## 1.2 Literature Review

### 1.2.1 AGV-Related Literature and Practice

Various researchers have addressed the deployment of automated, zone-controlled, guidepath-based material handling systems (MHSs), starting with [36], which provided a theoretical framework for the operations research/industrial engineering (OR/IE) community to model these MHSs. However, the analysis contained in [36] focused primarily on the design of efficient physical layouts of the guidepath network rather than on the simultaneous routing of the vehicles contained within the guidepath network. Other sources (e.g., the works discussed in [20, 44, 56]) define the dynamics of AGV environments in greater detail and extend the relevant abstractions beyond the AGV context. However, all of these sources acknowledge the need for new models and control algorithms aimed at reducing congestion delays and avoiding deadlocks while maintaining computational tractability.

Existing industrial practice largely sidesteps the difficulty of making these hard control decisions by using restrictive configurations–e.g., “tandem” layouts that decompose the system traffic into a series of interconnected loops with buffers at the interfaces [5]. But such restrictions come at a price–namely, (i) needlessly lengthy trips between locations that are close to each other, (ii) a rate of movement that is determined by the slowest vehicles on any given loop, and (iii) the need for “double-handling” for transfers between locations on different loops.

These inefficiencies motivate the study of concurrent, multi-agent routing policies in zone-controlled guidepath networks. As part of this effort, a number of researchers have studied in depth the problem of eliminating deadlocks from the considered traffic systems, along with a similar problem known as “livelocks”, wherein a set of moving agents prevents one or more other agents from reaching their destinations. These two types of problems are united by one imperative: The system state (which can sufficiently be described by the location and direction of all agents on the guidepath network at a given moment in time, as well as their remaining visitation requirements) must always be “safe”; i.e., there must exist a deadlock- and livelock-free schedule that can return all agents to a “depot” or “charging station” (i.e., a single location on the guidepath with the capacity to retain all agents in the system and to dispatch them in arbitrary order) after they have successfully visited their destinations.

Reveliotis ([47]) describes a computationally efficient method for enforcing liveness within AGV systems. Reveliotis and Roszkowska ([46]) further determined that maximally permissive deadlock avoidance (i.e., deadlock avoidance that guarantees that a system state will be classified as safe if and only if at least one multi-agent schedule exists to route all agents back to a charging station) is an NP-hard problem. However, sub-optimal solution methods for ensuring safety within various resource allocation systems do exist. These include the application of Banker’s algorithm (formulated initially in [13] and applied to flexible manufacturing systems in [31, 15]), and of methods that rely on finite-state automata [38, 57, 49] and Petri nets [14, 34, 42]. These methods and their applications to job shops and other resource allocation systems are surveyed in [33, 66, 48, 45]. In particular, the methods of deadlock avoidance for broader resource allocation systems have been adapted to the problem of deadlock avoidance in guidepath-based AGV systems in [47, 61, 17, 46]. Roszkowska and Reveliotis ([50]) extended these results to closed AGV systems, and [23] applied them specifically to rail networks.

Deadlock-avoidance algorithms can be used to guarantee that any prescribed schedule cannot cause deadlocks. They do not, however, address the problem of optimizing the performance of multi-agent schedules with respect to some time-based objective. One perspective that can be used to approach this problem, and is adopted in the rest of this thesis, is that of job-shop scheduling. When the multi-agent routing problem is mapped into the job-shop scheduling paradigm, machines are replaced by "zones" of the guidepath network, the "jobs" performed are transportation jobs between different locations, and the "makespan", a term commonly used to describe the length of time required for all jobs in the system to be completed, represents the length of time required for every agent in the system to reach its next destination. The makespan will be used as the objective function for the research described in the following chapters.[1]

The literature on job-shop scheduling offers some distinct methods to approach this problem. These include "branch-and-bound" algorithms, which would find optimal solutions but will not necessarily run in polynomial time [41, 6, 37]. They also include disjunctive programming [43, 37] and various efficient heuristics such as the "shifting-bottleneck" heuristic [43, 40, 1]; but these algorithms cannot easily accommodate the extraordinary flexibility that exists in most guidepath networks.

There is also some prior research that does attack traffic coordination problems similar to those considered in this work more explicitly. Notably, [30] employs column generation

[1]Though other objective functions can be considered (e.g., minimization of average routing time across all agents), this research focuses on minimizing the time required to complete all immediate transportation jobs, for two reasons: (i) minimizing the time to complete all outstanding transportation jobs (which is referred to in this thesis as the "makespan") ensures that no single job will be unduly postponed in order to facilitate the completion of other transportation tasks; and (ii) in the broader literature of combinatorial scheduling theory, minimization of the makespan is a surrogate objective for maximizing the system throughput.

techniques to address a traffic scheduling problem similar to the problem addressed in this work, and [12] extends the work of [30] by using column generation as part of a branch-and-cut scheme. However, these techniques suffer from questionable scalability, and the more recent of these papers ([12]) applies its methodology to a maximum of four vehicles.

### 1.2.2 Literature on Multi-Robot Path Planning (MPP)

Further literature relating to the problem that is addressed in this thesis exists among the artificial intelligence (AI) and robotics communities concerning "multi-robot path planning" (MPP). A precursor to this class of problems exists in the form of a $4 \times 4$ square grid that constrains the movement of 15 non-overlapping, labeled pieces that must reach a particular location from an arbitrary starting configuration. In this problem, a "move" is feasible if and only if it moves a single piece into the empty space of the puzzle. Wilson ([58]) considered this problem, as well as the broader class of problems defined by a 2-connected graph, $G$, with $n$ vertices and $n - 1$ pieces (which the author refers to as "pebbles").

In [58], the author proves that, whenever $G$ is not bipartite, there always exists a finite sequence of moves capable of transforming an arbitrary initial solution to any target state, and that when $G$ *is* bipartite, then the possible configurations of the graph can be grouped into two equivalence classes.

Kornhauser *et al.* ([29]) extended the results of [58] not only by considering the more general case where the number of pebbles, $p$, is less than or equal to $n - 1$, but also by exploring the conditions determining reachability between various configurations on 1-connected and 2-connected graphs, and providing an $O(n^3)$ algorithm for determining this reachability and finding a feasible "move" sequence whenever one exists.

In subsequent years, the AI and robotics communities explored the aforementioned problem, as well as certain variations upon it, under the aegis of "multi-robot path planning" (MPP). These variations include versions of the problem with up to $n$ pebbles, where pebbles (or robots) are assumed to have the ability to move synchronously, in rotating loops, whenever it is necessary or useful to do so [64], a property that has enabled researchers to design new algorithms for testing problem feasibility [65]. On the other hand, [63] establishes that the problem of finding an "optimal" sequence of moves (where optimality is defined based on various objectives, such as minimizing total time for all pebbles, or minimizing maximum completion time) is NP-hard, even when these synchronized loop rotations are permitted.

Efforts to optimize multi-robot path plans can be classified broadly into "coupled" and "decoupled" approaches. Decoupled methods are marked by their decomposition of the problem into a series of single-robot subproblems [51]. The corresponding optimization routines typically consider each robot in sequence, using the "current" path plans of other robots to place constraints on the path plan for the robot under consideration. While these algorithms can be effective at mitigating complexity, they often struggle to provide feasible solutions in cases of even moderate congestion [53, 51, 54]. These algorithms resort to methods such as perturbation or iteration in an effort to mitigate these difficulties, but there is no well-defined logic to guide this additional exploration.

Coupled approaches consider MPPs more holistically by modeling the traffic dynamics as a finite-state automaton (FSA), where the system state is defined based on the location of the traveling agents, and where the constraints of the considered problem are used to define the transitions allowable between various system states. The resulting optimization problems can then be expressed as mixed-integer programs (MIPs) [64], as dynamic programming (DP) problems [54], or as satisfiability (SAT) problems [55]. Though these methods have the virtue of returning optimal results consistently, they are not scalable, as the size of the FSA state space grows explosively with the number of robots.

This research considers a class of problems that is, both, distinct and, in some sense, more general than what is currently described in the MPP literature. For example, perfectly synchronous movement is not assumed to be possible, as it is in [64] and [65]. In addition, the research in this thesis is developed with consideration for the possibility that robots (or, more generally, "agents") may have constraints placed upon their movement based not only on their current location, but on their direction of travel (which is easily generalizable to other characteristics of the agent state).

In order to illustrate the difficulty of addressing the considered class of problems using methods in the existing literature, we can consider the example portrayed in Figure 1.1, where three agents must start in the lower-left corner and finish in the upper-right, *after reversing their order.* Tractable, "decoupled" approaches are described in [28] and [21], where agents are routed iteratively with time-window constraints to find solutions that are efficient and often deadlock-free. In these papers, the authors route each agent based on a shortest-path problem with time windows (SPPTW), where the time windows prevent conflicts with the agents routed earlier. Though this method is clever and elegant, Figure 1.1 illustrates its shortcomings. If one were to attempt to route these three agents using the SPPTW approach for each agent, then one would find that Agent 3 *must* be the first to move. After that, *no conflict-free path will exist for either Agent 1 or Agent 2 to reach its destination.* Hence, approaching this problem as a series of SPPTWs would return no solution.

The ability of such a small problem to defeat such algorithms hints at a very difficult set of implied constraints. Namely, as mentioned in Section 1.1, it is important that no subset of the agents that have come to rest at their respective destinations forms a cut blocking any agent still in transit from its own destination. One of the key contributions of the research described in this thesis is the ability to overcome such complications.

Moreover, though the "coupled" methods described earlier in this section would be capable of finding an optimal solution to this problem, as we show in Chapter 5, these approaches would not be scalable to problem instances with larger guidepath graphs and larger numbers of agents. The presented research seeks to provide an optimization methodology that is scalable to larger problems, yet robust to "difficult" problems such as the one shown in Figure 1.1.

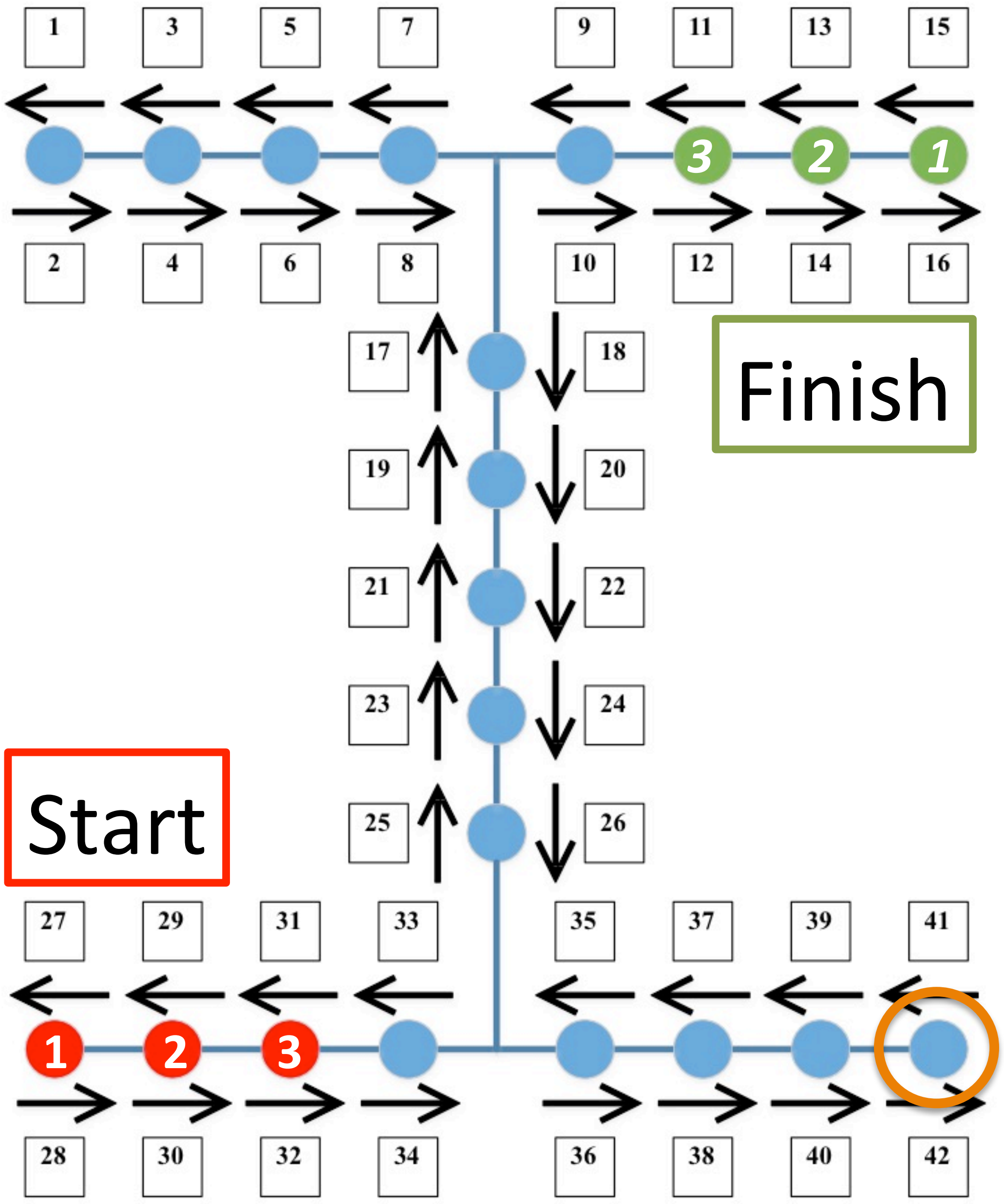

Figure 1.1: This figure represents a problem where agents 1, 2, and 3 must travel from the lower-left corner to the upper-right, after reversing their order. A depot location is circled in the lower-right corner, where agents may be pooled and re-dispatched, but doing so would not be a strong solution to this problem. The arrows are an abstraction that is useful for contexts where agents cannot necessarily reverse their direction of travel; this and other abstracting representations are discussed in more detail in the next chapter.

## 1.3 Goals and Outline of the Proposed Research Program

Motivated by the above observations, the presented research program addresses the problem of finding conflict-free, deadlock-free, and livelock-free multi-agent routing schedules for the aforementioned applications, with minimized makespans. The research presented in this thesis is designed for incorporation into an online, model predictive control (MPC) approach to the problem of scheduling and completing various transportation tasks. In particular, the presented research assumes a "rolling-horizon" decomposition scheme where every agent's current location and next destination are known, and this thesis details methods for finding lower bounds, upper bounds, and efficient feasible solutions to the subproblems that seek to route the traveling agents to their next destinations.

Hence, the primary goal of this research is to design and demonstrate an algorithm that minimizes the makespan for an arbitrary set of agents traveling from their original locations to some specified destination edges on an arbitrary guidepath network. The algorithm will be sufficiently general to address both AGV and qubit routing problems, including problems such as the one represented in Figure 1.1. However, for the sake of specificity, test results will be based on simulated instances of the qubit routing problem (which, in general, tends to be more difficult than AGV routing on account of the fact that all destination edges can also be used as transit locations for agents that are not destined for that edge).

The contributions of this thesis can be summarized as follows:

- The thesis abstracts the considered multi-agent routing problem through a set of modeling assumptions and formulates it analytically as an MIP [60].
- A Lagrangian relaxation (LR) approach is developed to obtain lower bounds for optimal makespans of instances of the considered problem, as well as a set of Lagrange multipliers that may be useful for heuristic optimization algorithms.
- Two different methods for optimizing the Lagrangian dual function are provided:

  - An iterative dual-ascent method that has been carefully adapted for problems of the given type [8, 9].
  - A method to solve the considered dual problem more robustly (finding both a lower bound on the optimal makespan and a set of Lagrange multipliers for the "difficult" constraints) by reducing it to a single linear program (LP).

- The thesis also presents a heuristic, "local-search" type of algorithm for the considered problem. In this context, first we detail an efficient algorithm for constructing an initial solution for any problem instance, which further establishes the *completeness* of our methods. Given an initial feasible solution (i.e., a schedule for all agents that routes each agent from its origin to its destination without any conflicts among different agents) our method then uses dynamic programming (DP) in a novel way to eliminate conflicts between agents as the makespan is iteratively reduced [10, 11]. This technique can be considered as a "decoupled" method, in the terminology of Section 1.2.2, with the following important characteristics:
  - Given *any* initial feasible solution, this method iteratively reduces the makespan until its termination, ensuring that some feasible solution (and usually an optimal or near-optimal solution) is returned.
  - The algorithm reroutes agents in a maximally flexible way, considering all possible paths between a given origin and a given destination that fit within the given time horizon.
  - The algorithm is sufficiently versatile to accommodate diverse operational constraints, such as those that might prohibit agents from reversing their direction of travel.
  - The algorithm is scalable with respect to the size of the guidepath graph *and* the number of agents.

- Finally, the thesis presents experimental results that demonstrate and assess the efficiency of the heuristic approach for the considered traffic problem, as well as results on the algorithms for the aforementioned dual problem.

The rest of this document is organized as follows: Chapter 2 provides the MIP that formally describes the considered class of problems. Chapter 3 details the Lagrangian approach and solution algorithms for the corresponding dual problem. Chapter 4 presents the heuristic algorithm for computing optimized agent routes. Chapter 5 provides experimental results from the implementation of the methodologies given in Chapters 3 and 4. Lastly, Chapter 6 summarizes the results and contributions of this thesis, and recommends a few directions for future research.

# CHAPTER 2

# PROBLEM FORMULATION

## 2.1 Overview of the Main Modeling Representations and Assumptions

In this section we describe abstractions of the guidepath network, and of the agents and their routes on this graph. We then proceed to define the routing problem that the following research addresses, and to formulate it as a mixed integer program (MIP). As stated in Chapter 1, the presented abstractions are sufficiently general to be applicable to a range of different contexts, including the problems of routing multiple AGVs simultaneously and of routing qubits inside a quantum computer.

To begin, the guidepath network can be represented as a graph $G = (V, E \cup \{h\})$, where $V$ represents a set of vertices connected by a set of edges $E$, and $h$ represents a "storage" or "home" location where agents can be held (effectively removed from the system) when not in use.

For the present research, each traveling agent $a \in \mathcal{A}$ is assumed to traverse an edge of the guidepath network in uniform lengths of time. These lengths of time define a natural *"time unit"* for the model and make it possible to discretize the motion dynamics of the agents within the considered system. A "route" between two locations, then, is a sequence of edge indices describing where an agent will be located at any given time, and a "schedule" is a collection of routes describing where all agents will be located between some start time and the end of a given time horizon. For any given edge $e$, transitions must be selected from the set $\{e^\bullet \cup e\}$, where $e^\bullet \in E$ represents the set of edges that an agent can transition onto directly from $e$. $e^\bullet$ is identical for all agents and is assumed to be a static property of $G$, $\forall e \in E$.

Some variability between problem instances must also be considered. In particular,

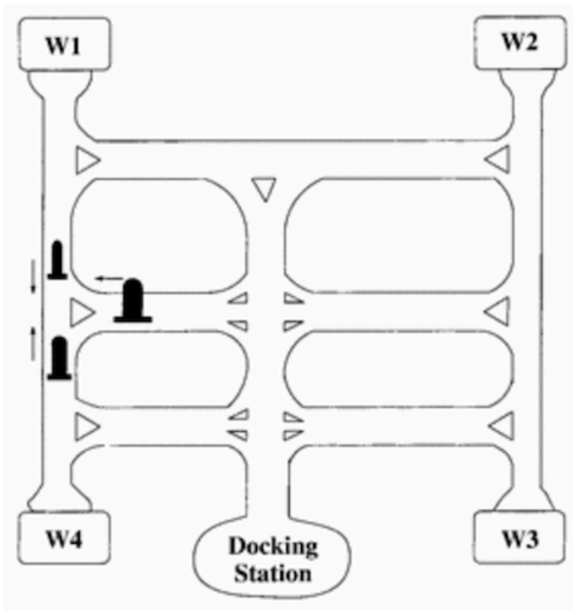

Figure 2.1: Depiction of deadlocked AGVs at an intersection. Any multi-agent scheduling algorithm must prevent such situations from occurring.

whether or not an agent is permitted to reverse direction on an edge varies from one case to another; notably, the abstraction of qubits inside a quantum computer allows this behavior, whereas many AGV problems do not. However, both types of instances can be described using a directed graph. That is, if each edge connecting vertices $v_i$ and $v_j$ is replaced by two directed edges–$(v_i, v_j)$ and $(v_j, v_i)$–then we have a representational framework that is sufficiently general for both cases. For any given edge $(v_i, v_j)$, the edge $(v_j, v_i)$ will be referred to as the "reverse" or "complementary" edge and will be designated as $\bar{e}$.

The reason that the above abstraction is sufficient to model cases where an agent is or is not allowed to reverse direction, is the flexibility with which $e^\bullet$ can be defined; if reversibility is permitted, then one need only enforce that $e^\bullet = \bar{e}^\bullet$, $\forall e \in E$. This remark also exemplifies the flexibility of the presented modeling framework.

Finally, in order to prevent agents from switching places while on neighboring edges, and to avoid the unrealistic requirement that agents move with perfect synchronicity from one location to another–a restriction that is necessary in order to prevent collisions between agents–an agent $a$ is not permitted to transition from some edge $e$ at time $t-1$ to an edge $e' \in e^{\bullet}$ at time $t$ unless edge $e'$ is unoccupied by other agents at both time $t$ and at time $t-1$. In the case that agents cannot reverse their motion within their current edges, this set of constraints can lead to deadlocks such as the one shown in Figure 2.1; therefore, in that case, it is necessary that any effort to define feasible schedules must ensure that they are also deadlock-free [47].

## 2.2 Mathematical Formulation

A "trip" for an agent $a$ can be conceptualized as a sequence of edges $\Sigma_a = \langle e \in E \rangle$ that an agent must visit in the given order before returning to the home edge $h$ and effectively being removed from the system until the next trip. As mentioned in Chapter 1, however, the research (and therefore the MIP formulation) presented here restricts its attention to the problem of routing each agent $a$ from its given starting point, $s_a$, to its next destination, $d_a$. In a larger context, this is equivalent to considering the question of routing every agent to its next destination in $\Sigma_a$ as efficiently as possible. This problem can be formulated as an MIP [60] as follows:

### 2.2.1 Notation

- $V = \{v_1, v_2, ..., v_m\}$: Guidepath-graph vertices
- $E = \{e_1, e_2, ..., e_n\}$, with $e_l = (v_i, v_j)\ \forall l \in \{1, ..., n\}$: Guidepath-graph edges (or "zones")[1]
- $\mathbb{T}\ (n \times n)$: A binary matrix expressing the agent transitional dynamics on the guidepath graph; $\mathbb{T}_{i,j} = 1$ iff a direct transition from $e_i$ to $e_j$ is allowed. We also set $\mathbb{T}_{i,i} = 1\ \forall i$ s.t. $e_i \in E$

[1]To avoid straightforward but notationally awkward conditions, in the subsequent discussion it is assumed that no traveling agent is currently located at, or is destined to, the "home" edge $h$.

- $\bar{e} = (v_j, v_i)$: Complementary edge of edge $e = (v_i, v_j)$
- ${}^{\bullet}e_l = \{e_q \in E : \mathbb{T}_{q,l} = 1 \wedge q \neq l\}$: The set of input edges for $e_l$, $\forall e_l \in E$
- $e_l^{\bullet} = \{e_q \in E : \mathbb{T}_{l,q} = 1 \wedge q \neq l\}$: The set of output edges for $e_l$, $\forall e_l \in E$
- $\mathcal{A} = \{a_1, a_2, ..., a_K\}$: The set of traveling agents
- $d_a$: Destination edge for agent $a \in \mathcal{A}$
- $s_a$: Starting edge for agent $a \in \mathcal{A}$; this edge also specifies initial orientation for the agent motion
- $T$: An upper bound on the required transport time, across all agents (and therefore an upper bound on the optimal value of the objective function)
- $t \in \mathcal{T} = \{0, 1, ..., T\}$: Time index

### 2.2.2 Decision Variables

- $\forall a \in \mathcal{A}$, $\forall e \in E$, $\forall t \in \mathcal{T}$, $x_{a,e,t} \in \{0, 1\}$ indicates whether or not a given traffic schedule places agent $a$ on the directed edge $e$ at timestep $t$; these variables constitute the primary decision variables of the considered traffic coordination problem. For notational convenience, $\boldsymbol{x}$ will represent the vector that collects all the variables $x_{a,e,t}$ in the following sections.
- $w$: An auxiliary variable that represents the makespan (i.e., the total time to completion, as described in Chapter 1) of a given traffic schedule.

### 2.2.3 MIP

Given the notation in Sections 2.2.1 and 2.2.2, the following MIP defines the problem at hand:

$$\min w \tag{2.1}$$

s.t.

$$\forall a \in \mathcal{A},\ \forall t \in \mathcal{T},\ \sum_{e \in E} x_{a,e,t} = 1 \tag{2.2}$$

$$\forall a \in \mathcal{A},\ \forall e \in E\ \ x_{a,e,0} = I_{\{e=s_a\}} \tag{2.3}$$

$$\forall a \in \mathcal{A}, \quad x_{a,d_a,T} = 1 \tag{2.4}$$

$$\forall a \in \mathcal{A},\ \forall t \in \mathcal{T} \setminus \{T\}, \quad x_{a,d_a,t+1} \geq x_{a,d_a,t} \tag{2.5}$$

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{T\},\ \ x_{a,e,t} \leq x_{a,e,t+1} + \sum_{e' \in e^\bullet} x_{a,e',t+1} \tag{2.6}$$

$$\forall e = (v_i, v_j) \in E \text{ s.t. } i < j,\ \forall t \in \mathcal{T} \setminus \{0, T\}, \quad \sum_{a \in \mathcal{A}} (x_{a,e,t} + x_{a,\bar{e},t}) \leq 1 \tag{2.7}$$

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{0\},\ \ x_{a,e,t} + \sum_{a' \in \mathcal{A}: a' \neq a} (x_{a',e,t-1} + x_{a',\bar{e},t-1}) \leq 1 \tag{2.8}$$

$$\forall a \in \mathcal{A}, \quad w\ \geq\ \sum_{t=0}^{T} (1 - x_{a,d_a,t})\ =\ T + 1 - \sum_{t=0}^{T} x_{a,d_a,t} \tag{2.9}$$

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T}, \quad x_{a,e,t} \in \{0,1\} \tag{2.10}$$

Equation (2.2) ensures that each agent will occupy exactly one location on the guidepath at any given time. Equation (2.3) dictates that each agent be located at a prescribed starting point, $s_a$, at time $t = 0$. Equation (2.4) enforces the constraint that every agent must reach its destination, $d_a$, by the upper temporal bound, $T$, and Equation (2.5) requires each agent to remain at its destination edge upon reaching it. Equation (2.6) ensures that, for each time $t$, an agent located at edge $e$ must either remain at edge $e$ or transition to an edge in $e^{\bullet}$.

Together, Equations (2.2)–(2.6) form what will be referred to as the "single-agent constraints". Each of these constraint sets contains a distinct subset of constraints for each agent, and these subsets are disjoint. As will be shown in the following chapters, these constraints can be used to form subproblems that can easily be optimized in polynomial time for various objective functions. Moreover, the constraints describe separate problems for each agent, since each constraint only contains variables pertaining to a single agent. This property will prove useful in Chapters 3 and 4.

Moving on to the rest of the MIP, Equation (2.7) ensures that no two agents occupy the same location at the same time. Equation (2.8) requires that an agent located on edge $e$ at time $t-1$ cannot move onto edge $e'$ at time $t$ unless that edge is unoccupied by any other agent at time $t-1$. This prevents agents from swapping places or allowing one agent to move onto an edge at the same instant that another is moving off of it in a manner that is either physically infeasible or unsafe. Equation (2.9) helps to define the auxiliary variable $w$ that represents the makespan of the problem. Since (a) $w$ is minimized, (b) it is not constrained beyond Equation (2.9), and (c) $w$ is greater than or equal to the right-hand side of Equation (2.9), the optimal $w$ will be tight for at least one of the constraints in Equation (2.9).[2]

[2] We further notice, for completeness, that in order to capture potential "rendezvous" effects among the traveling agents in the MIP formulation of Equations (2.1)–(2.10), we shall need to (i) define the "rendezvous" agent sets $\mathcal{A}(e) = \{a \in \mathcal{A} : d_a = e\}$, $\forall e \in E$, and then (ii) replace Constraints (2.7) and (2.8) in the original

Equations (2.7)–(2.9), when taken together, will be referred to as "multi-agent constraints", "coupling constraints", or "congestion constraints". These descriptors refer to the fact that each of these constraints allows the route chosen for one agent to constrain the solution set available to other agents; hence, these constraints serve as a mathematical formalization of traffic congestion within the guidepath network. These constraints also "couple" the single-agent routing problems to one another, and make the resulting MIP much more difficult to solve. The above partitioning of the constraints of our MIP formulation into the groups of Equations (2.2)–(2.6) and (2.7)–(2.9) will prove helpful in Chapters 3 and 4.

The above MIP formulation is not unique, in the sense that it is not the only way to formulate the considered problem. The MIP presented here is based on a natural representation of the traffic state (i.e., the location of each agent at each time step) and is designed to make efficient use of the "resource allocation system" (RAS) perspective. A different formulation, that is based on a reconceptualization of the single-agent constraints as part of a "min-cost multi-commodity flow" problem, and where the coupling constraints provide additional "side constraints" to the resulting problem, is presented in Appendix A.

Regardless of the particular formulation, it is possible, in general, that the problem described by Equations (2.1)–(2.10) is infeasible. Discerning between feasible and infeasible problem instances is a hard problem in its own right. Infeasibility may be the result either of a combination of the underlying guidepath graph structure and the positioning of the various sources $s_a$ and destinations $d_a$, or of a selection of $T$ that does not allow sufficient

formulation with the following three constraints:

$$\forall e = (v_i, v_j) \in E \text{ s.t. } i < j,\ \forall t \in \mathcal{T}\backslash\{0, T\},\ \forall a \in \mathcal{A}(e), \quad \sum_{a' \in \mathcal{A}\backslash\mathcal{A}(e)} (x_{a',e,t} + x_{a',\bar{e},t}) + x_{a,e,t} + x_{a,\bar{e},t} \leq 1 \tag{2.11}$$

$$\forall a \in \mathcal{A},\ \forall e \in E \setminus \{d_a\},\ \forall t \in \mathcal{T} \setminus \{0\},\ \ x_{a,e,t} + \sum_{a' \in \mathcal{A}: a' \neq a} (x_{a',e,t-1} + x_{a',\bar{e},t-1}) \leq 1 \tag{2.12}$$

$$\forall a \in \mathcal{A} \text{ s.t. } s_a \neq d_a,\ \forall t \in \mathcal{T} \setminus \{0\},\ \ x_{a,d_a,t} + \sum_{a' \in \mathcal{A}\backslash\mathcal{A}(d_a)} (x_{a',e,t-1} + x_{a',\bar{e},t-1}) \leq 1 \tag{2.13}$$

time for the agents to reach their respective destinations.[3] The first of these problems can be approached as a hard "reachability" problem using a formal modeling framework provided by qualitative discrete-event systems (DES) theory, such as Petri nets or finite-state automata [7]. In the broader MPC context, wherein an online scheduler must solve repeated instances of this finite-horizon problem, it might be necessary for the underlying planning system to constrain proactively the selection of the intermediate target states, as well as the entire set of states generated by the traveling agents, in order to ensure the feasibility of the formulated problem instances.

In the rest of this document, as part of the effort to address the problem of finding efficient, multi-agent schedules on a shared guidepath network that was formulated in this chapter, Chapter 3 will define the Lagrangian relaxation of the MIP defined above, and then it will proceed to discuss two methods for optimizing the corresponding dual problem, yielding, thus, (i) a tight lower bound on the optimal makespan of the MIP described by Equations (2.1)–(2.10), and (ii) a set of Lagrange multipliers that can be used to assign a "cost" for having agent $a$ in location $e$ at time $t$, $\forall a$, $\forall e$, $\forall t$, based on the coupling constraints.

Chapter 4 will then define a heuristic scheduling algorithm for the problem of MIP (2.1)–(2.10). This algorithm begins by finding an initial feasible solution with some makespan $T$. The routes for each agent in this schedule are then iteratively revised, using a novel application of dynamic programming (DP), to obtain improved schedules with reduced makespans. These iterations amount to a "local-neighborhood" search for conflict-free solutions within an improved makespan; the end result is a feasible, optimized solution.

[3] It should be noted that instances of the qubit routing problem are guaranteed to have feasible solutions for any feasible initial conditions, given sufficient time. This follows from the universal ability of agents to reverse direction in any location; it implies that an initial feasible solution can be found by sending all agents to the home location and then dispatching them in a certain order to their respective destinations. This is shown and discussed in depth in Chapter 4.

# CHAPTER 3

# A LAGRANGIAN-DUALITY APPROACH FOR THE CONSIDERED MIP

The operations research literature includes substantial precedent for applying Lagrange relaxation (LR) methods to integer programs in general [22, 19] and, more specifically, to resource-constrained scheduling problems [18]. In [52, 27], the authors have adapted Lagrangian approaches to the flexible job-shop scheduling problem; these applications include heuristic methods for generating feasible solutions based on the results of (augmented) Lagrangian relaxations, as well as methods for obtaining lower bounds for the optimal objective value of the considered problem. Next, we introduce an LR for the MIP formulation of Section 2.2, which is similar, in spirit, to the LR employed in [52] for the flexible job-shop scheduling problem.

## 3.1 Formulation of the Lagrangian Relaxation and the Dual Problem

The last part of the previous chapter describes ways that the constraints of the considered problem can be divided into two distinct sets–the single-agent constraints (represented by Equations (2.2)–(2.6)) and the coupling constraints (represented by Equations (2.7)–(2.9)). If the vectors $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$ are composed of Lagrange multipliers for constraint sets (2.7), (2.8), and (2.9), respectively, then it is possible to derive the following Lagrangian function:

$$
\begin{aligned}
L(\boldsymbol{x}, w; \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \equiv w \; + & \sum_{\{e \in E: v_i < v_j\}} \sum_{t \in \mathcal{T} \setminus \{0, T\}} \lambda_{e,t} \left[ \sum_{a \in \mathcal{A}} (x_{a,e,t} + x_{a,\bar{e},t}) - 1 \right] + \\
& + \sum_{a \in \mathcal{A}} \sum_{e \in E} \sum_{t \in \mathcal{T} \setminus \{0\}} \mu_{a,e,t} \left[ x_{a,e,t} + \sum_{\{a' \in \mathcal{A}: a' \neq a\}} (x_{a',e,t-1} + x_{a',\bar{e},t-1}) - 1 \right] + \\
& + \sum_{a \in \mathcal{A}} \nu_a \left[ T + 1 - w - \sum_{t \in \mathcal{T}} x_{a,d_a,t} \right] \qquad (3.1)
\end{aligned}
$$

with

$$
\boldsymbol{\lambda} \geq \mathbf{0}; \;\; \boldsymbol{\mu} \geq \mathbf{0}; \;\; \boldsymbol{\nu} \geq \mathbf{0}. \qquad (3.2)
$$

In Equation 3.1, the indices of the Lagrange multipliers correspond to the agent, edge, and/or time indices over which the corresponding constraint sets range. As an example, one can consider the case of $\boldsymbol{\lambda}$, whose components are indexed by $e$ and $t$ in Equation 3.1. These indices exist because constraint set (2.7) ranges over (a) the set of edge indices $e = (v_i, v_j)$ such that $\{(v_i, v_j) \in E : i < j\}$, and (b) the set of time indices $t$ such that $t \in \mathcal{T} \setminus \{0, T\}$, and because there is a distinct Lagrange multiplier, $\lambda_{e,t}$, for each of these constraints.

The Lagrangian function $L(\boldsymbol{x}, w; \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ is an affine-linear function of $\boldsymbol{x}$ and $w$ for any given $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$, and it is an affine-linear function of $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$ for any given $\boldsymbol{x}$ and $w$. However, a piecewise-linear, concave function of the Lagrange multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$ can be derived by optimizing $L(\boldsymbol{x}, w; \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ with respect to $\boldsymbol{x}$ and $w$ for fixed values of the Lagrange multipliers, s.t. contraint sets (2.2)–(2.6) and (2.10) from the original MIP in Section 2.2. This gives us the following "relaxed" version of that MIP:

$$\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \equiv \min_{\boldsymbol{x},w} \Bigg\{ w + \sum_{\{e \in E: v_i < v_j\}} \sum_{t \in \mathcal{T} \setminus \{0,T\}} \lambda_{e,t} \left[ \sum_{a \in \mathcal{A}} (x_{a,e,t} + x_{a,\bar{e},t}) - 1 \right] +$$
$$+ \sum_{a \in \mathcal{A}} \sum_{e \in E} \sum_{t \in \mathcal{T} \setminus \{0\}} \mu_{a,e,t} \left[ x_{a,e,t} + \sum_{\{a' \in \mathcal{A}: a' \neq a\}} (x_{a',e,t-1} + x_{a',\bar{e},t-1}) - 1 \right] +$$
$$+ \sum_{a \in \mathcal{A}} \nu_a \left[ T + 1 - w - \sum_{t \in \mathcal{T}} x_{a,d_a,t} \right] \Bigg\} \tag{3.3}$$

s.t. the primal constraint sets (2.2)–(2.6) and (2.10).

The function $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ is the "*dual function*" of the MIP described in Chapter 2, that is induced by the relaxed constraint set. Since the minimization problem that defines $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ is a relaxation of the original MIP for the considered problem, the function $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ must form a lower bound on the optimal objective for the original MIP for any set of nonnegative values for $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$. Therefore, the tightest possible lower bound that $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ can offer is given by the optimal objective value of the following problem:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}} \theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \tag{3.4}$$

s.t. Equation (3.2)

This is known as the "*dual problem*" of the original MIP [4]. Let $\boldsymbol{\lambda}^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\nu}^*$ represent a set of Lagrange multipliers that optimize the dual problem, and let $\theta(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ represent the optimal objective value of the dual problem. Solving the dual problem has two benefits. First, as mentioned above, $\theta(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ offers a strong lower bound on the optimal objective value, which can be instrumental in evaluating the quality of any feasible solution to the original MIP when a problem instance is too large to be solvable to optimality by the available solvers. Second, $\boldsymbol{\lambda}^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\nu}^*$ can be used to evaluate a "price" of having a certain agent in a certain place at a certain time (i.e., a price for having $x_{a,e,t} = 1$ for some $a$, $e$, and $t$), and this information can be useful as part of a heuristic solution method for the original MIP.

For reasons that will become clear in Section 3.2.2, we can simplify the Lagrangian function further. First, it is possible to collect the Lagrange multiplier terms for each $x_{a,e,t}$ together and to rewrite Equation (3.1) as follows:

$$
\begin{aligned}
L(\boldsymbol{x}, w; \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \;=\; & \sum_{a\in\mathcal{A}} \nu_a (T+1) - \left[ \left( \sum_{\{e\in E: v_i < v_j\}} \sum_{t\in\mathcal{T}\setminus\{0,T\}} \lambda_{e,t} \right) + \right. \\
& \left. \left( \sum_{a\in\mathcal{A}} \sum_{e\in E} \sum_{t\in\mathcal{T}\setminus\{0\}} \mu_{a,e,t} \right) \right] + w(1 - \sum_{a\in\mathcal{A}} \nu_a) + \sum_{a\in\mathcal{A}} \left\{ \sum_{t\in\mathcal{T}\setminus\{0,T\}} \left( \right.\right. \\
& \sum_{\{e\in E: v_i < v_j\}} \left[ \lambda_{e,t} + \mu_{a,e,t} + \sum_{a'\in\mathcal{A}: a'\neq a} (\mu_{a',e,t+1} + \mu_{a',\bar{e},t+1}) \right] x_{a,e,t} + \\
& + \sum_{\{e\in E: v_i > v_j\}} \left[ \lambda_{\bar{e},t} + \mu_{a,e,t} + \sum_{a'\in\mathcal{A}: a'\neq a} (\mu_{a',e,t+1} + \mu_{a',\bar{e},t+1}) \right] x_{a,e,t} \left. \right) + \\
& \left. + \sum_{e\in E} \left[ \sum_{a'\in\mathcal{A}: a'\neq a} (\mu_{a',e,1} + \mu_{a',\bar{e},1}) \right] x_{a,e,0} + \sum_{e\in E} \mu_{a,e,T} x_{a,e,T} - \nu_a \sum_{t\in\mathcal{T}} x_{a,d_a,t} \right\}
\end{aligned} \tag{3.5}
$$

Let us further define:

$$
\Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}} \;\equiv\; - \left[ \left( \sum_{\{e\in E: v_i < v_j\}} \sum_{t\in\mathcal{T}\setminus\{0,T\}} \lambda_{e,t} \right) + \left( \sum_{a\in\mathcal{A}} \sum_{e\in E} \sum_{t\in\mathcal{T}\setminus\{0\}} \mu_{a,e,t} \right) \right]; \tag{3.6}
$$

$$
C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} \;\equiv\; \begin{cases} \lambda_{e,t} + \mu_{a,e,t} + \sum_{a'\in\mathcal{A}: a'\neq a} (\mu_{a',e,t+1} + \mu_{a',\bar{e},t+1}), \\ \forall a \in \mathcal{A},\ \forall t \in \mathcal{T} \setminus \{0,T\},\ \forall e \in E : v_i < v_j; \\ \lambda_{\bar{e},t} + \mu_{a,e,t} + \sum_{a'\in\mathcal{A}: a'\neq a} (\mu_{a',e,t+1} + \mu_{a',\bar{e},t+1}), \\ \forall a \in \mathcal{A},\ \forall t \in \mathcal{T} \setminus \{0,T\},\ \forall e \in E : v_i > v_j; \end{cases} \tag{3.7}
$$

$$
C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,0} \;\equiv\; \sum_{a'\in\mathcal{A}: a'\neq a} (\mu_{a',e,1} + \mu_{a',\bar{e},1}),\ \forall a \in \mathcal{A},\ \forall e \in E; \tag{3.8}
$$

$$
C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,T} \;\equiv\; \mu_{a,e,T},\ \forall a \in \mathcal{A},\ \forall e \in E. \tag{3.9}
$$

This allows for the following compact representation of the Lagrangian function:

$$\begin{aligned}
L(\boldsymbol{x}, w; \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \;=\; & \sum_{a\in\mathcal{A}} \nu_a(T+1) + \Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}} + w(1-\sum_{a\in\mathcal{A}}\nu_a) + \\
\sum_{a\in\mathcal{A}} \Bigg\{ \sum_{e\in E}\sum_{t\in\mathcal{T}\setminus\{0,T\}} & C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} x_{a,e,t} + \sum_{e\in E} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,0} x_{a,e,0} + \sum_{e\in E} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,T} x_{a,e,T} - \sum_{t\in\mathcal{T}} \nu_a x_{a,d_a,t} \Bigg\} = \\
= \sum_{a\in\mathcal{A}} \nu_a(T+1) + \Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}} + w(1-\sum_{a\in\mathcal{A}}\nu_a) & + \sum_{a\in\mathcal{A}} \left[ \sum_{e\in E}\sum_{t\in\mathcal{T}} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} x_{a,e,t} - \nu_a \sum_{t\in\mathcal{T}} x_{a,d_a,t} \right]
\end{aligned} \tag{3.10}$$

It follows, then, that the dual function can be simplified as

$$\begin{aligned}
\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \;=\; & (T+1)\sum_{a\in\mathcal{A}} \nu_a + \Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}} + \\
& + \min_{\boldsymbol{x},w} \left\{ w(1-\sum_{a\in\mathcal{A}}\nu_a) + \sum_{a\in\mathcal{A}} \left[ \sum_{e\in E}\sum_{t\in\mathcal{T}} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} x_{a,e,t} - \nu_a \sum_{t\in\mathcal{T}} x_{a,d_a,t} \right] \right\}
\end{aligned} \tag{3.11}$$

s.t. the primal constraint sets (2.2)–(2.6) and (2.10).

Here, $\Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}}$ condenses the sum of all components of the Lagrangian function that do not depend on changes in $\boldsymbol{x}$ or $w$. By contrast, $C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t}$ effectively defines a cost incurred by the Lagrangian function when $x_{a,e,t} = 1$ and $e \neq d_a$. When $e = d_a$, the cost of placing agent $a$ on edge $d_a$ is given by $C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} - \nu_a$. These costs will be used in a dynamic program described in Section 3.2.2 to select, for any given $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$, an $\boldsymbol{x}$ that achieves the minimization embedded in Equation 3.3.

Minimizing the Lagrangian function with respect to $w$ requires a different approach. In particular, we can start by noticing that, from the definition of the original MIP described in

Chapter 2,

$$0 \leq w \leq T + 1. \tag{3.12}$$

These bounds are implied by constraint sets (2.9) and (2.10) in the original MIP. Applying these bounds to the right-hand side of Equation (3.11) tells us that, to minimize the Lagrangian function with respect to $w$ with $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$ given, we should set

$$\begin{aligned} w &:= 0 \quad \text{for} \quad \sum_{a \in \mathcal{A}} \nu_a < 1.0; \\ w &:= T + 1 \quad \text{for} \quad \sum_{a \in \mathcal{A}} \nu_a > 1.0. \end{aligned} \tag{3.13}$$

When $\sum_{a \in \mathcal{A}} \nu_a = 1.0$, on the other hand, $w$ vanishes from the right-hand side of Equation (3.11), implying (i) that $w$ can take any value between $0$ and $T + 1$ when the dual function is evaluated, and (ii) that it is possible to simplify the dual function (and the dual problem) by excluding from direct consideration the value of $w$ (i.e., dropping the $w$ term). This last remark is useful because the optimal solution of the dual problem is guaranteed to have $\sum_{a \in \mathcal{A}} \nu_a = 1.0$.

**Proposition 3.1.1.** *At any optimal solution $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ of the dual problem that is defined by Equations (3.4) and (3.2), it must hold that*

$$\sum_{a \in \mathcal{A}} \nu_a^* = 1.0 \tag{3.14}$$

**Proof:** Let $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ represent an optimal solution of the dual problem defined in Equations (3.4) and (3.2), and let $(\boldsymbol{x}^*, w^*)$ denote the optimal solution of the minimization problem used to define the dual function in Equation (3.11).

Note that, since $x_{a,e,t}$ is a binary variable $\forall a \in \mathcal{A}$, $\forall e \in E$, $\forall t \in \mathcal{T}$, and since

$|\mathcal{T}| = T + 1$,

$$\forall a \in \mathcal{A},\ \ T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t} \ \geq\ 0, \tag{3.15}$$

with equality holding for an agent $a \in \mathcal{A}$ only if $s_a = d_a$, and with

$$\sum_{a \in \mathcal{A}} \left[T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t}\right] > 0 \tag{3.16}$$

for any non-trivial instance of the original MIP described in Chapter 2. With this in mind, we can define an $|\mathcal{A}|$-dim vector $\boldsymbol{u}^*$ such that its components are given by $T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t},\ a \in \mathcal{A}$.

Next, let us assume that $\sum_{a \in \mathcal{A}} \nu^*_a < 1.0$. Equation (3.13) tells us that $w^*$ must equal $0$. Since $1.0 - \sum_{a \in \mathcal{A}} \nu^*_a > 0$, we can define a positive constant

$$\delta \in \left(0, \frac{1.0 - \sum_{a \in \mathcal{A}} \nu^*_a}{\sum_{a \in \mathcal{A}} \left[T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t}\right]}\right) \tag{3.17}$$

and use it to define the vector

$$(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}) = (\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*) + \delta \cdot (\mathbf{0}, \mathbf{0}, \boldsymbol{u}^*) \tag{3.18}$$

For the above vector of Lagrange multipliers, the dual function can be written as

$$\begin{aligned} \theta(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}) \ &\equiv\ (T+1) \sum_{a \in \mathcal{A}} \hat{\nu}_a + \Delta_{\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}} + \\ &+ \min_{\boldsymbol{x}, w} \left\{ w(1 - \sum_{a \in \mathcal{A}} \hat{\nu}_a) + \sum_{a \in \mathcal{A}} \left[ \sum_{e \in E} \sum_{t \in \mathcal{T}} C^{\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}}_{a,e,t} x_{a,e,t} - \hat{\nu}_a \sum_{t \in \mathcal{T}} x_{a,d_a,t} \right] \right\} \end{aligned} \tag{3.19}$$

s.t. the primal constraint sets (2.2)–(2.6) and (2.10).

If $(\hat{\boldsymbol{x}}, \hat{w})$ represents an optimal solution to the minimization problem embedded in (3.19), then it follows that $\hat{w} = 0$ (based on Equation (3.13) and the fact that Equation (3.17) defines $\delta$ to ensure that $\sum_{a \in \mathcal{A}} \hat{\nu}_a < 1.0$). This allows us to relate $\theta(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}})$ to $\theta(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ as follows:

$$
\begin{aligned}
\theta(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}}) &= (T+1)\sum_{a \in \mathcal{A}} \hat{\nu}_a + \Delta_{\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}} + \sum_{a \in \mathcal{A}} \left[ \sum_{e \in E} \sum_{t \in \mathcal{T}} C_{a,e,t}^{\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}} \hat{x}_{a,e,t} - \hat{\nu}_a \sum_{t \in \mathcal{T}} \hat{x}_{a,d_a,t} \right] \\
&= \Delta_{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*} + \sum_{a \in \mathcal{A}} \left[ \sum_{e \in E} \sum_{t \in \mathcal{T}} C_{a,e,t}^{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*} \hat{x}_{a,e,t} + \nu_a^* \cdot \big(T + 1 - \sum_{t \in \mathcal{T}} \hat{x}_{a,d_a,t}\big) \right] + \\
&\quad + \delta \cdot \sum_{a \in \mathcal{A}} \big(T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t}\big) \cdot \big(T + 1 - \sum_{t \in \mathcal{T}} \hat{x}_{a,d_a,t}\big) \\
&\geq \Delta_{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*} + \sum_{a \in \mathcal{A}} \left[ \sum_{e \in E} \sum_{t \in \mathcal{T}} C_{a,e,t}^{\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*} x^*_{a,e,t} + \nu_a^* \cdot \big(T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t}\big) \right] + \\
&\quad + \delta \cdot \sum_{a \in \mathcal{A}} \big(T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t}\big) \cdot \big(T + 1 - \sum_{t \in \mathcal{T}} \hat{x}_{a,d_a,t}\big) \\
&= \theta(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*) + \delta \cdot \sum_{a \in \mathcal{A}} \big(T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t}\big) \cdot \big(T + 1 - \sum_{t \in \mathcal{T}} \hat{x}_{a,d_a,t}\big) \\
&> \theta(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)
\end{aligned} \tag{3.20}
$$

The second equality above is based directly on the definition of $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\nu}})$ given in Equation 3.18. The first inequality follows from that based on the definition of $\boldsymbol{x}^*$. The next equality follows due to the definition of $\boldsymbol{x}^*$, the fact that $w^* = 0$, and Eq. (3.15). The strictness of the final inequality follows from the fact that $T + 1 - \sum_{t \in \mathcal{T}} x^*_{a,d_a,t} \geq 0$ and $T + 1 - \sum_{t \in \mathcal{T}} \hat{x}_{a,d_a,t} \geq 0$, with both being strictly positive for any agent $a \in \mathcal{A}$ with $s_a \neq d_a$.

Since $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ was selected as an optimal solution for the dual problem described in Equation (3.4), the strict inequality in the last part of Equation (3.20) defines a contradiction. This implies that $\sum_{a \in \mathcal{A}} \nu_a^*$ cannot be less than $1.0$. A similar proof by contradiction can be constructed (using $\hat{w} = T + 1$ in place of $\hat{w} = 0$) to show that $\sum_{a \in \mathcal{A}} \nu_a^*$ cannot be greater than $1.0$. Together, these proofs validate Proposition 3.1.1. $\square$

In the following discussion, it will be useful to define a set $X$ as

$$X \;\equiv\; \Big\{\boldsymbol{x} \text{ satisfying the primary constraint sets (2.2)–(2.6) and (2.10)}\Big\} \tag{3.21}$$

This enables us to use Proposition 3.1.1 to reduce the dual problem of Eq. (3.11) to the following form:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}} \theta(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}) \;\equiv\; (T+1)+\Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}}+\min_{\boldsymbol{x}\in X}\left\{\sum_{a\in\mathcal{A}}\left[\sum_{e\in E}\sum_{t\in\mathcal{T}} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t}x_{a,e,t}-\nu_a\sum_{t\in\mathcal{T}}x_{a,d_a,t}\right]\right\} \tag{3.22}$$

s.t.

$$\boldsymbol{\lambda}\geq\mathbf{0};\;\; \boldsymbol{\mu}\geq\mathbf{0};\;\; \boldsymbol{\nu}\geq\mathbf{0};\;\; \sum_{a\in\mathcal{A}}\nu_a = 1.0 \tag{3.23}$$

This formulation of the dual problem reveals some additional properties. First, the minimization problem embedded in the right-hand side of Equation (3.22) is *separable* with respect to the agent set, $\mathcal{A}$. That is, it is possible, for any given $(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu})$, to perform the minimization with respect to each agent in series, without compromising the minimization with respect to all agents. This separability has very significant consequences in the developments that follow.

Secondly, the structure of Equations (3.22) and (3.23) makes it possible to simplify Equation (3.22) once again to

$$\theta(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}) = \min_{\boldsymbol{x}\in X}\Big\{(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu})^T\cdot\boldsymbol{g}(\boldsymbol{x})\Big\}+(T+1), \tag{3.24}$$

where $\boldsymbol{g}(\boldsymbol{x})$ can be calculated for any given $\boldsymbol{x}\in X$ based on Equation (3.22), using Equations (3.6)–(3.9) to define $\Delta_{\boldsymbol{\lambda},\boldsymbol{\mu}}$ and $C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t}$. Based on Equation (3.24), it becomes clear that the dual function, $\theta$, is a *polyhedral concave* function of the Lagrange multipliers $(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu})$. This structure implies that $\theta$ is non-differentiable in regions where the binding hyperplanes intersect, meaning that the dual problem defined by Equations (3.22) and (3.23) must be addressed using subgradient optimization methods. Such methods often suffer from slow and/or non-monotonic convergence towards optimality. However, in Section 3.2 we define a subgradient-based algorithm for the problem formulated in Equations (3.22) and (3.23), using the above structure of $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ to guarantee *finite, monotonic* convergence.

## 3.2 A Dual-Ascent Approach to the Optimization of the Relaxed Dual Problem

### 3.2.1 Definition of a Streamlined Dual-Ascent Algorithm

This section describes a subgradient algorithm for the solution of the following problem:

$$\max_{\boldsymbol{\pi}} f(\boldsymbol{\pi}) \equiv \min_{i \in \mathcal{I}} \left\{ \boldsymbol{\pi}^T \cdot \boldsymbol{g}_i \right\} \tag{3.25}$$

s.t.

$$\sum_{j \in \tilde{\mathcal{J}}} \pi_j = 1.0; \quad \boldsymbol{\pi} \geq \mathbf{0}, \tag{3.26}$$

where $\mathcal{I}$ is a finite set with $|\mathcal{I}| = k$, $\dim(\boldsymbol{\pi}) = m$, and $|\tilde{\mathcal{J}}| = n \leq m$. The optimization problem described by Equations (3.25) and (3.26) is a generalization of the one described by Equations (3.22) and (3.23). This generalization is a consequence of eliminating any dependence on the set $X$ (as defined in Equation (3.21)) for the purposes of the present section. However, we retain here a concave, polyhedral objective function, subject to nonnegativity constraints and the restriction to a simplex as described in Equation (3.26).

Some methods already exist that are capable of finding optimal or $\epsilon$-optimal solutions to constrained convex optimization problems such as the one described by Equations (3.25) and (3.26). Notably, cutting-plane methods (as described in [25] and [26]) can be used to approach such problems, and they are even known to converge, in a finite number of iterations, to optimal solutions for polyhedral convex or concave problems such as the one described by Equations (3.25) and (3.26). But these methods often converge in a non-monotonic manner exhibiting an erratic transient behavior. This last problem can be mitigated with "stabilization" techniques such as the trust region method, the regularized method [25, 26], or the level method [16]. The level method developed by Lemarechal, Nemirovskii, and Nesterov (as described in [32]) is known to converge to $\epsilon$-optimal solutions in a polynomial number of iterations with respect to $\frac{1}{\epsilon}$. On the other hand, the dual-ascent method described in this section is distinguished by its ability to guarantee *finite, monotonic* convergence to an optimal solution.

In order to help us prove these convergence properties, first we must establish necessary and sufficient conditions for the optimality of a solution to the problem formulated in Equations (3.25) and (3.26). These conditions will be helpful as we construct a solution algorithm.

**Theorem 3.2.1.** *Let $\boldsymbol{\pi}$ represent a feasible solution for the problem described by Equations (3.25) and (3.26). Also define $\Omega(\boldsymbol{\pi}) \equiv \{j \in \{1, \ldots, m\} : \pi_j > 0\}$ and $\tilde{\mathcal{I}} \equiv \arg\min_{i \in \mathcal{I}} \{\boldsymbol{\pi}^T \cdot \boldsymbol{g}_i\}$. Finally, let $I_{m \times m}$ denote the $(m \times m)$ diagonal matrix, and define $\mathbf{1}_{\tilde{\mathcal{J}}}$ as an $m$-dimensional binary vector whose unit elements coincide with the components that are defined by the set $\tilde{\mathcal{J}}$.*

*Then, $\boldsymbol{\pi}$ is an optimal solution for the formulation of Equations (3.25) and (3.26) if and only if* (iff) *the following linear program (LP) has an optimal value of zero:*

$$\min_{\boldsymbol{\gamma}, \boldsymbol{y}, \boldsymbol{z}, \delta} \sum_{j=1}^{m} y_j + \sum_{j \in \Omega(\boldsymbol{\pi})} z_j \tag{3.27}$$

*s.t.*

$$[I_{m \times m} \quad -I_{m \times m} \; \mathbf{1}_{\tilde{\mathcal{J}}}] \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{z} \\ \delta \end{bmatrix} = \sum_{i \in \tilde{\mathcal{I}}} \gamma_i \, \boldsymbol{g}_i \tag{3.28}$$

$$\sum_{i \in \tilde{\mathcal{I}}} \gamma_i = 1.0 \tag{3.29}$$

$$\boldsymbol{\gamma} \geq 0; \quad \boldsymbol{y} \geq 0; \quad \boldsymbol{z} \geq 0. \tag{3.30}$$

**Proof:** Based on Equation (3.25) and the definition of $\tilde{\mathcal{I}}$, it is easy to see that the right-hand-side of Equation (3.28) represents the subdifferential $\partial f(\boldsymbol{\pi})$ of the objective function $f(\boldsymbol{\pi})$, parameterized by the variables $\gamma_i$. The vectors $\boldsymbol{y}$, $\boldsymbol{z}$ and the scalar $\delta$ make it possible to decompose any subgradient $\boldsymbol{g} \in \partial f(\boldsymbol{\pi})$ to (i) a vector that is perpendicular to the hyperplane $\sum_{j \in \tilde{\mathcal{J}}} \pi_j = 1.0$, and (ii) the projection of $\boldsymbol{g}$ onto this hyperplane. More explicitly, vector $\boldsymbol{y}$ will represent the positive part of this projection, vector $-\boldsymbol{z}$ will represent the negative part, and vector $\delta \mathbf{1}_{\tilde{\mathcal{J}}}$ will represent the component that is perpendicular to the hyperplane $\sum_{j \in \tilde{\mathcal{J}}} \pi_j = 1.0$.

The theory of subgradient-based optimization methods tells us that $\boldsymbol{\pi}$ will be an optimal solution for the formulation of Equations (3.25) and (3.26) *iff* the subdifferential $\partial f(\boldsymbol{\pi})$ contains a subgradient $\boldsymbol{g}$ with a projection $\overline{\boldsymbol{g}}$ onto the hyperplane $\sum_{j \in \tilde{\mathcal{J}}} \pi_j = 1.0$ such that

$$\overline{g}_j \begin{cases} = 0, & \forall j \in \Omega(\boldsymbol{\pi}) \\ \leq 0, & \text{otherwise} \end{cases} \tag{3.31}$$

Then, given (i) that the variables $\gamma_i$ parameterize the subdifferential, (ii) that the components of $\boldsymbol{y}$ and $\boldsymbol{z}$ are nonnegative, and (iii) the decomposition of subgradients described above, $\boldsymbol{\pi}$ will be an optimal solution for the formulation of Equations (3.25) and (3.26) *iff* $\sum_{j=1}^{m} y_j = 0 \wedge \sum_{j \in \Omega(\boldsymbol{\pi})} z_j = 0$. Therefore, the condition of Equation (3.31) will be satisfied *iff* the LP of Theorem 3.2.1 has an optimal value of zero. □

Next, we formulate the dual of the LP of Theorem 3.2.1. This will provide an alternative optimality test for the primal formulation of Equations (3.25) and (3.26). Our primary reason for considering it, however, is because it can be used to find an improving direction when a given $\boldsymbol{\pi}$ fails to meet the optimality condition described in Theorem 3.2.1.

In this dual LP formulation, $\boldsymbol{\rho}$ will denote the vector collecting the dual variables that correspond to the constraints of Equation (3.28), and $u$ will denote the dual variable corresponding to the constraint of Equation (3.29). Then, using standard LP duality theory, the dual LP formulation can be written as follows:

$$\max_{\boldsymbol{\rho}, u} \; u \tag{3.32}$$

s.t.

$$-1.0 \leq \rho_j \leq 1.0, \;\; \forall j \in \Omega(\boldsymbol{\pi}) \tag{3.33}$$

$$0.0 \leq \rho_j \leq 1.0, \;\; \forall j \in \Omega^c(\boldsymbol{\pi}) \equiv \{1, \ldots, m\} \setminus \Omega(\boldsymbol{\pi}) \tag{3.34}$$

$$\sum_{j \in \tilde{\mathcal{J}}} \rho_j = 0.0 \tag{3.35}$$

$$u \leq \boldsymbol{\rho}^T \cdot \boldsymbol{g}_i, \quad \forall i \in \tilde{\mathcal{I}}. \tag{3.36}$$

The utility of this dual formulation is established by the following theorem:

**Theorem 3.2.2.** *The LP described in Equations (3.32)–(3.36) will have an optimal objective value of zero* iff *the vector $\boldsymbol{\pi}$ is an optimal solution for the primal formulation of Equations (3.25)–(3.26).*

*In addition, if the optimal objective value of this LP is positive, then the vector $\boldsymbol{\rho}^*$ taken from any optimal solution, defines a feasible direction of ascent for $\boldsymbol{\pi}$, with respect to the primal problem of Equations (3.25) and (3.26).*

**Proof:** The first half of the above theorem follows immediately from Theorem 3.2.1 and LP strong duality. Therefore, it remains for us only to prove the second part.

Constraints (3.33)–(3.35) of the dual LP formulation guarantee that a $\boldsymbol{\pi}$ vector that is a feasible solution for the formulation of Equations (3.25) and (3.26), can be moved some positive distance in direction $\boldsymbol{\rho}^*$ while retaining feasibility.

To show that $\boldsymbol{\rho}^*$ is also a direction of ascent for $f(\boldsymbol{\pi})$ (i.e., that it will improve the objective function of the primal problem for some sufficiently small, positive step size), it is sufficient to show that

$$\forall \boldsymbol{g} \in \partial f(\boldsymbol{\pi}), \;\; (\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g} > 0. \tag{3.37}$$

Since the subdifferential $\partial f(\boldsymbol{\pi})$ is equivalent to the convex hull of the vectors $\boldsymbol{g}_i, \; i \in \tilde{\mathcal{I}}$, we can write

$$(\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g} \;\geq\; \min_{i \in \tilde{\mathcal{I}}} (\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g}_i = u^* > 0. \tag{3.38}$$

The positivity claimed in Equation (3.38) follows from two facts: (i) for any suboptimal $\boldsymbol{\pi}$, the resulting primal problem (and, by strong duality, the resulting dual problem) will have a positive optimal objective value, $u^*$; and (ii) Equation (3.36), together with the maximization of $u^*$, ensures that $\min_{i \in \tilde{\mathcal{I}}} (\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g}_i = u^*$. $\square$

Theorem 3.2.2 tells us that, given a vector $\boldsymbol{\pi}$ that is feasible with respect to the problem described in Equations (3.25) and (3.26), it is possible to improve the objective function $f(\boldsymbol{\pi})$ while maintaining feasibility with respect to Equation (3.26) by taking a sufficiently small step in direction $\boldsymbol{\rho}^*$. Assuming also a mechanism to find such a step size, one can straightforwardly obtain an optimization procedure for the problem of Equations (3.25) and (3.26), as detailed in Figure 3.1. Next, we turn to the issue of determining an appropriate step size, $s$, at each iteration.

Since $f(\boldsymbol{\pi})$ has a concave, polyhedral structure, it is possible to find an optimized value of $s$ by solving a single-variable optimization problem with a piecewise-linear, concave function defined by $f(\boldsymbol{\pi} + s \cdot \boldsymbol{\rho}^*)$, s.t. $s > 0$. Efficient solution methodologies for problems of this type are available in a variety of texts on gradient-based optimization methods (e.g., [4]). However, we will consider a different mechanism for the specification of $s$ that takes

1. Start with some feasible solution of the considered formulation; e.g., set $\pi_j$ equal to $1/n$, if $j \in \tilde{\mathcal{J}}$, and zero, otherwise.
2. Based on the solution, $\boldsymbol{\pi}$, solve the corresponding dual LP formulation of Equations (3.32)–(3.36).
3. If the optimal value of the dual LP objective function, $u^*$, is equal to zero, then exit, returning the current vector $\boldsymbol{\pi}$ as an optimal solution.
4. If $u^* \neq 0$, update the current solution $\boldsymbol{\pi}$ to $\boldsymbol{\pi} + s \cdot \boldsymbol{\rho}^*$, where $\boldsymbol{\rho}^*$ is obtained from the optimal solution of the dual LP formulation, and $s$ is an aptly-selected step size, and return to Step 2.[1]

Figure 3.1: An iterative algorithm for the solution of the optimization problem of Equations (3.25) and (3.26), based on the results of Theorem 3.2.2.

advantage of the structure of the optimization problem described by Equations (3.25) and (3.26) and that will allow us to argue the *monotone, finite* convergence of the algorithm presented here.

When considering the selection of the step size, there are two types of events that may constrain the optimization of this selection when moving in direction $\boldsymbol{\rho}^*$ from a feasible solution $\boldsymbol{\pi}$. Namely:

**Type-I event:** Some new vectors $\boldsymbol{g}_i$ enter the current set of the minimizers of the right-hand-side of Eq. (3.25). Note that this does not necessarily mean that $f$ cannot be improved further by continuing in direction $s$. It does, however, imply that if $f(\boldsymbol{\pi} + s \cdot \boldsymbol{\rho}^*)$ does continue to improve, then it will improve more slowly than before the Type-I event, on account of its piecewise-linear, concave structure.

**Type-II event:** One or more of the decision variables $\pi_j$ reach their lower limit of zero (cf. Equation (3.26)).

If both types of events are used to bound the step size $s$–i.e., if $s$ is selected to be the largest step size achievable before one of these events occurs–and if the algorithm returns to Step 2 (as described in Figure 3.1) as soon as such a step size is obtained, then it can be shown that the resulting algorithm converges monotonically and finitely toward an optimal solution of the problem formulated in Equations (3.25) and (3.26). Theorem 3.2.3 establishes this result.

**Theorem 3.2.3.** *Consider a variation of the algorithm in Figure 3.1 aimed at optimizing an instance of the problem formulated in Equations (3.25) and (3.26), where the step size $s$ corresponds to the largest step size possible such that no type-I or type-II events occur for any step size smaller than $s$. Then, there is an implementation of this algorithm that will either converge to an optimal solution for the considered problem instance, or determine that the optimum is unbounded in a* finite *number of steps. Furthermore, the sequence of the values for $f$ that are generated by the successive iterations of this algorithm is* monotonically increasing.

**Proof:** The monotonic increase of the sequence of $f$-values that are generated by successive iterations of the given algorithm follows immediately from Theorem 3.2.2 and the decision to select step sizes based on the type-I and type-II events described above. It therefore remains for us to prove only that the algorithm terminates, regardless of the result, in a finite number of steps.

To that end, we can note first that the number of dual LP formulations of Equations (3.32)–(3.36) is finite since, for any given $\boldsymbol{\pi}$, the dual LP is uniquely determined by the index sets $\Omega(\boldsymbol{\pi})$ and $\tilde{\mathcal{I}}$. Since the number of distinct sets that can exist for either $\Omega(\boldsymbol{\pi})$ or $\tilde{\mathcal{I}}$ is finite, it follows that the number of dual LPs that can be formulated is finite.

Furthermore, it is possible to show that the optimal objective value of the dual LP described in Equations (3.32)–(3.36) cannot increase from one iteration to the next. To see this, consider a "new" solution $\hat{\boldsymbol{\pi}} = \boldsymbol{\pi} + s \cdot \boldsymbol{\rho}^*$. Regardless of whether $\hat{\boldsymbol{\pi}}$ was reached as a consequence of a type-I event or of a type-II event, the new dual LP defined at $\hat{\boldsymbol{\pi}}$ will include the constraints based on the vectors $\boldsymbol{g}_i$ that define the optimal solution of the dual LP at $\boldsymbol{\pi}$. The occurrence of a type-I event implies that the set of $\boldsymbol{g}_i$ vectors will be augmented by one or more new vectors. These vectors arise from the fact that a type-I event occurs when we reach a point where new indices minimize the right-hand-side of Equation (3.25). We can define the resulting index set as $\hat{\mathcal{I}}$ and note that, since $\hat{\mathcal{I}}$ defines a set of dual constraints that comprises a superset of the binding constraints for the dual LP formulated at $\boldsymbol{\pi}$, it follows that the resulting optimal objective value of the (maximized) dual LP (i.e., the dual LP formulated from $\hat{\boldsymbol{\pi}}$) will be less than or equal to the optimal objective value of the dual LP formulated from $\boldsymbol{\pi}$.

Furthermore, any sequence of type-I events that do not reduce the optimal objective value of the dual program of Equations (3.32)–(3.36) will result in increasingly constrained LPs until a reduction of the optimal objective does occur. More formally, let $\tilde{\mathcal{I}}_{\boldsymbol{\rho}^*} \subseteq \tilde{\mathcal{I}}$ represent the set of constraints based on the vectors $\boldsymbol{g}_i$ that are binding (i.e., that hold at equality) at the optimal solution $\boldsymbol{\rho}^*$, and let $\hat{\mathcal{I}}_{\hat{\boldsymbol{\rho}}^*} \subseteq \hat{\mathcal{I}}$ represent a similarly-defined set for the next dual program of Equations (3.32)–(3.36), following a type-I event. If the optimal objective value of the new dual program does not improve (i.e., if it is equal to the optimal objective value of the preceding program), then the constraints $\tilde{\mathcal{I}}_{\boldsymbol{\rho}^*}$ will be binding for the optimal solution of the new program, and will further bind the new constraint(s) introduced by the type-I event. This gives us that $\tilde{\mathcal{I}}_{\boldsymbol{\rho}^*} \subset \hat{\mathcal{I}}_{\hat{\boldsymbol{\rho}}^*}$. Since each of these constraints is taken from a vector $\boldsymbol{g}_i$ that is a member of a finite set, any progression of type-I events must eventually result in a reduction of the optimal objective value of the dual program of Equations (3.32)–(3.36).

For type-II events, let $\omega(\boldsymbol{\pi})$ denote the following superset of $\Omega(\boldsymbol{\pi})$:

$$\omega(\boldsymbol{\pi}) = \Omega(\boldsymbol{\pi}) \cup \{i \in \{1, 2, ..., m\} | \rho_i^* > 0\}, \tag{3.39}$$

where $\boldsymbol{\rho}^*$ refers specifically to the optimal solution for the LP described by Equations (3.32)–(3.36), used to transition between $\boldsymbol{\pi}$ and $\hat{\boldsymbol{\pi}}$. Now, let $\mathcal{D}(\boldsymbol{\pi})$ represent the set of feasible $\boldsymbol{\rho}$ vectors for the dual LP formulation based on $\boldsymbol{\pi}$ and Constraints (3.33)–(3.35), and let $\tilde{\mathcal{D}}(\boldsymbol{\pi})$ represent a set characterized similarly to $\mathcal{D}(\boldsymbol{\pi})$, but with Constraints (3.33) and (3.34) replaced by

$$-1.0 \le \rho_j \le 1.0, \;\; \forall j \in \omega(\boldsymbol{\pi}) \tag{3.40}$$

and

$$0.0 \le \rho_j \le 1.0, \;\; \forall j \in \omega^c(\boldsymbol{\pi}) \equiv \{1, \dots, m\} \setminus \omega(\boldsymbol{\pi}). \tag{3.41}$$

Then,

$$\max_{\boldsymbol{\rho} \in \hat{\mathcal{D}}(\boldsymbol{\pi})} \min_{i \in \tilde{\mathcal{I}}} \left\{ \boldsymbol{\rho}^T \cdot \boldsymbol{g}_i \right\} \; = \; \max_{\boldsymbol{\rho} \in \mathcal{D}(\boldsymbol{\pi})} \min_{i \in \tilde{\mathcal{I}}} \left\{ \boldsymbol{\rho}^T \cdot \boldsymbol{g}_i \right\}. \tag{3.42}$$

Since type-II events do not alter $\tilde{\mathcal{I}}$ but instead set one or more nonzero vector elements of $\boldsymbol{\pi}$ to zero at $\hat{\boldsymbol{\pi}}$, we find that $\Omega(\hat{\boldsymbol{\pi}}) \subset \omega(\boldsymbol{\pi})$. Though this inclusion does not quite ensure that the dual LP at $\hat{\boldsymbol{\pi}}$ is further constrained than the dual LP at $\boldsymbol{\pi}$, it does guarantee that the dual LP at $\hat{\boldsymbol{\pi}}$ can be formulated as a further-constrained version of an LP that has the same optimal objective value as the dual LP at $\boldsymbol{\pi}$. Hence, like type-I events, type-II events cannot increase the optimal objective value between successive iterations of the dual LP.

It is possible, however, for successive iterations of the dual LP to return identical optimal objective values due to degeneracy among the optimal values of $\boldsymbol{\rho}$. This degeneracy often manifests itself when motion along certain $\boldsymbol{\rho}$-coordinates is inconsequential to the value of the optimal dual objective. One consistent way to find unique solutions from these degenerate sets is to use a secondary LP to minimize the $l_1$ norm of $\boldsymbol{\rho}^*$. This method automatically sets all of the dual variables that do not impact the optimal value of the dual objective to zero. Hence, if $\boldsymbol{\rho}^*$ is an ascending direction for a given $\boldsymbol{\pi}$ that minimizes the $l_1$ norm among all vectors $\boldsymbol{\rho}$ that optimize the dual LP of Equations (3.32)–(3.36), then $\rho_i^* \ge 0$ only if $\boldsymbol{\rho}^*$ is a strictly better solution for the LP than all $\boldsymbol{\rho}$ with $\rho_i^* < 0$, and a similar statement can be made about any $i$ such that $\rho_i^* \le 0$.

Moreover, since (again) type-II events do not affect $\tilde{\mathcal{I}}$, this method of combating degeneracy has two consequences. First, if $\rho_i^* > 0$ for the dual LP at $\boldsymbol{\pi}$, then, following a type-II event, the corresponding component of the new ascending direction will still be nonnegative. More importantly, if $\rho_i^* \leq 0$ (where $\rho_i^* < 0$ is possible only if $i \in \Omega(\boldsymbol{\pi})$), then, following a type-II event, the corresponding elements of the new ascending direction will also be less than or equal to zero.

The latter of these consequences, combined with the fact that a type-II event converts one or more type-(3.33) constraints into type-(3.34) constraints, implies that $\omega(\hat{\boldsymbol{\pi}}) \subset \omega(\boldsymbol{\pi})$, where the notation here implies a proper subset. Since the cardinality of $\omega(\boldsymbol{\pi})$ cannot exceed the number of Lagrange multipliers for any $\boldsymbol{\pi}$, this implies that a new reduction of the optimal value of the dual LP must eventually take place. Since the number of distinct dual LPs that can be formulated (and hence the number of distinct optimal values for the dual objective) is also finite, it then follows that the algorithm must terminate after a finite number of iterations.

With this result for the iterative properties of the algorithm from Figure 3.1 and the concave polyhedral structure of $f$ in hand, we need only address the character of the termination. Two outcomes are possible:

1. The dual LP returns an optimal objective value of zero, which allows the algorithm to return the last $\boldsymbol{\pi}$ as a finite optimal solution.

2. The algorithm finds that the step size $s$ is not bounded by any further type-I or type-II events. This implies the unboundedness of the LP formulated in Equations (3.25) and (3.26) and returns a feasible solution $\boldsymbol{\pi}$ and an improving direction $\boldsymbol{\rho}$ that together define a ray along which the objective function of Equation (3.25) can improve without bound.

$\square$

In order to describe the proposed algorithm completely, a few implementational considerations must be addressed. The first point of discussion relates to the selection of the step size, $s$. While it is possible to perform a one-dimensional search to optimize the gain for the objective function of the considered problem (i.e., Equation (3.25)), and while such methods are addressed in the literature on subgradient optimization, the algorithm discussed in Theorem 3.2.3–where $s$ is selected to coincide with the first type-I or type-II event encountered while moving in a given direction, or to return with an indication of the unboundedness of the LP–lends itself more easily to a convergence proof. This method is similar in concept to the "pivot" operations performed in the LP simplex algorithm described in [59] and can be implemented in similar fashion. On the other hand, a method that optimizes the step size $s$, in contrast with the one described here, can skip over some type-I events as long as the objective function, (3.25), continues to improve. But then, the arguments used in the proof of Theorem 3.2.3 to show that the algorithm will converge in a finite number of steps, can also be extended to an implementation that optimizes step sizes.

The second point of discussion concerns the degeneracy in the optimal solution of the dual LP of Equations 3.32–3.36. Though this issue is addressed to a certain extent within the proof of Theorem 3.2.3, here we discuss the importance of addressing this degeneracy effectively. One problem that can arise when solutions are chosen arbitrarily (and when $\rho_i$ can vary without affecting the dual-objective function for some indices $i$) is that large numbers of type-II events can occur because certain components of $\boldsymbol{\pi}$ may be reduced to zero unnecessarily. At the same time, components of $\boldsymbol{\pi}$ that already equal zero may be increased unnecessarily, which provides more opportunities for type-II events to occur in the future, possibly with very small step sizes. In the worst case, cycling can occur if degenerate solutions are chosen arbitrarily. However, as remarked in the proof of Theorem 3.2.3, these problems can be addressed effectively through a "goal programming" ([59]) approach that minimizes the $l_1$ norm of the returned solution $\boldsymbol{\rho}^*$ as a secondary objective.

Our last practical concern is the size of the dual LP formulated in Equations (3.32)–

(3.36). It is important to note that it scales not only with the dimensionality of $\boldsymbol{\pi}$, but also with the cardinality of $\tilde{\mathcal{I}}$ used in each iteration. In some contexts (e.g., the optimization of the dual problem detailed in Section 3.1), these sets can grow prohibitively large for a direct implementation of the algorithm described in this section. However, in the next section we describe how the dual-ascent algorithm described in Figure 3.1 can be adapted to solve the dual problem of Equations (3.22) and (3.23) using efficient representations of $\mathcal{I}$ and $\tilde{\mathcal{I}}$.

### 3.2.2 Customization of the Algorithm of Section 3.2.1 to the Dual Problem of Section 3.1

In this section we further adapt the algorithm that we developed in Section 3.2.1 in order to solve efficiently the dual problem of Equations (3.22) and (3.23). The primary basis for this adaptation is the representation of the dual function through Equation (3.24).

We start this adaptation process by reminding the reader that Equation (3.21) defines $X$, for the sake of the dual problem, as the set of multi-agent schedules that deliver each agent from its starting location, $s_a$, to its destination, $d_a$, within the allotted time horizon, and that adhere to all single-agent constraints from the primal MIP of Chapter 2 (i.e., the constraints that describe the topology of the guidepath network and govern the movements of individual agents upon it). The set $X$ is useful for defining the dual problem of Equations (3.22) and (3.23), based on the Lagrangian function described in Equation (3.1), since it is the set over which $\boldsymbol{x}$ must be minimized in order to evaluate the dual function for given values of the Lagrange multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$. Since $X$ is not restricted by the coupling constraints (2.7)–(2.9), however, $X$ can alternatively be represented by

$$X = \times_{a\in\mathcal{A}} X_a, \tag{3.43}$$

where $X_a$ represents the set of all routes obeying the constraints that define $X$ for a particular agent, $a$. Though the number of distinct routes in any set $X_a$ can be combinatorially large with respect to the length of the time horizon, it can be represented as a spatio-temporal graph having no more than $|E| \cdot (T-1) + 2$ nodes and no more than $|E|^2 \cdot (T-2) + 2 \cdot |E|$

edges, and in practice, these numbers will tend to be much smaller. These graphs will be discussed in more detail later in this section. For now, however, we will turn our attention to the question of performing the minimization necessary to evaluate the dual function.

Evaluation of the dual function $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ for given Lagrange multipliers $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ requires us to identify an optimal element of the set $X$ that minimizes the right-hand-side of Equation (3.24). Computation and representation of the subdifferential $\partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ requires us to go further and find *all* elements of $X$ that minimize the right-hand-side of Equation (3.24) for the given $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$. Fortunately, Equation (3.22) shows us that this optimization problem can be decomposed into a series of separate minimization problems–one for each agent $a \in \mathcal{A}$. In addition, each one of these optimization problems is amenable to dynamic programming (DP), as described in [3]. In particular, we can specify the following value function, $V_a(e,t)$, for each DP:

For $t = T$:

$$V_a(e,t) = \begin{cases} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,T} - \nu_a & \text{if } e = d_a \\ \infty & \text{otherwise} \end{cases} \tag{3.44}$$

For $t \in \{0, 1, 2, ..., T-1\}$:

$$V_a(e,t) = \begin{cases} C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} - \nu_a + V_a(e, t+1) & \text{if } e = d_a \\ C^{\boldsymbol{\lambda},\boldsymbol{\mu}}_{a,e,t} + \min\limits_{e' \in e^\bullet \cup \{e\}} \{V_a(e', t+1)\} & \text{if } e \in E \setminus \{d_a\} \end{cases} \tag{3.45}$$

The value of '$\infty$' in Equation (3.44) is used to ensure that each DP will return a feasible solution that brings agent $a$ to its destination within the specified time horizon, $T$, so long as such a route exists. In general, a feasible solution will always exist for a connected guidepath network $G$, given a sufficiently large $T$.

If $\tilde{X}_a,\ a \in \mathcal{A}$, denote the sets of optimal solutions for the DPs described by Equations (3.44) and (3.45), then, due to the separability of the minimization problem in Equation (3.22), we can represent the full set of optimal solutions to that problem (i.e., the subset of $X$ composed of all $\boldsymbol{x}$ that define the dual function for a given set of Lagrange-multiplier values) as

$$\tilde{X} = \times_{a \in \mathcal{A}} \tilde{X}_a. \tag{3.46}$$

For each $a \in \mathcal{A}$, $\tilde{X}_a$ can be represented compactly as an acyclic digraph, which we can refer to as $\mathcal{G}_a$. The nodes of graph $\mathcal{G}_a$ represent location-time pairs $(e, t) \in E \times T$. In order to be included in the graph, a node must represent a pairing $(e, t)$ for the agent $a$ that is included in $\tilde{X}_a$. Directed arcs between nodes are drawn such that each path in $\mathcal{G}_a$ corresponds to a route in $\tilde{X}_a$. Once the value functions $V_a(e, t)$ have been established, the corresponding graphs $\mathcal{G}_a$ can be created by starting with nodes corresponding to $(s_a, 0)$ for each agent $a$, then applying the following logic: Visit each new node in turn. For each visited node $(e, t)$, if $e \neq d_a$, then create and attach nodes $(e', t+1)$ for every $e'$ that minimizes the second branch of the right-hand-side of Equation (3.45). If $e = d_a$, however, then $(e, t)$ will link only to $(e, t+1)$ $\forall t < T$. As long as the DP has a feasible solution with respect to the constraint sets (2.2)–(2.6) and (2.10), $(d_a, T)$ will form the single terminal node of $\mathcal{G}_a$.

The distribution of $\tilde{X}$ across $\mathcal{A}$ and its representation by means of the digraph $\mathcal{G}_a$ also enables a natural, similarly distributed representation of the subdifferential $\partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$. Any faithful representation of the subdifferential must admit and distinguish among all convex combinations of the elements of $\tilde{X}$. However, given $\mathcal{G}_a$, $a \in \mathcal{A}$, we can parameterize the convex hull of $\tilde{X}$–to be denoted by $Conv(\tilde{X})$)–using a set of unit flows, $\mathcal{F}_a$ on $\mathcal{G}_a$, such that each flow $\mathcal{F}_a$ transfers one unit from the source node of $\mathcal{G}_a$ to its terminal node. Each flow $\mathcal{F}_a$ can be represented as a vector $\boldsymbol{q}_a$ whose cardinality is equivalent to the number of directed arcs in $\mathcal{G}_a$. Its components represent the fluid flow across $\mathcal{G}_a$, and also adhere to the relevant flow-balance constraints (i.e., incoming flow equals outgoing flow except for the source node, which has no input and unit output, and the terminal node, which has unit input and zero output). These constraints can be represented in vector form as

$$F_a \cdot \boldsymbol{q}_a = \boldsymbol{\beta}_a^1, \ \ \forall a \in \mathcal{A}, \tag{3.47}$$

which, in turn, can be represented more compactly as

$$F \cdot \boldsymbol{q} = \boldsymbol{\beta}^1. \tag{3.48}$$

In Equation (3.48), $F$ is a block-diagonal matrix whose diagonal blocks are comprised of $F_a, a \in \mathcal{A}$, and the vectors $\boldsymbol{q}$ and $\boldsymbol{\beta}^1$ are the concatenations of the vector sets $\{\boldsymbol{q}_a, \forall a \in \mathcal{A}\}$ and $\{\boldsymbol{\beta}^1_a, \forall a \in \mathcal{A}\}$, respectively.

The vector $\boldsymbol{q}$, as it is defined in Equation (3.48), is equivalent to a parametric representation of the convex hull of $Conv(\tilde{X})$. The following lemma formalizes the relationship between this parametrization and the subdifferential of the dual function:

**Lemma 3.2.4.** *The subdifferential $\partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ of the dual function $\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ is given by*

$$\partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}) \ = \ \Big\{\boldsymbol{g}(\boldsymbol{q}) : \boldsymbol{q} \in Conv(\tilde{X})\Big\}, \tag{3.49}$$

*where the set $Conv(\tilde{X})$ is evaluated at $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$, and the function $\boldsymbol{g}(\cdot)$ is taken from Equation (3.24).*

**Proof:** Since $\boldsymbol{g}(\boldsymbol{x})$ is a linear function, it can be written in the form

$$\boldsymbol{g}(\boldsymbol{x}) \ = \ A \cdot \boldsymbol{x} + \boldsymbol{\beta}^2 \tag{3.50}$$

for an appropriately defined matrix $A$ and vector $\boldsymbol{\beta}^2$. [2]

[2]One can easily understand how these elements are "appropriately defined" by referring to the discussion of $\boldsymbol{g}(\boldsymbol{x})$ following Equation (3.24).

Equations (3.24) and (3.50), along with the definition of $\partial\theta(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu})$, allow us to write

$$\begin{aligned}\partial\theta(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}) &= \Big\{\tilde{g} \equiv \sum_{\boldsymbol{x}\in\tilde{X}} \xi_{\boldsymbol{x}}\big(A\cdot\boldsymbol{x}+\boldsymbol{\beta}^2\big) : \forall \boldsymbol{x}\in\tilde{X},\ \xi_{\boldsymbol{x}}\geq 0;\ \sum_{\boldsymbol{x}\in\tilde{X}}\xi_{\boldsymbol{x}} = 1.0\Big\}\\ &= \Big\{\tilde{g} \equiv A\cdot\sum_{\boldsymbol{x}\in\tilde{X}} \xi_{\boldsymbol{x}}\cdot\boldsymbol{x}+\boldsymbol{\beta}^2 : \forall \boldsymbol{x}\in\tilde{X},\ \xi_{\boldsymbol{x}}\geq 0;\ \sum_{\boldsymbol{x}\in\tilde{X}}a_{\boldsymbol{x}} = 1.0\Big\}\\ &= \Big\{\tilde{g} \equiv A\cdot\boldsymbol{q}+\boldsymbol{\beta}^2 : \boldsymbol{q}\in Conv(\tilde{X})\Big\}. \end{aligned} \tag{3.51}$$

□

Lemma 3.2.4 and Equation (3.48) describe an efficient representation of $\partial\theta(\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu})$, which allows us to adapt and apply Theorem 3.2.1 as follows:

**Corollary 3.2.5.** *Consider a point $\boldsymbol{\pi}\equiv(\boldsymbol{\lambda}^T,\boldsymbol{\mu}^T,\boldsymbol{\nu}^T)^T$ that constitutes a feasible solution for the dual problem of Equations (3.22) and (3.23). Furthermore, set $\Omega(\boldsymbol{\pi})\equiv\big\{j\in\{1,\ldots,m\}:\pi_j>0\big\}$ and $\dim(\boldsymbol{\pi})=m$. Finally, let $I_{m\times m}$ denote the $(m\times m)$ diagonal matrix, and let $\mathbf{1}_\nu$ denote an $m$-dimensional binary vector with its unit elements appearing at the components that correspond to the positions of the dual variables $\boldsymbol{\nu}$ in $\boldsymbol{\pi}$.*

*Then, $\boldsymbol{\pi}$ is an optimal solution for the considered dual problem* iff *the following LP has an optimal value of zero:*

$$\min_{\boldsymbol{q},\boldsymbol{y},\boldsymbol{z},\delta} \sum_{j=1}^{m} y_j + \sum_{j\in\Omega(\boldsymbol{\pi})} z_j \tag{3.52}$$

*s.t.*

$$\begin{bmatrix} F & 0 & 0 & 0\\ -A & I_{m\times m} & -I_{m\times m} & \mathbf{1}_\nu \end{bmatrix}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{y}\\ \boldsymbol{z}\\ \delta\end{bmatrix} = \begin{bmatrix}\boldsymbol{\beta}^1\\ \boldsymbol{\beta}^2\end{bmatrix} \tag{3.53}$$

$$\boldsymbol{q}\geq 0;\ \ \boldsymbol{y}\geq 0;\ \ \boldsymbol{z}\geq 0. \tag{3.54}$$

If we dualize this LP and let $\boldsymbol{\eta}$ and $\boldsymbol{\rho}$ represent the dual variables for the first and the second rows of Equation (3.53), respectively, we arrive at the following customization of

Theorem 3.2.2 to the considered dual problem:

**Corollary 3.2.6.** *The following LP is the dual of the LP formulated in Corollary 3.2.5:*

$$\max_{\boldsymbol{\eta},\boldsymbol{\rho}} \, (\boldsymbol{\beta}^1)^T \cdot \boldsymbol{\eta} \; + \; (\boldsymbol{\beta}^2)^T \cdot \boldsymbol{\rho} \tag{3.55}$$

*s.t.*

$$F^T \cdot \boldsymbol{\eta} \; - \; A^T \cdot \boldsymbol{\rho} \; \leq 0 \tag{3.56}$$

$$-1.0 \leq \rho_j \leq 1.0, \;\; \forall j \in \Omega(\boldsymbol{\pi}) \tag{3.57}$$

$$0.0 \leq \rho_j \leq 1.0, \;\; \forall j \in \Omega^c(\boldsymbol{\pi}) \equiv \{1, \ldots, m\} \setminus \Omega(\boldsymbol{\pi}) \tag{3.58}$$

$$\mathbf{1}_{\boldsymbol{\nu}}^T \cdot \boldsymbol{\rho} = 0. \tag{3.59}$$

*Hence, given the assumptions of Corollary 3.2.5 and the developments of Section 3.2.1, the LP formulation of Equations (3.55)–(3.59) will have an optimal value equal to zero* iff *the considered point $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ is an optimal solution for the dual problem of Equations (3.22) and (3.23).*

*Furthermore, if the optimal value of this dual LP formulation is positive, then the vector $\boldsymbol{\rho}^*$ corresponding to any optimal solution of this LP formulation defines a feasible direction of ascent for the formulation of Equations (3.22) and (3.23).*

**Proof:** As with Theorem 3.2.2, the first part of this corollary results immediately from Corollary 3.2.5 and LP strong duality.

To establish the second part of the corollary, we can first note that the feasibility of movement along $\boldsymbol{\rho}^*$ from the position $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ results from the stated assumptions and

Constraints (3.57)–(3.59). To show that $\boldsymbol{\rho}^*$ is a direction of ascent, however, we further need to show that

$$\forall \boldsymbol{g} \in \partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}), \;\; (\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g} > 0. \tag{3.60}$$

To this end, let us consider a subgradient $\boldsymbol{g} \in \partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$, and notice that, according to the previous discussion in this section, there exists a nonnegative vector $\boldsymbol{q}$ such that

$$\boldsymbol{g} \;=\; A \cdot \boldsymbol{q} \;+\; \boldsymbol{\beta}^2 \tag{3.61}$$

and

$$F \cdot \boldsymbol{q} \;=\; \boldsymbol{\beta}^1. \tag{3.62}$$

Then, for the considered subgradient vector $\boldsymbol{g}$,

$$\begin{aligned}
(\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g} \;&=\; (\boldsymbol{\rho}^*)^T \cdot (A \cdot \boldsymbol{q} \;+\; \boldsymbol{\beta}^2) \\
&=\; (\boldsymbol{\rho}^*)^T \cdot A \cdot \boldsymbol{q} \;+\; (\boldsymbol{\rho}^*)^T \cdot \boldsymbol{\beta}^2 \\
&=\; \boldsymbol{q}^T \cdot A^T \cdot \boldsymbol{\rho}^* \;+\; (\boldsymbol{\beta}^2)^T \cdot \boldsymbol{\rho}^*.
\end{aligned} \tag{3.63}$$

Since we have assumed that the value of the dual LP of Equations (3.55)–(3.59) is strictly positive, Equation (3.55) yields

$$\begin{aligned}
(\boldsymbol{\beta}^2)^T \cdot \boldsymbol{\rho}^* \;&>\; -(\boldsymbol{\beta}^1)^T \cdot \boldsymbol{\eta}^* \\
\text{(from Eq. (3.62))} \;&=\; -(F \cdot \boldsymbol{q})^T \cdot \boldsymbol{\eta}^* \\
&=\; -\boldsymbol{q}^T \cdot F^T \cdot \boldsymbol{\eta}^*.
\end{aligned} \tag{3.64}$$

Applying Equations (3.63), (3.64), and (3.56), and bearing in mind the non-negativity of the components of $\boldsymbol{q}$, we find

$$\begin{aligned}(\boldsymbol{\rho}^*)^T \cdot \boldsymbol{g} &> \boldsymbol{q}^T \cdot A^T \cdot \boldsymbol{\rho}^* \; - \; \boldsymbol{q}^T \cdot F^T \cdot \boldsymbol{\eta}^* \\ &= \boldsymbol{q}^T \cdot (A^T \cdot \boldsymbol{\rho}^* \; - \; F^T \cdot \boldsymbol{\eta}^*) \\ &\geq 0, \end{aligned} \tag{3.65}$$

from which Equation (3.60) immediately follows. □

Corollary 3.2.6 permits an implementation of the algorithm from Section 3.2.1 to address the dual problem described in Section 3.1 using distributed, efficient representations of $X$, $\tilde{X}$, $Conv(\tilde{X})$, and the subdifferential $\partial\theta(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$, as described above. The resulting algorithm also inherits the implementational features and the convergence properties that were developed in Section 3.2.1 for the more generic algorithm that was discussed in that section.

## 3.3 Optimizing the Lagrangian Dual Problem Using a Single LP

Using Equation (3.24), it is further possible to formulate the dual problem of Equations (3.22) and (3.23) as a single LP:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu},r} \; r \tag{3.66}$$

s.t.

$$\forall \boldsymbol{x} \in X, \;\; r \;\leq\; \boldsymbol{g}(\boldsymbol{x}) \cdot (\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})^T \tag{3.67}$$

$$\sum_{a \in \mathcal{A}} \nu_a = 1.0 \tag{3.68}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}; \;\; \boldsymbol{\mu} \geq \boldsymbol{0}; \;\; \boldsymbol{\nu} \geq \boldsymbol{0} \tag{3.69}$$

Standard LP duality theory ([35]) tells us that this linear program can be used to formulate a dual LP whose optimal objective value matches the optimal value of Equation 3.66. That LP can be represented as

$$\min_{\boldsymbol{\gamma},\boldsymbol{z},w} w \tag{3.70}$$

s.t.

$$[-I \ \mathbf{1}_{\boldsymbol{\nu}}] \cdot \begin{bmatrix} \boldsymbol{z} \\ w \end{bmatrix} = \sum_{\boldsymbol{x}\in X} \gamma_{\boldsymbol{x}} \cdot \boldsymbol{g}(\boldsymbol{x})^T \tag{3.71}$$

$$\sum_{\boldsymbol{x}\in X} \gamma_{\boldsymbol{x}} = 1.0 \tag{3.72}$$

$$\boldsymbol{\gamma} \geq \mathbf{0}; \ \ \boldsymbol{z} \geq \mathbf{0}. \tag{3.73}$$

In the above formulation, $\boldsymbol{\gamma}$ is a nonnegative vector that collects the dual variables corresponding to the constraint set described by Equation (3.67) in the primal problem, and $w$ is a free variable that corresponds to the dualization of Equation (3.68). Vector $\boldsymbol{z}$ is comprised of a set of "slack" variables that we used to formulate Equation (3.71) as an equality constraint. Also in Equation (3.71), $I$ represents an $m \times m$ identity matrix, where $m$ is equal to the total number of Lagrange multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$. Lastly, $\mathbf{1}_{\boldsymbol{\nu}}$ is an $m$-dimensional binary column vector whose unit elements are placed such that they correspond to the indices of the Lagrange multipliers $\boldsymbol{\nu}$ in the concatenated vector $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$.

Even though the formulations of Equations (3.66)–(3.69) and Equations (3.70)–(3.73) represent effectively the dual problem of Equations (3.22) and (3.23), these formulations suffer from a similar problem to the one that arose when we sought to apply the results of Section 3.2.1 to the dual problem from Section 3.1. Namely, $X$ does not typically lend itself to explicit enumeration, which restrains the explicit formulation of the constraint set (3.67), as well as the computation of the sum on the right-hand side of Equation (3.71). However, using similar representations to those discussed in Section 3.2.2, we can derive appropriate matrices $F$ and $A$, as well as vectors $\boldsymbol{q}$, $\boldsymbol{\beta}^1$, and $\boldsymbol{\beta}^2$ that allow us to reformulate the LP of Equations (3.70)–(3.73) as

$$\min_{\boldsymbol{q},\boldsymbol{y},w} \; w \tag{3.74}$$

s.t.

$$\begin{bmatrix} F & 0 & 0 \\ -A & -I_{m\times m} & \mathbf{1}_\nu \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{z} \\ w \end{bmatrix} = \begin{bmatrix} \boldsymbol{\beta}^1 \\ \boldsymbol{\beta}^2 \end{bmatrix} \tag{3.75}$$

$$\boldsymbol{q} \geq 0; \;\; \boldsymbol{z} \geq 0. \tag{3.76}$$

Furthermore, the formulation of Equations (3.74)–(3.76) can be dualized to obtain

$$\max_{\boldsymbol{\eta},\boldsymbol{\rho}} \; (\boldsymbol{\beta}^1)^T \cdot \boldsymbol{\eta} \; + \; (\boldsymbol{\beta}^2)^T \cdot \boldsymbol{\rho} \tag{3.77}$$

s.t.

$$F^T \cdot \boldsymbol{\eta} \; - \; A^T \cdot \boldsymbol{\rho} \; \leq 0 \tag{3.78}$$

$$\mathbf{1}_{\boldsymbol{\nu}}^T \cdot \boldsymbol{\rho} = 1.0 \tag{3.79}$$

$$\rho_j \geq 0, \;\; \forall j. \tag{3.80}$$

Because Equations (3.77)–(3.79) represent the dual program of Equations (3.74)–(3.76), which in turn is a reformulation of the LP of Equations (3.70)–(3.73), Equations (3.77)–(3.79) are an efficient representation of the LP from Equations (3.66)–(3.69). This implies the following theorem:

**Theorem 3.3.1.** *The LP of Equations (3.77)–(3.79) is an efficient representation of the LP* *of Equations (3.66)–(3.69). Its optimal objective value is equal to the optimal objective value of the Lagrangian dual problem of Equations (3.22) and (3.23). Furthermore, any vector $\boldsymbol{\rho}^*$ taken from an optimal solution $(\boldsymbol{\eta}^*, \boldsymbol{\rho}^*)$ of the problem of Equations (3.77)–(3.79) specifies an optimal set of Lagrange multipliers for Equations (3.22) and (3.23); that is, $\boldsymbol{\rho}^* = (\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$.*

The optimal values of the Lagrange multipliers can be useful as a set of heuristic "cost" values for a scheduling algorithm. In particular, the coefficients described by Equations (3.7)–(3.9) can be used to collect terms pertaining to $x_{a,e,t}$, $\forall a \in \mathcal{A}$, $\forall e \in E$, $\forall t \in \mathcal{T}$, which allows us to specify "costs" or "prices" for locating a specific agent in a particular place at a given time. One possible method for putting this information to use is described in Section 4.6.3 as a potential enhancement to the algorithm detailed in Section 4.4.

# CHAPTER 4
# A HEURISTIC APPROACH TO THE CONSIDERED TRAFFIC SCHEDULING PROBLEM

While Chapter 3 detailed methods for applying a Lagrangian relaxation to the considered MIP from Chapter 2, and while these methods yield, at a minimum, new ways to find lower bounds for the optimal makespan of instances of the considered problem, it remains for us to develop a scheduling algorithm aimed at producing fully feasible, optimized solutions for instances of the considered MIP. With this in mind, we present in this chapter an algorithm, based on local search and a novel use of DP, that builds and optimizes these schedules.

## 4.1 Formal Representation of the Solution Space

To proceed with the development of the sought algorithm, we first need to define and formalize a solution space for the MIP from Chapter 2; this definition will help us to motivate the presented optimization algorithm and the primary data structures that are employed by this algorithm.

Hence, let $T$ represent an upper bound for the optimal makespan, $w^*$. If we assume that a feasible solution exists for the given instance of the considered problem, then $T$ and $w^*$ are positive integers.[1] Furthermore, to specify a complete schedule for any finite $T$, all we need to do is to specify the edge, $e_t^a$, held by each agent $a \in \mathcal{A}$ at each period $t \in \{0, 1, \ldots, T\}$. In other words, a complete (traffic) schedule can be described as a set $\mathcal{S}$ of $|\mathcal{A}|$ finite sequences $\sigma^a$, each consisting of edges $e \in \hat{E} \equiv E \cup \{h\}$ and having length

[1]While the existence of a feasible solution is not always the case (and is definitely *not* the case when a system contains a deadlock or a subset of its agents that cannot avoid a deadlock), such feasible solutions *do* always exist under certain conditions. If, for instance, a depot (or "home" location, $h$) exists that can retain all agents and dispatch them in any order, and if agents can reverse direction in any travel location, as we assume for the quantum-computing context of the problem, then, as Proposition 4.2.2 establishes, a feasible solution is sure to exist. In contexts where these assumptions do not necessarily apply, it is necessary to ensure that a feasible solution always exists from a certain system state before making the decision to enter that state.

$T + 1$.

It should be noted, however, that a *complete* schedule is not necessarily a *feasible* schedule. Based on the MIP in Chapter 2, we define a given schedule to be feasible *iff* it satisfies the following conditions:

1. $\forall a \in \mathcal{A},\ e_0^a = s_a \ \wedge\ e_T^a = d_a$.

2. $\forall a \in \mathcal{A},\ \forall t \in \{0, 1, \ldots, T-1\},\ e_{t+1}^a \in \{e_t^a\} \cup NH(e_t^a)$, where $NH(e_t^a)$ denotes the set of the neighboring edges of edge $e_t^a$ in graph $G$.

3. $\forall a, a' \in \mathcal{A},\ \forall t \in \{1, \ldots, T\},\ e_t^a = e \ \wedge\ e_t^{a'} = e' \Longrightarrow (e \neq e') \vee (e = e' = h) \vee (e = e' = e_q^a = e_q^{a'},\ \forall q \in \{0, \ldots, t\}) \vee (e = d_a \wedge e' = d_{a'})$.

4. $\forall a, a' \in \mathcal{A},\ \forall t \in \{1, \ldots, T\},\ e_t^a = e_{t-1}^{a'} = e \neq e_{t-1}^a \Longrightarrow (e = h) \vee (d_a = d_{a'} = e)$.

The first condition above ensures that an agent starts at its origin, $s_a$, and that it reaches its destination $d_a$ within the allotted time horizon, $T$. Condition 2 states that every agent route is subject to the connectivity of the underlying guidepath network. Condition 3 prevents agents from cohabiting in a given location at any time, with the following exceptions: (i) The location is the "home" edge $h$; (ii) The location is a shared source; or (iii) The location is a shared destination. Lastly, Condition 4 enforces that an agent $a$ can move onto an edge $e$ only if that edge was unoccupied at the previous time step, unless $e$ is a common destination or is the home edge.

The presented algorithm also requires an efficient representation of the set of routes that can transport any agent, $a \in \mathcal{A}$, from its location at period $t \in \{0, 1, \ldots, T-1\}$, represented as $e_t^a = e \in \hat{E}$, to its destination, $d_a$, within the remaining time interval $\{t+1, \ldots, T\}$. A directed acyclic graph (DAG), $\mathcal{D}(a, e, t, T)$, can be designed to serve this purpose. The nodes of $\mathcal{D}(a, e, t, T)$ represent unique time-location pairs, $(e', t')$, for certain $e' \in \hat{E}$ and $t' \in \{t, \ldots, T\}$. These pairs correspond to the location of agent $a$ at edge $e'$ at time $t'$, and any path in $\mathcal{D}(a, e, t, T)$ that passes through node $(e', t')$ corresponds to a solution of the

1. Start by introducing to the constructed DAG $\mathcal{D}(a, e, t, T)$ the "root" node $(e, t)$, indicating the positioning of agent $a$ at edge $e$ at time $t$.
2. For each new node introduced in the constructed DAG $\mathcal{D}(a, e, t, T)$, append a next layer of nodes $(e', t+1)$, for all those edges $e'$ that (i) belong to the set $\{e\} \cup NH(e)$, and (ii) there exists a feasible route that can take agent $a$ from edge $e'$ to its destination in no more than $T - t$ time periods; link each node $(e', t+1)$ to node $(e, t)$ with a directed edge leading from the latter node to the former; furthermore, in this expansion, avoid the re-introduction of nodes that have been already introduced in the constructed DAG $\mathcal{D}(a, e, t, T)$.
3. Terminate when all nodes of DAG $\mathcal{D}(a, e, t, T)$ have been processed according to the logic of Step 2.

Figure 4.1: A "forward-search" algorithm for constructing DAG $\mathcal{D}(a, e, t, T)$.

original MIP from Chapter 2 with $x_{a,e',t'} = 1$. Placement of a directed edge between two nodes requires accessibility of the "sink" node, $(e'', t'+1)$, from the "source" node, $(e', t')$, in one time step, and implies that one or more feasible routes exist that contain the move from $(e', t')$ to $(e'', t'+1)$.

Construction of DAG $\mathcal{D}(a, e, t, T)$ can be accomplished using a "forward-search" algorithm, as depicted in Figure 4.1. The second condition in the second step of Figure 4.1 (i.e., that inclusion of $(e', t')$ in $\mathcal{D}(a, e, t, T)$ requires that a route must exist that transports agent $a$ from edge $e'$ to edge $d_a$ within $T - t$ time steps) can be assessed offline (i.e., before the algorithm execution) using the Floyd-Warshall algorithm [2] to compute all pairwise distances between pairs of locations on the guidepath graph, and to store those numbers in a lookup table.

The number of nodes in DAG $\mathcal{D}(a, e, t, T)$ is bounded from above by $|\hat{E}| \cdot T$, and in practice this number will tend to be much smaller since achievement of this bound is based on a guidepath graph, $G$, that is complete. Figure 4.2 shows what a typical DAG $\mathcal{D}(a, s_a, 0, T)$ might look like.

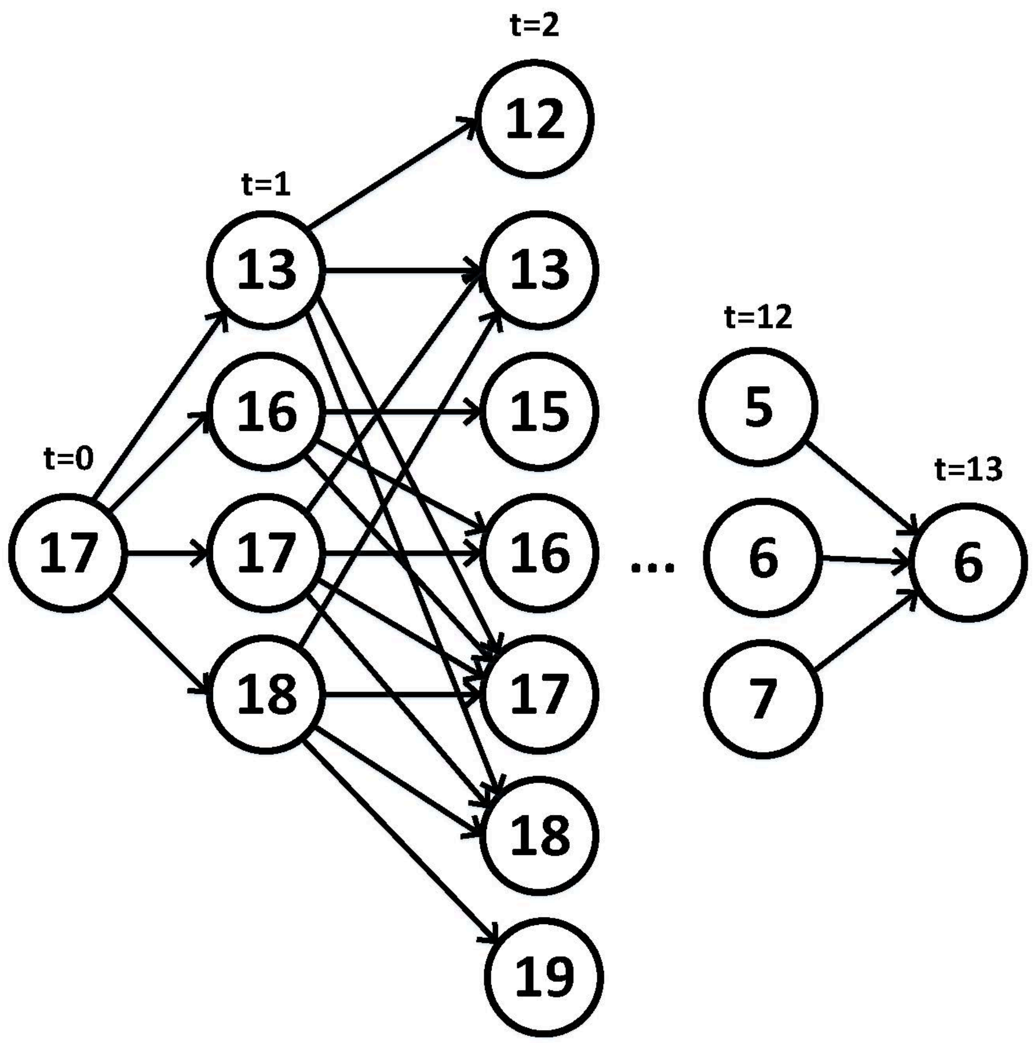

Figure 4.2: Example of a spatio-temporal DAG for agent $a$, given $t = 0$ and $T = 13$, $\mathcal{D}(a, s_a, 0, 13)$. This graph describes all possible routes taking agent $a$ from its current location, $s_a$, to its destination location, $d_a$, within the time horizon $T$, while observing the corresponding single-agent constraints.

## 4.2 Obtaining an Initial Feasible Solution

The third and fourth conditions used to define feasible schedules in the previous section can be used to define "routing conflict" among different agents. These conditions, combined with the arbitrary topology of the guidepath graph $G$, can make the question of feasibility of some traffic coordination problems of the considered class very difficult to answer [46].

The methodology that is presented in this chapter requires the construction of an initial feasible schedule for the traveling agents. Importantly, some contexts, such as the quantum-computing context, do permit efficient algorithms for finding feasible solutions for all instances of the problem. In the quantum-computing case, this universal feasibility arises because of (i) the existence of a "home" edge $h$ that has sufficient capacity to retain all agents $a \in \mathcal{A}$ and has the ability to receive and dispatch those agents in arbitrary order, and (ii) the reversibility of the agent motion on any edge $e \in E$ of the guidepath network. These two features enable the specification of a two-stage algorithm that first routes all agents to the home edge $h$ and then routes all agents to their respective destinations, $d_a$. The precise details of this algorithm are described by the following lemma and proposition.

**Lemma 4.2.1.** *For instances of the traffic coordination problem defined in Chapter 2 that guarantee (i) the existence of a "home" edge $h$ that has sufficient capacity to retain all agents $a \in \mathcal{A}$ and has the ability to receive and dispatch those agents in arbitrary order, and (ii) the reversibility of the agent motion on any edge of the underlying guidepath network, it is always possible to reach from the initial traffic state $\mathrm{s}_0$ that is defined by the edge set $\{s_a : a \in \mathcal{A}\}$, to the traffic state $\mathrm{s}_h$ where every agent is located at the "home" edge $h$.*

*Proof:* If $s_a = h, \forall a \in \mathcal{A}$, then the lemma follows trivially. Otherwise, we can select any agent, $a_1$, from the set of agents $a$ with $s_a \neq h$, that is located closest to the "home" edge $h$, where proximity is measured as the smallest number of edges that must be traversed in order to reach $h$ from $s_{a_1}$. Clearly, this selection of agent $a_1$ implies that all edges on any shortest path leading from its current location $s_{a_1}$ to the "home" edge $h$ are free. Hence, agent $a_1$

can reach edge $h$ while keeping all other agents in their initial positions. Lemma 4.2.1 then follows by induction from the above argument for the remaining set of agents $a$ with $s_a \neq h$. $\square$

**Proposition 4.2.2.** *For instances of the traffic coordination problem defined in Chapter 2 that guarantee (i) the existence of a "home" edge $h$ that has sufficient capacity to retain all agents $a \in \mathcal{A}$ and has the ability to receive and dispatch those agents in arbitrary order, and (ii) the reversibility of the agent motion on any edge of the underlying guidepath network, a routing schedule always exists that transports all agents $a \in \mathcal{A}$ from their initial locations $s_a$ to their respective destinations $d_a$, and that abides by the Conditions 1–4 of Section 4.1.*

*Proof:* Lemma 4.2.1 established that it is possible to reach from the initial state $\mathrm{s}_0$ to the state $\mathrm{s}_h$ where all agents are collected on the "home" edge $h$. An argument similar to that used in the proof of Lemma 4.2.1 can be used to establish that, from state $\mathrm{s}_h$, all agents $a \in \mathcal{A}$ can be routed to their destinations $d_a$ one at a time, starting with the agents for which the destination edge $d_a$ is furthest from the "home" edge $h$. $\square$

Proposition 4.2.2 establishes the feasibility of traffic coordination problems in the quantum-computing context, as well as in other contexts that share similar assumptions. While the methods described in Lemma 4.2.1 and in Proposition 4.2.2 establish the existence of feasible solutions under the given assumptions, the described solutions will tend to produce feasible solutions with unnecessarily long makespans. Under the given conditions, it is possible to generate more efficient feasible solutions by using the remaining travel distance required to route an agent to the home edge, or to its destination, as a mechanism to prioritize the movement of each agent and then produce a schedule that moves each agent simultaneously along its shortest path, as long as this movement does not result in a conflict with a higher-priority agent. This scheme can be implemented for both phases of the proposed algorithm–that of transporting the agents to the home edge, and also the second phase that transports the agents from the home edge to their respective destinations. Building an efficient initial feasible solution in this way serves two purposes: (i) upper-bounding the makespan of the sought schedule, and (ii) moderating the size of the spatio-temporal DAGs used in the implementation of the optimization algorithm described later in the text.

From a broader standpoint, Proposition 4.2.2 makes it very easy for us to guarantee the overall traffic liveness in the corresponding contexts. As noted above, however, Proposition 4.2.2 is restricted by certain conditions. Though many other settings will have a home edge, they may not satisfy the "reversibility" requirement. When this is the case, the question of ensuring liveness is much more difficult but can be addressed using adaptations of Dijkstra's Banker's algorithm [13], similar to those discussed in [47].

Lastly, it should be noted that one of the merits of the iterative makespan-reduction procedure described in the following section is that it is independent from the method used to generate initial feasible solutions. Though the technique described in this section for generating initial solutions is subject to the two assumptions described in Proposition 4.2.2 (assumptions that hold true for our test cases), the algorithm described below can be implemented even when these assumptions do not hold, as long as some initial feasible solution is readily obtainable.

## 4.3 Local Search for Improved Solutions

The considered algorithm uses an initial feasible solution[2] as a "seed" for generating further solutions with an improved makespan. Let $\mathcal{S} = \{\sigma^a : a \in \mathcal{A}\}$ represent a feasible schedule with makespan $w$. There must exist a set of agents $\dot{\mathcal{A}} \subseteq \mathcal{A}$ that define the makespan (i.e., that reach their destinations exactly at period $w$). Schedule $\mathcal{S}$ can potentially be improved by prescribing a new route, $\hat{\sigma}^{\hat{a}}$, for an agent $\hat{a} \in \dot{\mathcal{A}}$ that transports agent $\hat{a}$ to its destination $d_{\hat{a}}$ at a time earlier than $w$ while avoiding any conflict with the routes $\sigma^a$ specified by $\mathcal{S}$ for any other agent, $a \in \mathcal{A} \setminus \{\hat{a}\}$. When such an operation is performed and the original schedule $\mathcal{S}$ is replaced by $\hat{\mathcal{S}} \equiv \{\sigma^a : a \in \mathcal{A} \setminus \{\hat{a}\}\} \cup \{\hat{\sigma}^{\hat{a}}\}$, then the latter is said to be an "improving" schedule w.r.t. $\mathcal{S}$. More importantly, accomplishing such an operation for every

[2]possibly, but not necessarily, provided by the techniques that were described in the previous section

agent $a \in \dot{\mathcal{A}}$ effectively defines an improving schedule with a new, reduced makespan.

In order to specify the improving step fully, it remains for us to detail a mechanism for finding a conflict-free route, $\hat{\sigma}^{\hat{a}}$, for the agent $\hat{a}$. This can be accomplished by formulating a "shortest-path" problem and solving it using DP methods [2] on the DAG $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w-1)$ that was defined in Section 4.1. To see how this DP approach will work, let us first remind the reader that DAG $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w-1)$ represents *all* routes that transport agent $\hat{a}$ from its initial location $s_{\hat{a}}$ to its destination $d_{\hat{a}}$ in $w-1$ or fewer time steps. We also recall that nodes in the spatio-temporal DAG $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w-1)$ represent pairings $(e, t)$ that determine the location $e$ of agent $\hat{a}$ at time $t$. If, for each of these nodes $(e, t)$, we count the number of conflicts that are incurred w.r.t. the other routes $\sigma^a,\ a \in \mathcal{A} \setminus \{\hat{a}\}$ in the current schedule when agent $\hat{a}$ is placed at location $e$ at time $t$, then the numbers of these conflicts can be used to define a notion of "cost" for the corresponding placements, and a route $\hat{\sigma}^{\hat{a}}$ is feasible w.r.t. Conditions 1–4 in Section 4.1 *iff* it is represented by a path leading from the "root" node $(s_{\hat{a}}, 0)$ of DAG $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w-1)$ to its "terminal" node $(d_{\hat{a}}, w-1)$ with a total cost of zero. Using this fact, the method described here selects a new route $\hat{\sigma}^{\hat{a}}$ from the zero-cost paths identified as solutions to the aforementioned "shortest-path" problem formulated on $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w-1)$.[3]

## 4.4 A Complete and Canonical Description of the Presented Scheduling Algorithm

A complete canonical description of the algorithm described in this chapter can be summarized as follows: First, the algorithm constructs an initial feasible schedule, $\mathcal{S}$, as detailed in Section 4.2. Next, the algorithm switches to a makespan-reduction mode, where it iteratively seeks to generate improving schedules, as described in Section 4.3; at the $i$-th iteration, the algorithm will seek to improve the schedule $\mathcal{S}^{(i-1)}$ obtained during the previous iteration

[3]A streamlined version of this algorithm can consider a "reachability" problem rather than a "shortest-path" problem. Reachability can be determined by considering only the subgraph of $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w-1)$ defined by the $(e, t)$ pairings that incur zero cost. This version of the problem lends itself to enumerative techniques [7] that may save time when compared with "shortest-path" methods. However, the description here uses "shortest-path" computations because, as we will see in Section 4.6.1, this method provides some additional information that enables an important algorithmic enhancement.

```
1. Use the procedure outlined in Section 4.2 in order to construct an initial feasible
   schedule $\mathcal{S}^{(0)}$ for the considered problem instance.
2. $\mathcal{S} := \mathcal{S}^{(0)}$.
3. $Q := \dot{\mathcal{A}}$.
4. While ($Q \neq \emptyset$) do
   (a) Pick an element $a \in Q$.
   (b) Test whether agent $a$ can be used for generating an improving schedule $\hat{\mathcal{S}}$.
   (c) If the above test is positive, do
       i. $\mathcal{S} := \hat{\mathcal{S}}$.
       ii. Goto Step 3.
   (d) else $Q := Q \setminus \{a\}$.
5. Return $\mathcal{S}$.
```

Figure 4.3: A basic version of the heuristic algorithm for the traffic coordination problem that is considered in this work.

(where $\mathcal{S}^{(0)} = \mathcal{S}$), having found an agent $a^{(i)} \in \dot{\mathcal{A}}^{(i-1)}$ that is useful for constructing this improving schedule. Once an improving schedule, $\hat{\mathcal{S}}$, is obtained, then the incumbent schedule, $\mathcal{S}^{(i)}$, is set to equal $\hat{\mathcal{S}}$, and the algorithm progresses to the next iteration, $i + 1$. The algorithm will terminate at the first iteration, $k$, for which it is unable to find an improving solution for any agent $a \in \dot{\mathcal{A}}^{(k-1)}$. The algorithm will then return, as its output, the most recent feasible schedule, $\mathcal{S}^{(k-1)}$, with a makespan of $w^{(k-1)}$.

Pseudocode for the algorithm described above is given in Figure 4.3. The completeness and soundness of this algorithm follow from the results presented in the prior sections of this chapter. Furthermore, since each iteration of Steps 3–4 requires that an agent will reach its destination at least one time period earlier than before, the returned solution must have a makespan that is no worse than that of the initial schedule, $\mathcal{S}^{(0)}$, and the algorithm must terminate in a finite number of iterations. Next we take a closer look at the complexity of the algorithm described in Figure 4.3.

## 4.5 Complexity Analysis for the Algorithm of Figure 4.3

Construction of, both, the initial feasible schedule, $\mathcal{S}^{(0)}$, and the spatio-temporal DAGs that are employed in the improving phase, benefits from the existence of a lookup table of pairwise shortest distances for all locations in $\hat{E}$. This table can be obtained through a single execution of the Floyd-Warshall algorithm, which has a computational complexity of $O(|\hat{E}|^3)$ [2].

Next, focusing on the iterative component of the algorithm described in Figure 4.3, we establish the following result:

**Proposition 4.5.1.** *Let $w^*$ denote the optimal makespan for the considered problem instance, and $\hat{w}$ denote the makespan of the initial feasible schedule $\mathcal{S}^{(0)}$. The worst-case computational complexity of the iterative part of the algorithm described in Figure 4.3 is $O((\hat{w}^2 - w^{*2})|\mathcal{A}|^2|\hat{E}|^2)$.*

*Proof:* First consider the test that is performed in Step (4b) of the algorithm in Figure 4.3. As remarked in Section 4.3, this test can be organized efficiently as a "reachability" test that seeks the existence of a zero-cost path in DAG $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w^* + i)$, leading from node $(s_{\hat{a}}, 0)$ to node $(d_{\hat{a}}, w^*+i)$ for some $\hat{a} \in \mathcal{A}$ and $i \in \{0, \ldots, \hat{w}-1-w^*\}$. DAG $\mathcal{D}(\hat{a}, s_{\hat{a}}, 0, w^*+i)$ will have $O(|\hat{E}|(w^* + i))$ nodes and each node will have $O(|\hat{E}|)$ emanating edges. Determining the availability of any of these edges in the underlying reachability problem requires the evaluation of conflict between the corresponding step and the incumbent routes of the remaining agents, a task that carries a cost of $O(|\mathcal{A}|)$. Therefore, the total cost for a single execution of the considered test is $O(|\mathcal{A}||\hat{E}|^2(w^* + i))$. According to the "while"-loop of Step (4), at any given $i$, this test will be performed $O(|\mathcal{A}|)$ times. Finally, the result of Proposition 4.5.1 is obtained by further noticing that $\sum_{i=0}^{\hat{w}-1-w^*}(w^* + i) = O(\hat{w}^2 - w^{*2})$. $\square$

Proposition 4.5.1 implies a pseudo-polynomial worst-case computational complexity for the algorithm described in Figure 4.3, and quantifies the benefit of finding an efficient initial schedule. Chapter 5 includes execution-time data that show how this complexity is exhibited empirically and demonstrate that the algorithm can solve large problem instances very quickly while using modest computational resources. Furthermore, the next section shows that the performance of the presented algorithm can be further enhanced through a pertinent use of the information that is contained in the “shortest-path” implementations described in Section 4.3.

## 4.6 Enhancing the Algorithm Performance

The algorithm detailed in Figure 4.3 describes a complete heuristic solution for the multi-agent routing problem described in Chapter 2, but in this section we consider possible enhancements and variations of this algorithm.

### 4.6.1 Controlled Excursions into the Infeasible Solution Space

As a first step, it can be noted that, as with any algorithm that depends on local search methods, the solution can settle on a “local optimum” while better schedules still exist. In addition, the concept of the “local optimum” is affected by (i) our representation of the solution space, and (ii) the “neighborhood” that the algorithm is designed to consider. With this in mind, we will devote the remainder of this subsection to the discussion of a method that allows the algorithm to make controlled excursions outside the region of feasible solutions with the intent of escaping some of these local optima.

This method works by executing the algorithm of Figure 4.3 as prescribed in Section 4.4 until an iteration $k$ is reached where no improving schedule is identified (i.e., until no zero-conflict route exists for any agent that defines the makespan of the most recent feasible solution $\mathcal{S}^{(k-1)}$ that transports the agent to its destination in a reduced span of time). At this point, the revised algorithm switches to a different mode that (i) picks an improved but infeasible schedule, and then (ii) seeks to eliminate conflicts appearing in this schedule iteratively until none exist, if possible. In particular, we execute the following steps when no improving schedule is identified:

Among the paths of the DAGs $\mathcal{D}(a, s_a, 0, (w-1)^{(k-1)}),\ a \in \dot{A}^{(k-1)}$, constructed in iteration $k$, we identify one that incurs minimal cost w.r.t. the number of inter-agent conflicts for the fixed routes of schedule $\mathcal{S}^{(k-1)}$. If $a_1$ represents the corresponding agent and $\sigma_1^{a_1}$ represents the corresponding path, then let us use $\mathcal{S}_1^{(k-1)}$ to represent the routing schedule that results when the route of agent $a_1$ is replaced in schedule $\mathcal{S}^{(k-1)}$ by route $\sigma_1^{a_1}$. Then we can attempt to find a new feasible routing schedule by incrementally eliminating the conflicts that exist in $\mathcal{S}_1^{(k-1)}$.

This incremental improvement begins by finding an agent $a_2 \in A \setminus \{a_1\}$ that has a minimal-cost path in the DAG $\mathcal{D}(a_2, s_{a_2}, 0, w^{(k-1)})$ whose total cost is lower than the number of conflicts between agent $a_2$ and all other agents in the schedule $\mathcal{S}_1^{(k-1)}$. If $\sigma_2^{a_2}$ represents the corresponding minimal-cost path, then replacing the original route of agent $a_2$ in schedule $\mathcal{S}_1^{(k-1)}$ with $\sigma_2^{a_2}$ yields a new schedule, $\mathcal{S}_2^{(k-1)}$, that must include a smaller number of inter-agent routing conflicts than $\mathcal{S}_1^{(k-1)}$. Repeating this procedure, then, will result in either a schedule $\mathcal{S}_n^{(k-1)}$ with zero conflicts, or a schedule where further reduction of the number of conflicts is not possible. When a conflict-free schedule is found, then $\mathcal{S}_n^{(k-1)}$ can take the place of schedule $\mathcal{S}^{(k)}$ in the main iteration of the revised algorithm, and the algorithm can progress to its next iteration, $k+1$, in accordance with Figure 4.3. When a conflict-free solution is not found, then the algorithm will terminate and return the most recent conflict-free solution encountered–i.e., $\mathcal{S}^{(k-1)}$.

Using this procedure, it is possible to create a much more powerful implementation of the proposed algorithm without substantially increasing its computational complexity. Some results of the application of this enhancement are described in Chapter 5.

### 4.6.2 Multi-Agent Simultaneous Search

Another potential enhancement of the methodology described above involves increasing the size of the neighborhood searched. One way to accomplish this is by using DAGs whose nodes are characterized by multi-agent states. For the 2-agent case, this idea can be implemented as follows:

First, an implementation of the algorithm described in Figure 4.3 can be run, and this implementation can be enhanced using the digressions from the feasible region described in Section 4.6.1. Once it happens that the algorithm is unable to find an improving solution, even after allowing for a controlled excursion outside the set of feasible schedules, pairs of agents $(a_1, a_2)$ can be chosen, and a new spatio-temporal DAG $\mathcal{D}^2(a_1, a_2, s_{a_1}, s_{a_2}, 0, \tau - 1)$ can be built for each pair $(a_1, a_2)$. Here, $\tau$ represents the makespan of the most recent feasible schedule, and the nodes $(e, t)$ from the original DAG structure $\mathcal{D}(a, s_a, 0, \tau - 1)$ are replaced by nodes $(e_1, e_2, t)$, such that the positions of agents $a_1$ and $a_2$ are jointly described at time $t$.

This method requires some intricate computation, since (i) the construction of multi-agent spatio-temporal DAGs requires that we identify and eliminate internal conflicts between agents $a_1$ and $a_2$, and (ii) we cannot use something as simple as a lookup table generated by the Floyd-Warshall algorithm to determine whether a considered node $(e_1, e_2, t)$ generated using a "forward-search" algorithm will maintain the feasibility of transporting both $a_1$ and $a_2$ to their destinations $d_{a_1}$ and $d_{a_1}$ within $T - t$ time steps. On the other hand, the approach to be developed can be applied to various (limited) numbers of agents, to define ever-more-powerful (but computationally intensive) variations of the considered algorithm.

*Building Multi-Agent Spatio-Temporal Graphs*

One method for building a 2-agent spatio-temporal DAG $\mathcal{D}^2(a_1, a_2, s_{a_1}, s_{a_2}, 0, T)$ for a given time horizon, $T$, can be outlined broadly as follows: (i) Generate spatio-temporal DAGs $\mathcal{D}(a_1, s_{a_1}, 0, T)$ and $\mathcal{D}(a_2, s_{a_2}, 0, T)$, as shown in Figures 4.1 and 4.2, for the selected agents. (ii) Merge the two graphs, eliminating states and moves that cause conflict between the two agents.

The merger of these two graphs can be accomplished using the following steps: (i) Select an intermediate time step between 0 and $T$–e.g., $floor(T/2)$. Here, we will refer to this time step as $\hat{\tau}$. (ii) From time step 0, execute a "forward-search" algorithm, based on the two graphs being merged, to identify possible joint states (and transitions) that may be included in the graph (details of this search process are described in Figure 4.4); repeat this search until all possible joint states that can be reached from the joint state $(s_{a_1}, s_{a_2}, 0)$[4] in $\hat{\tau}$ time steps have been identified. (iii) Perform a similar operation in reverse; i.e., define and run a "backward-search" algorithm from time $T$ to identify joint states from which the joint destination $(d_{a_1}, d_{a_2}, T)$ can be reached in $T - \hat{\tau}$ time steps. (iv) At this point, we have two sets of joint states corresponding to time $\hat{\tau}$–one set of joint states that are accessible from $(s_{a_1}, s_{a_2}, 0)$ (which we shall denote as set $K_{s,\hat{\tau}}$), and one set of joint states from which the destination state $(d_{a_1}, d_{a_2}, T)$ can be reached (which we shall denote as set $K_{d,\hat{\tau}}$). We also have two graphs identifying the paths by which these states can be reached. We can identify the set for which both accessibility conditions are met by taking the intersection $K_{s,\hat{\tau}} \cap K_{d,\hat{\tau}}$. We will refer to this set as $K_{\hat{\tau}}$. (v) From $K_{\hat{\tau}}$, we can identify $K_t$ (i.e., the set of states at time $t$ that satisfy both accessibility conditions) for every $0 < t < \hat{\tau}$ (as well as the feasible joint moves between members of $K_t$ and $K_{t+1}$ for $0 < t < \hat{\tau}$) using a recursive method to identify the "parents" and "ancestors" of each member of $K_{\hat{\tau}}$ among $K_{s,\hat{\tau}-1}$ to establish $K_{\hat{\tau}-1}$, and we repeat this procedure all the way back to $K_1$. (vi) A similar recursion can be used to establish $K_t$ for $\hat{\tau} < t < T$ by identifying the "children" and "descendents" of members of $K_{\hat{\tau}}$. Once this process is complete, the graph composed of $(s_{a_1}, s_{a_2}, 0)$, $(d_{a_1}, d_{a_2}, T)$, $K_t$ for every $0 < t < T$, and the feasible transitions between the included states $(e_1, e_2, t)$ is $\mathcal{D}^2(a_1, a_2, s_{a_1}, s_{a_2}, 0, T)$.[5]

This process effectively describes how to merge two single-agent spatio-temporal graphs together in order to build a single, "compound" graph that describes all possible 2-agent schedules that can be achieved for a particular pair of agents within time $T$, given that no

[4]We use the notation $(e_1, e_2, t)$ to represent the "joint state" where agents $a_1$ and $a_2$ are located on edges $e_1$ and $e_2$ (respectively) at time $t$.

[5]Note that if $T$ is too low–i.e., if the optimal makespan for the two agents, $w^*$, is greater than $T$–then the resulting graph will be empty since no state $(e_1, e_2, t)$ exists that is reachable from state $(s_{a_1}, s_{a_2}, 0)$ in $t$ time steps and from which state $(d_{a_1}, d_{a_2}, T)$ is reachable in $T - t$ time steps, for any $0 \leq t \leq T$. This remark also holds when the procedure is generalized to accommodate more than two agents.

1. Let $(e_1, e_2, t)$ denote the state of the "current" node being expanded. Let $(e_1, t)^\bullet$ and $(e_2, t)^\bullet$ indicate the children of states $(e_1, t)$ and $(e_2, t)$ in the single-agent DAGs $\mathcal{D}(a_1, s_{a_1}, 0, T)$ and $\mathcal{D}(a_2, s_{a_2}, 0, T)$. Enumerate a set of candidate joint states $\mathcal{M}_{(e_1,e_2,t)} = (e_1, t)^\bullet \times (e_2, t)^\bullet$. Set $\hat{\mathcal{N}}_0 := (s_{a_1}, s_{a_2}, 0)$, and let $\hat{\mathcal{N}}_{t+1}$ indicate the set of nodes added to the graph by expansions of $\hat{\mathcal{N}}_t$. For any element $(e'_1, e'_2, t+1) \in \hat{\mathcal{N}}_{t+1}$, let $^\bullet(e_1, e_2, t+1)$ indicate the set of nodes in $\hat{\mathcal{N}}_t$ whose expansions add $(e_1, e_2, t+1)$ to $\hat{\mathcal{N}}_{t+1}$.

2. While $t \leq \hat{\tau}$, do

   For each element $(e'_1, e'_2, t+1)$ of $\mathcal{M}_{(e_1,e_2,t)}$ do

   (a) Test whether $(e'_1, t+1)$ conflicts with $(e'_2, t+1)$.
   (b) Test whether $(e'_1, t+1)$ conflicts with $(e'_2, t)$.
   (c) Test whether $(e'_2, t+1)$ conflicts with $(e'_1, t)$.
   (d) If any conflict test is positive, break.
   (e) Else, test whether $(e'_1, e'_2, t+1) \in \hat{\mathcal{N}}_{t+1}$.
   (f) If all conflict tests are negative and $(e'_1, e'_2, t+1) \in \hat{\mathcal{N}}_{t+1}$, $^\bullet(e'_1, e'_2, t+1) := {}^\bullet(e'_1, e'_2, t+1) \cup (e_1, e_2, t)$.
   (g) If all conflict tests are negative and $(e'_1, e'_2, t+1) \notin \hat{\mathcal{N}}_{t+1}$, do
       i. $\hat{\mathcal{N}}_{t+1} := \hat{\mathcal{N}}_{t+1} \cup (e'_1, e'_2, t+1)$.
       ii. $^\bullet(e'_1, e'_2, t+1) := {}^\bullet(e'_1, e'_2, t+1) \cup (e_1, e_2, t)$

3. For each $t$ such that $0 < t \leq \hat{\tau}$, do

   (a) Return $\hat{\mathcal{N}}_t$.
   (b) For each $(e_1, e_2, t) \in \hat{\mathcal{N}}_t$, return $^\bullet(e_1, e_2, t+1)$.

Figure 4.4: Details of the "forward-search" node generation process. A similar, "backward-search" procedure is also performed from time $T$ to time $\hat{\tau}$.

other agents exist in the graph. This procedure can be straightforwardly generalized to any number of agents.

One method to accomplish an $n$-agent compound spatio-temporal graph describing schedules for agents $\{a_1, a_2, ...a_n\}$ is to merge the "simple" (i.e., single-agent) spatio-temporal graphs for $a_1$ and $a_2$, then merge the resulting compound graph with the simple graph for $a_3$, and so on until the simple graph for $a_n$ is merged with the compound graph for $\{a_1, a_2, ...a_{n-1}\}$.

Another method to accomplish this is with successive pairings of graphs. That is, one could merge the simple graphs for $a_1$ and $a_2$, then merge the graphs for $a_3$ and $a_4$, and repeat until no simple graphs remained. Then one could merge the compound graph for $a_1$ and $a_2$ with the graph for $a_3$ and $a_4$, merge the compound graph for $a_5$ and $a_6$ with the graph for $a_7$ and $a_8$, and so on. Eventually, one would be left with a single graph describing all feasible $n$-agent schedules that can be executed within $T$ time steps.

It should be noted that, as long as a feasible schedule exists requiring fewer than $T$ time steps to route a set of $n$ agents to their respective destinations, the aforementioned construction of an $n$-agent graph with a time horizon of $T$ can be used to identify the *optimal* makespan of the $n$-agent routing problem, as well as the set of all optimal schedules. In particular, the minimum makespan corresponds to the earliest time step $\tau$ for which the state $(d_{a_1}, d_{a_2}, ..., d_{a_n}, \tau)$ is included in the compound graph. In practice, this minimum can be ascertained quickly by starting with the terminal node (i.e., the node corresponding to the state $(d_{a_1}, d_{a_2}, ..., d_{a_n}, T)$) and checking the set of that node's parents to determine whether $(d_{a_1}, d_{a_2}, ..., d_{a_n}, T-1)$ exists among them. If so, then the optimal makespan, $w^*$, is less than or equal to $T-1$. If not, then $w^* = T$. This process can be repeated until the precise value of $w^*$ is found. Then, if we wish to identify optimal schedules, we need only "back-track" from the spatio-temporal node corresponding to the state $(d_{a_1}, d_{a_2}, ..., d_{a_n}, w^*)$ to the "start" node, $(s_{a_1}, s_{a_2}, ..., s_{a_n}, 0)$. Moreover, the subgraph describing *all* optimal $n$-agent schedules can be constructed by starting with the node corresponding to the state $(d_{a_1}, d_{a_2}, ..., d_{a_n}, w^*)$, by including all of that node's parent nodes, and by repeating this process for $w^*$ steps until the start node is reached.[6]

[6]Since, as noted previously, the procedure described here for building multi-agent spatio-temporal graphs will produce an empty graph if $T < w^*$, this procedure can also be used to determine whether solutions exist with makespan less than or equal to $T$, for any positive $T$.

*Complexity of Multi-Agent Spatio-Temporal Graphs*

Though the process described above for building and utilizing multi-agent spatio-temporal graphs may be relatively efficient, it is not necessarily practical to execute for large numbers of agents. This section endeavors to demonstrate the following: In the worst case, the multi-agent optimal-solution algorithm from the previous section will identify optimal solutions in time that scales polynomially with respect to $|\hat{E}|$ and pseudo-polynomially with respect to $T$, with the order of the polynomial determined by the number of agents in the system.

**Proposition 4.6.1.** *Given (i) a problem instance with a set of $|\mathcal{A}|$ agents and a guidepath graph with $|\hat{E}|$ edges, and (ii) an upper bound $T$ such that $T \geq w^*$, the problem can be solved with a worst-case time complexity of $O(|\mathcal{A}|^2 T |\hat{E}|^{2|\mathcal{A}|})$. On the other hand, if a time horizon $T$ is specified such that $T < w^*$, it is possible, in the worst case, to determine that no solution exists with makespan less than or equal to $T$ in $O(|\mathcal{A}|^2 T |\hat{E}|^{2|\mathcal{A}|})$ time.*

*Proof:* Development of the 2-agent spatio-temporal graph can be simplified as follows: For $0 \leq t < T$, up to $|\hat{E}|^2$ nodes corresponding to states $(e_1, e_2, t)$ are checked for connectivity with up to $|\hat{E}|^2$ nodes corresponding to states $(e_1', e_2', t+1)$ in the succeeding time step. In the worst case, $|\hat{E}|^4$ directed arcs must be checked for conflict, requiring exactly three conditions to be tested (per Figure 4.4).

Development of n-agent spatio-temporal graphs (with $n > 2$) can be simplified in a similar way. However, the conflict-checking operation is not as simple as the set of included agents grows. More specifically, for a simple pair of agents, we must avoid the following conflicts:

1. Test whether $(e_1', t+1)$ conflicts with $(e_2', t+1)$.
2. Test whether $(e_1', t+1)$ conflicts with $(e_2', t)$.
3. Test whether $(e_2', t+1)$ conflicts with $(e_1', t)$.

However, for $n > 2$, more than three of these checks are required for an arc to be included in the spatio-temporal graph. First, it is necessary to verify that no agent at time $t + 1$ conflicts with any other agent at time $t + 1$. Since $\frac{n^2 - n}{2}$ distinct pairings exist between agents, this implies that $\frac{n^2 - n}{2}$ conflict checks are required. Once that process is complete, it is necessary to verify that the considered joint state $(e'_1, e'_2, ..., e'_n, t + 1)$ would not cause any conflicts with the state $(e_1, e_2, ..., e_n, t)$ that it is being connected to. This process requires $n^2 - n$ separate conflict checks to be made (corresponding to all pairings of one agent location in time $t + 1$ with another agent location at time $t$), yielding a total of $\frac{3}{2}(n^2 - n)$ conflict checks, implying that the conflict-checking operation scales according to $O(n^2) = O(|\mathcal{A}|^2)$ time.

Also, for $n > 2$, we must consider the expansion of the potential state space. For each $t$ such that $0 < t < T$, up to $|\hat{E}|^n$ states may be feasible, and up to $|\hat{E}|^{2n}$ transitions may exist between successive time steps. Therefore, the conflict-checking tests may be invoked, in the worst case, $O(T|\hat{E}|^{2n}) = O(T|\hat{E}|^{2|\mathcal{A}|})$ times.

From this it follows that a multi-agent spatio-temporal graph with time horizon $T$ can be constructed for $|\mathcal{A}|$ agents in $O(|\mathcal{A}|^2 T|\hat{E}|^{2|\mathcal{A}|})$ time. As mentioned in the previous section, the optimal makespan can be determined by checking the "parents" of the terminal node corresponding to the "end" state to see if they include state $(d_{a_1}, d_{a_2}, ..., d_{a_n}, T - 1)$, and then repeating until the minimum possible makespan is established. From there, optimal schedules can be found by backtracing from $(d_{a_1}, d_{a_2}, ..., d_{a_n}, w^*)$. In the worst case, this backtracing operation scales according to $O(T|\hat{E}|^2)$ and is therefore dominated by the $O(|\mathcal{A}|^2 T|\hat{E}|^{2|\mathcal{A}|})$ scaling of the graph generation procedure.

If $T < w^*$, then the same simplified procedure can be executed, except that the terminal node corresponding to the end state will have no parents. This will immediately imply that $w^*$ exceeds the given time horizon, $T$. $\square$

Given a fixed number of agents, $|\mathcal{A}|$, and a fixed number of edges, $|\hat{E}|$, Proposition 4.6.1 establishes a pseudo-polynomial worst-case computational complexity for the technique of optimizing the makespan for the MIP of Chapter 2 using multi-agent spatio-temporal graphs. Since the number of edges, $|\hat{E}|$, is polynomially related to the number of bits required to specify the MIP in Chapter 2, Proposition 4.6.1 also demonstrates that, given a fixed $T$ and a fixed number of agents, $|\mathcal{A}|$, multi-agent spatio-temporal graphs can be built with polynomial worst-case complexity. The precise procedure described in the earlier parts of this section is designed to leverage careful pruning and a "bi-directional" search in order to save both time and memory.

It is clear, however, that the presented technique for constructing multi-agent spatio-temporal graphs is not scalable for large numbers of agents. It is conjectured, but not proven here, that the problem defined by the MIP described in Chapter 2 is NP-hard when the number of agents in the system is not held constant.[7]

*Increasing Search Depth with Multi-Agent Spatio-Temporal Graphs*

It is possible to use multi-agent graphs to expand the neighborhood of the "local search" described in Section 4.3. This can be implemented in different ways. For instance, given $n$ agents, it is possible to generate $\frac{n^2 - n}{2}$ different two-agent graphs in place of the $n$ single-agent graphs used before (or, perhaps, a subset of these graphs for selected agent paris). It is then possible to implement the algorithm of Figure 4.3, along with the infeasible search of Section 4.6.1, using these graphs.

Deeper searches (using three-agent graphs or larger) are possible but are often not practical since, as seen in the previous section, this approach, in the worst case, may be exponentially complex with respect to the number of agents considered simultaneously. However, the fact that a method relying on an $n$-agent simultaneous search would be guaranteed to return an optimal solution implies that, given sufficient search depth, an implementation of the algorithm of Figure 4.3 can be "unstuck" from any local optimum.

[7]This conjecture is partially based on the similarity of the described class of problems to a highly flexible version of the "job-shop" scheduling problem, which is known to be NP-hard [62]. In addition, as noted in Section 1.2.1, the embedded "state safety" problem induced by the deadlock-avoidance requirement is known, in similar RAS contexts, to be NP-hard [46].

Some results related to multi-agent searches are included in Chapter 5.

### 4.6.3 Lagrange-Multiplier Heuristic

Other variants of the algorithm detailed in Figure 4.3 can be derived by embedding different heuristics. For instance, one can consider a number of heuristics for breaking "ties" in the "shortest-path" computation that will improve the solution time and/or quality. In particular, a heuristic that biases an agent to reach its destination at the earliest time step possible, when multiple conflict-free solutions exist, may allow us to reduce the makespan with fewer iterations of the schedule $\mathcal{S}^{(i)}$.

Alternatively, it is possible to exploit a set of Lagrange multipliers (computed using an LP as in Section 3.3). In particular, the coefficients $C_{a,e,t}^{\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*}$ corresponding to the optimized dual problem defined in Section 3.1 can be used to define "costs" of having an agent $a$ located in position $e$ at time $t$, and the Lagrange multiplier $\nu_a^*$ can be considered as a "reward" for having an agent $a$ located at its destination edge, $d_a$.

Since the corresponding costs must be defined over a time horizon for which a feasible solution is already known, and since the size of the LP defined in Section 3.3 is significantly affected by the length of the time horizon over which it is defined, it makes sense for us to consider a procedure along the following lines: First, we run the algorithm described in Figure 4.3, enhanced using the digressions from the feasible region described in Section 4.5. Once that algorithm is unable to find an improving (feasible) solution, we can obtain the optimal Lagrange multipliers for the corresponding dual problem using the LP method described in Section 3.3, and compute the "costs" and "rewards" for placing agent $a$ in location $e$ at time $t$. This information can then be used in a straightforward way to break "ties" that occur when eliminating conflicts using the "shortest-path" conflict-elimination approach described in Section 4.3.

More specificaly, since $C_{a,e,t}^{\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*}$ effectively defines a cost, based on the Lagrange multipliers, that is incurred when $x_{a,e,t} = 1$ and $e \neq d_a$, and since $\nu_a^*$ suggests a "reward" for having an agent $a$ located at its destination edge, $d_a$, we infer that when $e = d_a$, the value of placing agent $a$ on edge $d_a$ can be given by $C_{a,e,t}^{\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*} - \nu_a^*$. Given that the algorithm described in Figure 4.3 structurally ensures that each agent resides in its destination location, $d_a$, at the end of the considered time horizon, we can then define the following:

$$\mathcal{C}_{a,e,t} := \begin{cases} C_{a,e,t}^{\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*} - \nu_a^*, & \text{if } t = T \\ C_{a,e,t}^{\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*} - I_{\{e=d_a\}}\nu_a^* + \mathcal{C}_{a,\hat{e},t+1}, & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\hat{e} \in e^\bullet \cup e$ when $e \neq d_a$, $\hat{e} = d_a$ when $e = d_a$, and $\hat{e}$ corresponds to the edge chosen to succeed $e$, according to a DP implementation that first minimizes the number of conflicts and then uses the minimization of $\mathcal{C}_{a,e,t}\ \forall a \in \mathcal{A}, \forall e \in \hat{E}, \forall t \in \mathcal{T} \setminus T$ as a secondary objective.

This approach will be more computationally intensive (and more memory-intensive) than the basic approach detailed in Chapter 4, but it should be noted that once the “costs” and “rewards” have been computed for a time horizon of $\tau$, there is no need ever to recompute them for the same instance of the considered problem; rather, the costs and rewards defined by the Lagrange multipliers can be retained as triplets $(a, e, t)$ and efficiently re-used for all subsequent iterations of the algorithm, including those that digress into the infeasible region. Some results related to the implementation of a Lagrange-multiplier heuristic are included in Chapter 5.

# CHAPTER 5
# EXPERIMENTAL RESULTS

A problem instance that was used to motivate our research is shown in Figure 1.1. In Chapter 1, we discussed how this simple, 3-agent problem instance, which requires the agents to transit across the graph and reverse their order along the way, lies beyond the abilities of some of the most efficient scalable multi-agent routing algorithms, since agents can block one another when they reach their destinations.

A simple implementation of the algorithm described in Figure 4.3 on a MacBook Pro, however, returned an optimal schedule with a makespan of 16 time steps in just 25 milliseconds. That solution is detailed in Table 5.1.

Table 5.1 describes the solution found by our scheduling algorithm by providing the edge sequences used by each agent, $a_i,\ i = 1, 2, 3$, over the time horizon $\{0, \ldots 16\}$. Tracing the solution closely, one can observe the "intelligence" required to solve this problem instance, as some agents are "sidetracked" and the guidepath graph is used to reorder the agents as necessary.

As interesting as this example is, and as promising as the algorithmic result is, it only represents one small instance of the considered problem. We next sought to evaluate the considered algorithm by applying it to an especially difficult "challenge problem". Figure 5.1 represents the guidepath network designed for this purpose.[1]

[1]The representation used for the guidepath graphs in Figures 5.1, 5.2, and 5.3 differs slightly from the

Table 5.1: An optimal set of agent routes computed for the problem instance of Figure 1.1 – adapted from [10].

| $t$ | $a_1$ | $a_2$ | $a_3$ | $t$ | $a_1$ | $a_2$ | $a_3$ | $t$ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 28 | 30 | 32 | 6 | 25 | 35 | 37 | 12 | 12 | 17 | 21 |
| 1 | 28 | 30 | 34 | 7 | 23 | 33 | 37 | 13 | 14 | 10 | 19 |
| 2 | 28 | 32 | 25 | 8 | 21 | 25 | 35 | 14 | 16 | 12 | 17 |
| 3 | 30 | 34 | 36 | 9 | 19 | 23 | 33 | 15 | 16 | 14 | 10 |
| 4 | 32 | 25 | 38 | 10 | 17 | 21 | 25 | 16 | 16 | 14 | 12 |
| 5 | 34 | 36 | 38 | 11 | 10 | 19 | 23 | | | | |

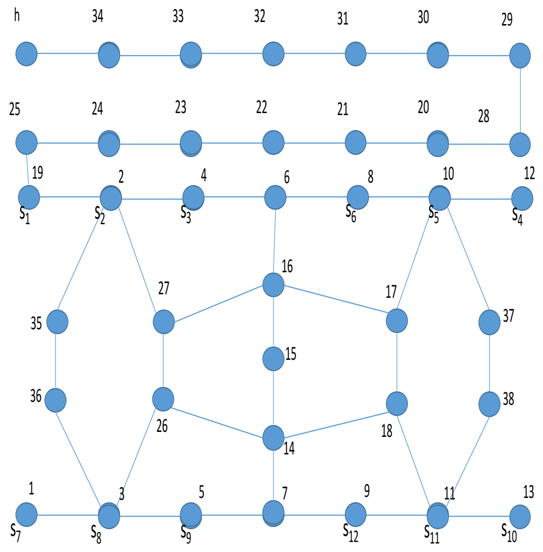

Figure 5.1: The problem instance employed in the second case study.

This problem instance involves twelve agents, $a_i$, $i = 1, \ldots, 12$. The starting location of the $i$th agent is marked in Figure 5.1 as $s_i$. The agents form a series of disjunct pairs that are required to swap positions with one another; those pairs are given by $\{a_1, a_{10}\}, \{a_2, a_{11}\}, \{a_3, a_{12}\}, \{a_4, a_7\}, \{a_5, a_8\}$ and $\{a_6, a_9\}$ and are graphically represented in Figure 5.2. In addition to the high density of agents on the graph (12 agents on 39 locations in the guidepath graph), the swaps are selected to maximize congestion by ensuring that every shortest path between each agent's origin and destination intersects with the shortest paths for every other agent. In addition, the "home" location, labeled as "h" in Figure 5.1 and as "D" in Figure 5.2, lies at the end of a long path that effectively places it very far away from any agent's origin or destination. This means that the home location cannot be used to reorganize agents without incurring a very high cost with respect to the makespan of the resulting schedule.

Using an HP Z230 workstation with an Intel core i7 processor and 8 GB RAM, running

representation used for Figure 1.1 in the sense that traveling agents reside at the nodes rather than at the edges. The definitions of "neighborhood" are similarly defined as before since the guidepath network in any problem instance can be defined fully from the connectivity of the locations on the guidepath graph.

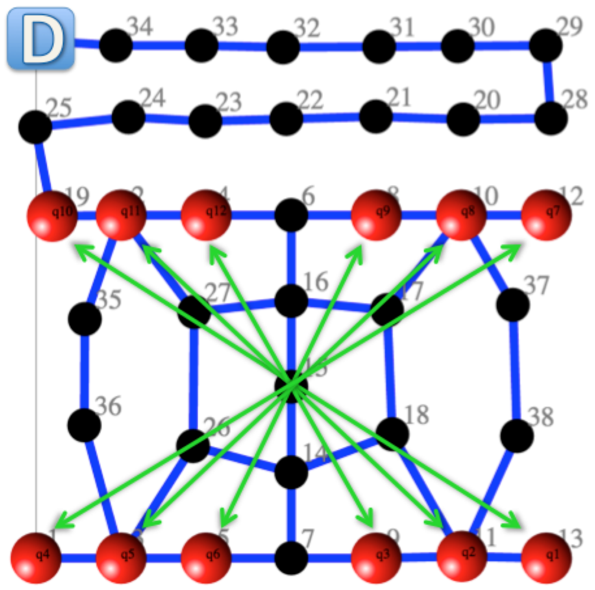

Figure 5.2: The problem instance employed in the second case study, with arrows representing the requirements that various pairs of agents swap places, and with "D" representing the depot.

Fedora, we implemented the algorithm from Figure 4.3 and applied it to the problem described by Figures 5.1 and 5.2. This canonical version of the algorithm required 739 milliseconds to run and returned a feasible solution with a makespan equal to 72 periods. However, by applying the version of the algorithm that allows for controlled excursions to the infeasible region (as described in Section 4.6.1) on the same computer, we obtained a feasible solution with a makespan of 24 periods after a runtime of 8.6 seconds. This result is reported in Table 5.2. Lastly, we used CPLEX to solve the MIP formulation of Section 2.2 for this problem instance. The resulting optimal makespan required only 12 periods, but the required solution time was about 5 days. The optimal schedule returned by CPLEX is given in Table 5.3.

Next, we designed another line of experimentation (originally reported in [10]) to measure the performance and the robustness of the algorithm described in Chapter 4. These experiments were performed using the guidepath network depicted in Figure 5.3. Note that Figure 5.3 depicts a "dual" version of the employed guidepath graph, where the various

Table 5.2: The agent routes computed for the second case study by the variation of the proposed algorithm that allows for excursions to the infeasible region – cf. Section 4.6.1

| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $a_9$ | $a_{10}$ | $a_{11}$ | $a_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | 2 | 4 | 12 | 10 | 8 | 1 | 3 | 5 | 13 | 11 | 9 |
| 25 | 2 | 4 | 12 | 10 | 6 | 1 | 26 | 5 | 13 | 18 | 7 |
| 25 | 19 | 4 | 12 | 8 | 6 | 3 | 27 | 5 | 11 | 17 | 14 |
| 25 | 19 | 2 | 10 | 8 | 6 | 26 | 27 | 7 | 18 | 16 | 15 |
| 25 | 19 | 35 | 10 | 8 | 4 | 14 | 27 | 7 | 17 | 16 | 15 |
| 25 | 19 | 36 | 10 | 6 | 2 | 18 | 27 | 7 | 17 | 16 | 15 |
| 24 | 19 | 3 | 8 | 4 | 35 | 11 | 27 | 14 | 17 | 16 | 15 |
| 24 | 25 | 5 | 6 | 2 | 36 | 38 | 27 | 18 | 17 | 16 | 15 |
| 24 | 25 | 7 | 4 | 19 | 3 | 37 | 27 | 18 | 17 | 16 | 15 |
| 24 | 25 | 9 | 2 | 19 | 5 | 10 | 27 | 18 | 17 | 6 | 15 |
| 24 | 25 | 9 | 35 | 19 | 5 | 12 | 27 | 18 | 16 | 8 | 15 |
| 24 | 25 | 9 | 36 | 2 | 5 | 12 | 26 | 17 | 6 | 8 | 15 |
| 24 | 19 | 9 | 3 | 27 | 5 | 12 | 14 | 16 | 4 | 8 | 15 |
| 25 | 2 | 9 | 1 | 26 | 5 | 12 | 18 | 6 | 4 | 10 | 15 |
| 19 | 27 | 9 | 1 | 3 | 5 | 12 | 17 | 8 | 4 | 37 | 16 |
| 19 | 26 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 2 | 38 | 6 |
| 19 | 14 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 4 | 11 | 6 |
| 2 | 7 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 4 | 18 | 6 |
| 27 | 7 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 4 | 14 | 6 |
| 16 | 7 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 2 | 26 | 6 |
| 17 | 7 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 | 27 | 4 |
| 18 | 7 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 | 2 | 4 |
| 11 | 14 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 | 2 | 4 |
| 13 | 18 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 | 2 | 4 |
| 13 | 11 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 | 2 | 4 |

Table 5.3: An optimal set of routes for the second case study computed through the MIP formulation of Section 2.2

| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $a_9$ | $a_{10}$ | $a_{11}$ | $a_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | 2 | 4 | 12 | 10 | 8 | 1 | 3 | 5 | 13 | 11 | 9 |
| 19 | 2 | 4 | 12 | 17 | 6 | 1 | 26 | 5 | 13 | 38 | 7 |
| 19 | 27 | 4 | 10 | 18 | 16 | 3 | 26 | 5 | 11 | 37 | 7 |
| 2 | 27 | 6 | 10 | 18 | 17 | 36 | 26 | 5 | 9 | 38 | 14 |
| 4 | 16 | 8 | 37 | 18 | 17 | 35 | 26 | 3 | 7 | 11 | 15 |
| 6 | 16 | 10 | 38 | 14 | 17 | 2 | 27 | 36 | 5 | 9 | 15 |
| 8 | 16 | 37 | 11 | 26 | 18 | 4 | 27 | 35 | 3 | 7 | 15 |
| 10 | 17 | 38 | 9 | 26 | 14 | 6 | 27 | 2 | 36 | 5 | 15 |
| 37 | 18 | 11 | 7 | 26 | 14 | 8 | 16 | 4 | 35 | 3 | 15 |
| 38 | 18 | 9 | 5 | 27 | 14 | 10 | 17 | 6 | 2 | 36 | 15 |
| 11 | 18 | 9 | 3 | 27 | 7 | 10 | 17 | 8 | 19 | 35 | 16 |
| 13 | 18 | 9 | 1 | 26 | 5 | 12 | 17 | 8 | 19 | 2 | 6 |
| 13 | 11 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 | 2 | 4 |

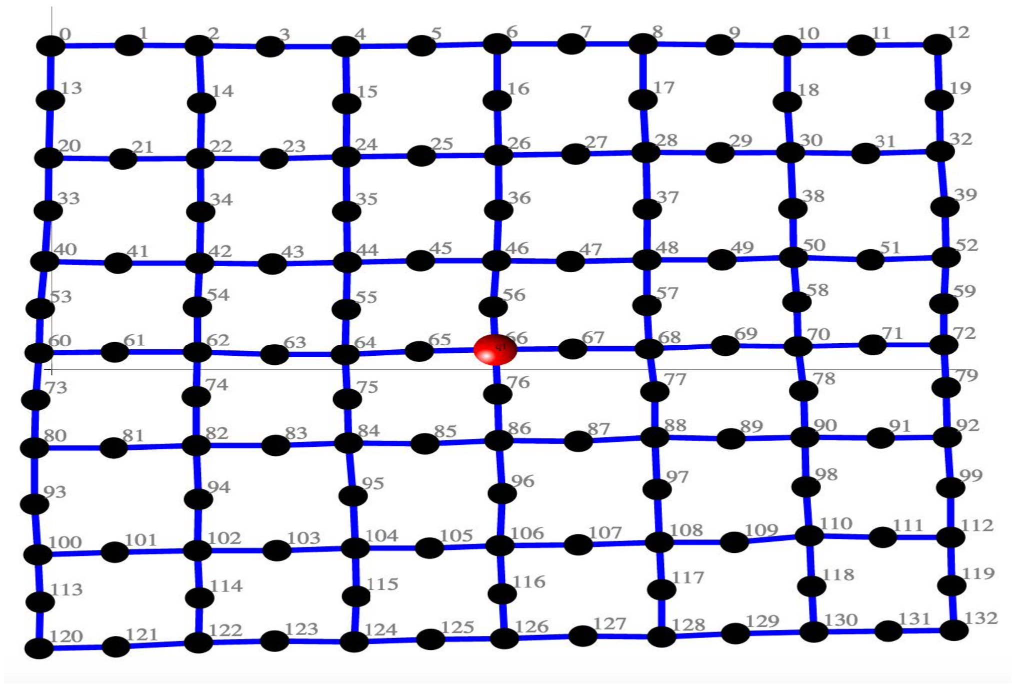

Figure 5.3: The guidepath graph used for the grid-based numerical experiment – reproduced from [10].

zones are represented by the nodes of the graph, and the edges that connect these nodes define the adjacency sets of these zones. This guidepath network, then, is composed of 133 zones, organized as a grid.

The red node at the center of the guidepath graph in Figure 5.3 indicates the location of the "depot". We conducted our experiments with two different depot locations: (i) the middle of the guidepath network (as shown in Figure 5.3), and (ii) one of the corners of the graph.[2] The results that follow in the remainder of this chapter demonstrate that this selection can make a difference in, both, the performance and the execution time of the algorithm described in Chapter 4.

The employed problem instances were built by first assigning 3 agents randomly to 3 separate initial locations and 3 destinations. After the execution time and makespan returned by the algorithm were recorded for the cases of, both, a central depot and a corner depot,

[2]We chose to use the upper-left corner in our experiments. However, due to the symmetries of the graph in Figure 5.3, all four corners are topologically equivalent.

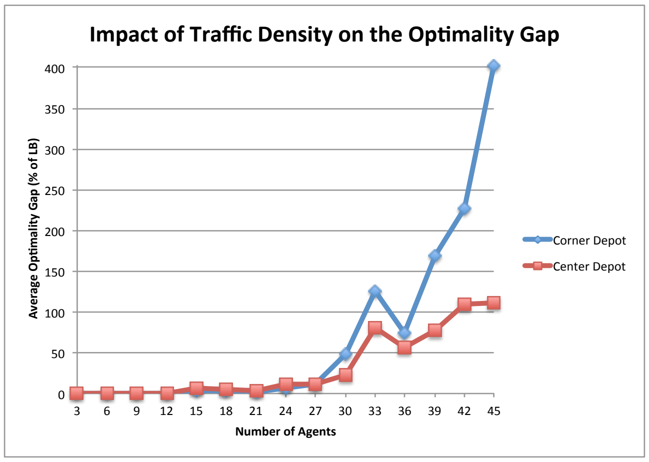

Figure 5.4: Plotting the optimality gap for the grid-based numerical experiment – reproduced from [10].

3 more agents were assigned to 3 more initial locations and 3 more destinations. In this way, the density of agents on the graph was progressively increased in increments of 3 until the graph (whose size never changed) was host to 45 agents, covering more than 1/3 of the available zones.

Each of these progressions of increasing density was replicated five separate times with different, re-randomized initial and final locations for each agent, for a total of 150 experiments. All replications were performed on an HP Z230 workstation with an Intel core i7 processor and 8 GB RAM, running Fedora Linux. The obtained results are summarized in Figures 5.4 and 5.5.

Figure 5.4 reports the optimality gaps for the obtained schedules, averaged across the five replications for each considered problem instance. These optimality gaps were calculated using the single-LP solution of the Lagrangian dual problem of the considered problem instances that is described in Section 3.3; for each replication, the value of the optimality gap was computed according to the following formula:

$$\frac{\text{obt. sched. makespan } - \text{ opt. value of Lagr. dual}}{\text{opt. value of Lagr. dual}} \times 100$$

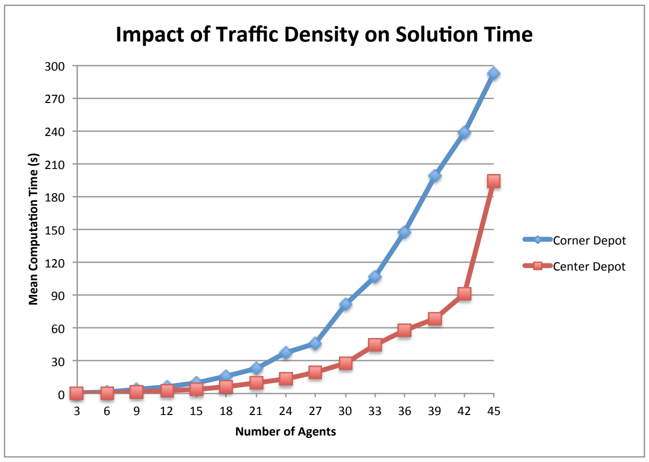

Figure 5.5: Plotting the computational times for the grid-based numerical experiment – reproduced from [10].

The plots shown in Figure 5.4 suggest that the performance of the considered algorithm is very close to the optimal result (and, in many cases, provably optimal) when zone occupancy by the agents is low, but that this performance eventually degrades as the ratio of agents to the number of locations in the guidepath increases.[3] Furthermore, it can be seen that the degradation is more severe in the cases where the depot is located in a far-flung location away from the "center" of the guidepath network.

Figure 5.5 details the computation times for the algorithm as a function of the number of agents placed on the grid. These times correspond to the same five replications used to create Figure 5.4. Figure 5.5 shows that the algorithm executes very quickly, even for problems characterized by highly congested guidepath networks. As with Figure 5.4, Figure 5.5 demonstrates the importance of, both, traffic density and centrality of the depot location to the difficulty of a considered instance of the multi-agent routing problem.

In addition to controlled excursions into the infeasible region, two other techniques were explored in an effort to improve upon these results: multi-agent simultaneous routing (Section 4.6.2) and use of Lagrange multipliers as a tie-breaking heuristic within the local

[3]It should be noted that the worst-case performance is always bounded for problem instances of this type (i.e., where reversibility holds and where a home edge exists with sufficient capacity to hold all agents in the system) since an initial feasible solution is always available that efficiently routes all agents from their initial locations to the home edge and then back out to their respective destinations.

Table 5.4: An optimal set of routes for the first ten agents of the challenge problem, computed using a 10-agent spatio-temporal graph, as described in Section 4.6.2
.

| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $a_9$ | $a_{10}$ |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 2 | 4 | 12 | 10 | 8 | 1 | 3 | 5 | 13 |
| 19 | 35 | 4 | 12 | 10 | 6 | 1 | 36 | 5 | 11 |
| 2 | 35 | 4 | 12 | 8 | 16 | 3 | 36 | 7 | 11 |
| 27 | 35 | 6 | 10 | 8 | 15 | 5 | 36 | 9 | 38 |
| 26 | 2 | 16 | 17 | 8 | 14 | 7 | 36 | 11 | 37 |
| 3 | 27 | 15 | 17 | 6 | 14 | 9 | 35 | 38 | 10 |
| 5 | 16 | 15 | 18 | 4 | 26 | 11 | 2 | 37 | 8 |
| 7 | 17 | 15 | 14 | 4 | 3 | 38 | 27 | 10 | 6 |
| 9 | 17 | 15 | 26 | 2 | 5 | 37 | 16 | 8 | 6 |
| 11 | 18 | 14 | 3 | 27 | 5 | 10 | 16 | 8 | 4 |
| 13 | 18 | 7 | 1 | 26 | 5 | 12 | 17 | 8 | 2 |
| 13 | 11 | 9 | 1 | 3 | 5 | 12 | 10 | 8 | 19 |

search (Section 4.6.3). As an exercise, we attempted to find optimal schedules for the 12-agent challenge problem of Figure 5.2 by building a 12-agent spatio-temporal network on a MacBook Pro. This attempt proved to be too time- and memory-intensive to succeed. However, experiments with subsets of the agents within the challenge problem demonstrated that, in practice, the time and memory required to build multi-agent spatio-temporal graphs were very sensitive to the length of the time horizon, $T$, due to the fact that small values of $T$ result in smaller numbers of feasible schedules for any given set of agents.

Using $T = 11$, we were able to find optimal, multi-agent schedules for up to 10 agents within a few hours. Table 5.4 details one of these schedules, based on the guidepath graph from Figure 5.1. In this case, an optimal schedule, with $w^* = 11$, was obtained by building and applying a 10-agent spatio-temporal graph, as described in Section 4.6.2. This was achieved in 21,939 seconds (i.e., about 6 hours and 5 minutes) when implemented in C++ on a MacBook Pro.

As suggested in Section 4.6.2, we also used 2-agent spatio-temporal graphs to increase the size of the neighborhood searched by the local-search algorithm. The result was reasonably scalable in practice, but the observed improvements were either modest or non-existent. An implementation of the 2-agent search on the 12-agent challenge problem produced an improved, 22-time-step solution in 208 seconds on an HP Z230 workstation

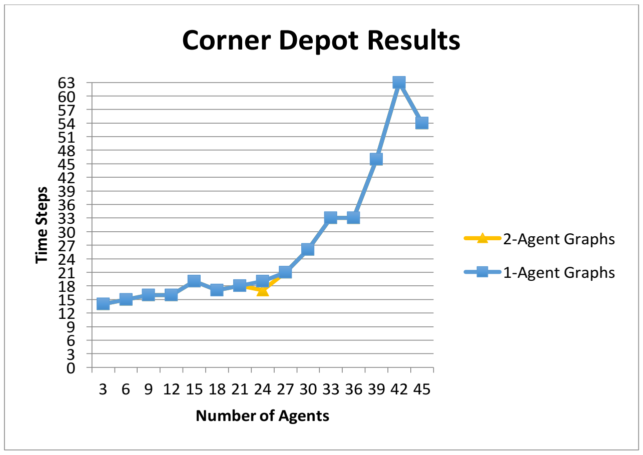

Figure 5.6: Comparison of the solutions obtained using 2-agent spatio-temporal graphs to the results from single-agent spatio-temporal graphs. In most cases, there is no improvement.

with an Intel core i7 processor and 8 GB RAM, running Fedora.

These results were further tested on the grid-based version of the considered problem whose guidepath graph is shown in Figure 5.3, assuming a corner depot. In almost all cases, no improvement was observed, and when improvement *was* observed, the effect was modest. These results are compiled in Figure 5.6. The time required for these experiments (again, using an HP Z230 workstation with an Intel core i7 processor and 8 GB RAM, running Fedora) is shown in Figure 5.7. Though the time required was not excessive for smaller problems, instances with large numbers of agents (and longer time horizons) required several hours to conduct local searches using all of the resulting 2-agent spatio-temporal graphs.

Though the improvements obtained using 2-agent spatio-temporal graphs are minor, one can be assured that sufficiently "deep" local searches (based on simultaneous consideration of sufficiently large sets of agents) can, eventually, free the algorithm from any "local optimum" that it might reach by searching neighborhoods based upon single-agent spatio-temporal graphs. However, as these "deeper" searches become increasingly complex, they will often become impractical unless time horizons are very short and/or considerable computing resources are available. On the flip side, the above results also suggest that the algorithmic implementation based on single-agent DAGs, using controlled excursions into

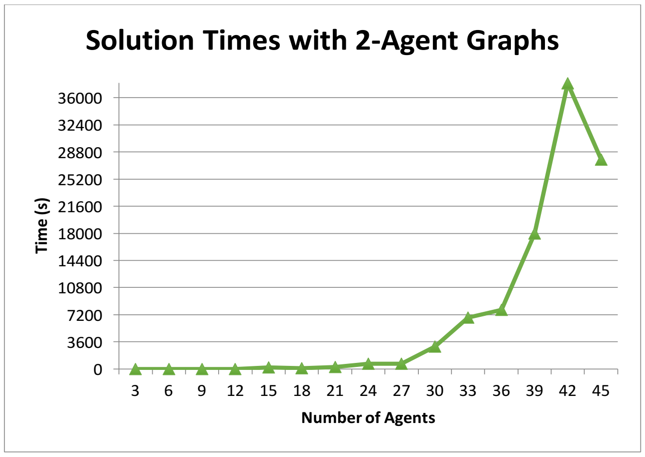

Figure 5.7: Completion time required when 2-agent spatio-temporal graphs are built and used after single-agent local search terminates. The reason for the spike observed for 42 agents is because, in that particular problem instance, the optimized makespan obtained using single-agent local search was greater than in the 45-agent case. This resulted in a longer time horizon for the 2-agent spatio-temporal graphs in the 42-agent problem.

the infeasible solution space, already exploits a substantial part of the information about the underlying problem that can be obtained from a 2-agent simultaneous search.

In addition to multi-agent spatio-temporal graphs, we experimented with the use of Lagrange multipliers as part of a "tie-breaking heuristic" within the local-search algorithm (cf. Section 4.6.3). When experimenting with the grid-like guidepath network of Figure 5.3, we found that a prohibitive amount of memory was often required for the consideration of large numbers of agents over lengthy time horizons. However, the technique was successfully implemented for the 12-agent "challenge" problem whose guidepath is shown in Figure 5.1. In this case, however, the information obtained from the Lagrange multipliers did not cause the algorithm to improve upon the "local optimum" found using single-agent spatio-temporal graphs and controlled infeasible search (cf. Section 4.6.1).

# CHAPTER 6
# CONCLUSION

This thesis analyzes and addresses the problem of generating efficient, multi-agent schedules, subject to congestion constraints on a shared guidepath network. This research question is motivated by a diverse set of real-world problems, ranging from the flexible routing of automated guided vehicles (AGVs) within a production or distribution setting, to that of routing ions (subject to similar congestion constraints) within a quantum computer. Following a review of the relevant literature, we formulated the considered problem as a mixed-integer program, similar in spirit to the formulation described in [52] for the flexible job-shop scheduling problem.

Given this formulation, we approached the problem in two ways. First, we formulated the Lagrangian relaxation and the corresponding dual problem, which we optimized using two distinct methods: (i) a customized dual-ascent algorithm guaranteeing precise convergence in finite time, and (ii) an LP formulation. Both of these approaches were useful for finding lower bounds for various problem instances, as well as for obtaining Lagrange multipliers that could be used as part of a heuristic for an algorithm that builds feasible schedules. Both methods are memory-intensive, but the LP formulation seems to be more robust for larger problems.

Secondly, we described an algorithm for iteratively reducing the makespan of multi-agent schedules using a novel, scalable local-search methodology. We described the implementation of this algorithm within the context of quantum computing, including a method for finding initial feasible schedules. We further described various "improvements" to the makespan-reduction algorithm, including (i) a controlled search through the infeasible region, (ii) multi-agent-based searches, and (iii) implementation of a heuristic based on Lagrange multipliers.

Our experimental results show that solving the Lagrangian dual provides tight lower bounds for many problems, although these bounds do appear to loosen when an environment becomes extremely congested. Similarly, the application of the local-search methodology with controlled excursions to the infeasible region to problems arising within quantum computing resulted in provably optimal or near-optimal solutions for many problem instances, although the algorithm performance did eventually degrade in cases of extreme congestion.

Two-agent-based local search did not improve on this performance in most cases, implying that the originally incurred search typically does not stop finding improving feasible solutions until a "local optimum" is encountered that cannot be escaped with simultaneous rerouting of just two agents. "Deeper" searches (simultaneously rerouting three or more agents) would eventually escape such "local optima", but the algorithm becomes increasingly difficult to scale to larger guidepaths and larger numbers of agents as this "depth" increases.

Similarly, the Lagrangian-relaxation heuristic did not clearly provide a major improvement in the performance of the local-search algorithm when applied to our challenge problem, and its memory-intensive nature made it difficult to apply consistently to cases where extreme congestion widened the optimality gap, underscoring both the quality of the local-search algorithm's results (when enhanced with a controlled search of the infeasible solution space), as well as the robustness of that approach.

Hence, although it is possible that multi-agent local searches and the Lagrange-multiplier heuristic (as implemented in this thesis) may provide a benefit that is sufficient to justify their implementation when considerable computing resources are available, the single-agent local-search schedule improvement scheme with controlled infeasible search demonstrates a much more practical combination of robustness, solution quality, and low computation time. This scheduler is scalable to large, congested guidepath networks, and it is capable of finding efficient, deadlock-free solutions for a wide range of problems.

One straightforward extension of the algorithm detailed in Figure 4.3 is an adaptation to problem instances where zone-traversal times are deterministic but non-uniform for various edge-agent pairs, $(e, a) \in E \times \mathcal{A}$. In such a case, the "time step" would be defined as the greatest common divisor (GCD) of the set of zone-traversal times, rather than being equivalent to the zone-traversal times themselves. This modification would not substantially increase the computational complexity of the heuristic algorithm since the non-uniform traversal times could be encoded into the arc connectivity of the spatio-temporal DAGs used to execute the algorithm; by incorporating deterministic, non-uniform zone-traversal times into the spatio-temporal DAGs, it is possible, as in the case of uniform zone-traversal times, to execute the heuristic algorithm in accordance with Figure 4.3, using DP to eliminate conflicts between agents.

Further research could investigate embedding the described algorithm into "rolling horizon" schemes for problem variations that do not possess all of the features (e.g., depots and reversibility) assumed to exist in Chapter 4. Furthermore, future research may improve the algorithm or enhance its utility by adapting it to stochastic environments, to continuous-time problems, or to other resource-allocation problems, such as flexible job-shop scheduling.

# Appendices

# APPENDIX A
# ALTERNATIVE MIP FORMULATION

If every agent is conceptualized as a unique commodity that must be transferred from location $s_a$ to $d_a$, it is possible to formulate the MIP of Section 2.2.3 in a different way, such that it can be viewed as a unit-capacity, integral multicommodity flow problem with "side" constraints. Here, we will use the same notation as that defined in Chapter 2, with the following additional definition: $f_{a,t}(e, e') \in \{0, 1\}$ represents a unit "flow" of agent $a$ from edge $e$ to edge $e' \in e^{\bullet} \cup \{e\}$ between time step $t$ and time step $t + 1$.

Then, given the spatio-temporal graph $\mathcal{N}_a$ whose nodes $(e, t)$ are elements of $E \times \mathcal{T}$ and are connected by arcs $((e, t), (e', t + 1))\ \forall e \in E, \forall e' \in e^{\bullet} \cup \{e\}, \forall T \in \mathcal{T} \setminus \{T\}$, we can interpret the (binary) variables $f_{a,t}(e, e')$ as flows over the arcs $((e, t), (e', t + 1))$ and interpret the (binary) decision variables $x_{a,e,t}$ from the MIP detailed in Chapter 2 as the total flow of agent $a$ entering, leaving, or passing through node $(e, t)$. The precise relationship depends on whether $(e, t)$ constitutes a "source node" for $a$, a "sink node" for $a$, or a "transit node" for $a$. We can define these node types and the relevant relationships between the flow variables, $f_{a,t}(e, e')$, and the decision variables from Chapter 2, $x_{a,e,t}$, as follows:

*"Source Nodes":* For an agent $a \in \mathcal{A}$, a spatio-temporal node, $(s_a, 0)$, corresponding to the agent's unique starting location $s_a$ at time $t = 0$, can be viewed as a "source" for the commodity represented by agent $a$, and the total flow exiting this location must sum to 1. In addition, no amount of flow corresponding to agent $a$ may originate from a node other than $(s_a, 0)$. That is,

$$\forall a \in \mathcal{A},\ \forall e \in E, \quad \sum_{e' \in e^{\bullet} \cup \{e\}} f_{a,0}(e, e') = I_{\{e = s_a\}} = x_{a,e,0}. \tag{A.1}$$

*"Sink Nodes":* For an agent $a \in \mathcal{A}$, a spatio-temporal node, $(d_a, 0)$, corresponding to the agent's destination location $d_a$ at time $t = T$, can be viewed as a "sink" for the commodity represented by agent $a$, and the total flow entering this location must sum to 1. That is,

$$\forall a \in \mathcal{A}, \quad \sum_{e \in E} f_{a,T-1}(e, d_a) = 1 = x_{a,d_a,T}. \tag{A.2}$$

*"Transit Nodes":* For an agent $a \in \mathcal{A}$, a spatio-temporal node, $(e, t)$, where $t \in \mathcal{T} \setminus \{0, T\}$, can be viewed as a "transit" node, and the decision variable $x_{a,e,t}$ can be conceptualized as the total flow of agent $a$ through node $(e, t)$, which tells us that

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{0, T\}, \quad \sum_{e' \in {}^{\bullet}e \cup \{e\}} f_{a,t-1}(e', e) = x_{a,e,t}, \tag{A.3}$$

and that

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{0, T\}, \quad \sum_{e'' \in e^{\bullet} \cup \{e\}} f_{a,t}(e, e'') = x_{a,e,t}. \tag{A.4}$$

The vector of flow variables, $\mathbf{f}$, can be combined, much like in Chapter 2, with an auxiliary variable, $w$, to form a complete set of decision variables for this new MIP. Using the equivalences that were defined in the above discussion to make the necessary substitutions, we can reformulate the MIP of Equations (2.1)–(2.10) in terms of $\mathbf{f}$ and $w$, as follows:

$$\min w \tag{A.5}$$

s.t.

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T}, \quad \sum_{e \in E} \sum_{e' \in e^{\bullet} \cup \{e\}} f_{a,t}(e, e') = 1 \tag{A.6}$$

$$\forall a \in \mathcal{A},\ \forall e \in E, \quad \sum_{e' \in e^{\bullet} \cup \{e\}} f_{a,0}(e, e') = I_{\{e = s_a\}} \tag{A.7}$$

$$\forall a \in \mathcal{A}, \quad \sum_{e \in E} f_{a,T}(e, d_a) = 1 \tag{A.8}$$

$$\forall a \in \mathcal{A},\ \forall t \in \mathcal{T} \setminus \{T\}, \quad \sum_{e \in E} f_{a,t+1}(e, d_a) \geq \sum_{e \in E} f_{a,t}(e, d_a) \tag{A.9}$$

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{0, T\}, \quad \sum_{e' \in {}^{\bullet}e \cup \{e\}} f_{a,t-1}(e', e) \leq \sum_{e'' \in e^{\bullet} \cup \{e\}} f_{a,t}(e, e'') \tag{A.10}$$

$$\begin{aligned} &\forall e = (v_i, v_j) \in E \text{ s.t. } i < j,\ \forall t \in \mathcal{T} \setminus \{0, T\}, \\ &\qquad \sum_{a \in \mathcal{A}} \Big[ \sum_{e' \in e^{\bullet} \cup \{e\}} f_{a,t}(e, e') + \sum_{e' \in \bar{e}^{\bullet} \cup \{\bar{e}\}} f_{a,t}(\bar{e}, e') \Big] \leq 1 \end{aligned} \tag{A.11}$$

$$\begin{aligned} &\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{0\}, \quad \sum_{e' \in {}^{\bullet}e \cup \{e\}} f_{a,t-1}(e', e) + \\ &\qquad + \sum_{a' \in \mathcal{A} : a' \neq a} \Big[ \sum_{e'' \in e^{\bullet} \cup \{e\}} f_{a',t-1}(e, e'') + \sum_{e'' \in \bar{e}^{\bullet} \cup \{\bar{e}\}} f_{a',t-1}(\bar{e}, e'') \Big] \leq 1 \end{aligned} \tag{A.12}$$

$$\begin{aligned}\forall a \in \mathcal{A}, \quad w \;\geq\; & \sum_{t=1}^{T} \Big[1 - \sum_{e' \in {}^{\bullet}d_a \cup \{d_a\}} f_{a,t-1}(e', d_a)\Big] - f_{a,0}(d_a, d_a) \\ =\; & T + 1 - I_{\{s_a = d_a\}} - \sum_{t=1}^{T} \sum_{e' \in {}^{\bullet}d_a \cup \{d_a\}} f_{a,t-1}(e', d_a) \\ =\; & T - \sum_{t=1}^{T} f_{a,t-1}(d_a, d_a) \\ =\; & \sum_{t=0}^{T-1} \sum_{e \in E \setminus \{d_a\}} \sum_{e' \in \left\{e^{\bullet} \cup \{e\}\right\}} f_{a,t}(e, e') \end{aligned} \quad \text{(A.13)}$$

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall e' \in e^{\bullet} \cup \{e\},\ \forall t \in \mathcal{T}, \quad f_{a,t}(e, e') \in \{0, 1\} \quad \text{(A.14)}$$

Furthermore, the equivalence between the "flow" of an agent *through* a transit node, $x_{a,e,t}$, and the total flows of the same agent into and out of that node (as described in Equations (A.3) and (A.4)) imply that Equation (A.10) can be replaced with the equality:

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{0, T\}, \quad \sum_{e' \in {}^{\bullet}e \cup \{e\}} f_{a,t-1}(e', e) = \sum_{e'' \in e^{\bullet} \cup \{e\}} f_{a,t}(e, e'') \quad \text{(A.15)}$$

Much like in Chapter 2, Equations (A.6)–(A.9) and (A.15) describe "single-agent" constraints, and Equations (A.11)–(A.13) describe "coupling" contraints. Equations (A.7), (A.8), and (A.15) describe "flow-balance" contraints for sources, sinks, and transit edges, respectively. Equation (A.11) can be thought of as a set of "side" constraints, but also constitutes a stricter version of the link capacity constraints typical of unit-capacity multicommodity flow problems. Equations (A.6), (A.9), and (A.13) describe sets of "side" constraints that correspond, respectively, to Equations (2.2), (2.5), and (2.9). The second equality of Equation (A.13) is based on the requirement, established in Equation (A.9), that an agent will not leave its destination once it arrives there. The last equality of Equation

(A.13) is based on the requirement, established in Equation (A.6), that an agent's total flow between each time step must equal 1, meaning that the the sum of all flows for a given agent must equal $T$; formally,

$$\sum_{t=0}^{T-1} \sum_{e \in E} \sum_{e' \in e^{\bullet} \cup \{e\}} f_{a,t}(e, e') = T, \ \ \forall a \in \mathcal{A}. \tag{A.16}$$

An important property of this formulation is given by the following theorem:

**Theorem A.1.** *The linear relaxation of the "single-agent" constraints in the new MIP–i.e., Equations (A.6)–(A.9), (A.14), and (A.15)–will have an integer-valued optimal solution for any linear objective function.*

**Proof:** Equations (A.7), (A.8), and (A.15) constitute a set of "flow-balance" constraints. Since these equations ensure that (i) a unit of flow originates, uniquely, at $(s_a, 0)$, (ii) the full unit of flow reaches $(d_a, T)$, and (iii) flow in conserved for all $t \in \mathcal{T} \setminus \{0, T\}$, Equation (A.6) is actually redundant.

Let the vector $\mathbf{f}_a$ correspond to an integral flow of agent $a$ through the spatio-temporal network $\mathcal{N}_a$, and let $F_a$ and $\mathbf{b}_a$ represent the coefficients of Equations (A.7), (A.8), and (A.15) for agent $a$. Then we can distill the single-agent constraints down to Equation (A.9) and

$$\forall a \in \mathcal{A}, \ \ F_a \mathbf{f}_a = \mathbf{b}_a, \tag{A.17}$$

while the integrality constraint can be written as

$$\forall a \in \mathcal{A}, \ \ \forall ((e,t)(e', t+1)) \in \mathcal{N}_a, \quad f_{a,t}(e, e') \in \{0, 1\}. \tag{A.18}$$

Now, let us modify the spatio-temporal graph $\mathcal{N}_a$, for every $a \in \mathcal{A}$, such that it includes every node $(e, t) \in E \times \mathcal{T}$, but excludes those arcs $((e,t), (e', t+1))$ where $e = d_a$ and $e' \neq d_a$, and let us refer to these new networks as $\hat{\mathcal{N}}_a$.

Graph $\hat{\mathcal{N}_a}$ structurally embodies the constraint set of Equation (A.9), allowing us to simplify the single-agent constraints further; if $\hat{F}_a$ is the flow-balance matrix induced by applying Equations (A.7), (A.8), and (A.15) to the spatio-temporal graphs $\hat{\mathcal{N}_a}$, and if $\hat{\mathbf{f}}_a$ is revised to include only the arcs of graph $\hat{\mathcal{N}_a}$, then we obtain the following representation of the single-agent and integrality constraints:[1]

$$\forall a \in \mathcal{A}, \;\; \hat{F}_a \hat{\mathbf{f}}_a = \mathbf{b}_a, \tag{A.19}$$

$$\forall a \in \mathcal{A}, \;\; \forall ((e,t)(e',t+1)) \in \hat{\mathcal{N}_a}, \quad \hat{f}_{a,t}(e,e') \in \{0,1\}. \tag{A.20}$$

Since the flow-balance matrix $\hat{F}_a$ is totally unimodular, and since the components of $\mathbf{b}_a$ are integer-valued for every $a \in \mathcal{A}$, it follows that the polytope formed by the linear relaxation of Equations (A.19) and (A.20) will have integer extreme points. Furthermore, since the linearly relaxed version of Equation (A.20) ensures that the polytope is compact, it follows that an integer-valued optimal solution must exist for *any* linear objective function evaluated over the set specified by the linear relaxation of Equations (A.19) and (A.20). □

Theorem A.1 implies that the MIP of Equations (A.5)–(A.9), Equations (A.11)–(A.14), and Equation (A.15) possesses the *integrality property* described in [22]. Hence, the linear relaxation of Equations (A.5)–(A.9), Equations (A.11)–(A.14), and Equation (A.15) will provide a lower bound for the optimal makespan that is identical to the bound obtained by relaxing the coupling constraints (Equations (A.11)–(A.14)) of the same MIP.

This property does not necessarily hold for instances of the MIP described by Equations (2.1)–(2.10). The important difference is highlighted by the contrast between Equation (A.10) and Equation (A.15); whereas Equation (A.15) ensures that an edge $e$ cannot receive any flow after $t = 0$ beyond what is available from ${}^\bullet e \cup \{e\}$, Equation (A.10) (which is

[1]Note that $\mathbf{b}_a$ does not change between Equation (A.17) and Equation (A.19). This is because each row of $F_a$ corresponds to a specific node $(e,t) \in \mathcal{N}_a$, and the node set (as well as the flow balance for each node) is unaffected by the modification of $\mathcal{N}_a$ to create $\hat{\mathcal{N}_a}$.

based directly on Equation (2.6), using simple substitutions to formulate it in terms of arc flows) effectively allows for the possibility that some amount of flow will "teleport" elsewhere in the spatio-temporal graph.[2] This "teleportation" phenomenon cannot occur when integrality constraints are enforced, which has two consequences: (i) Instances of the (unrelaxed) MIP formulated in Chapter 2 will have equivalent sets of feasible solutions and equal optimal objective values to the MIP of Equations (A.5)–(A.9), Equations (A.11)–(A.14), and Equation (A.15). (ii) The Lagrangian dual derived from the MIP presented in Chapter 2 can provide a tighter lower bound on the optimal objective value, $w^*$, than a simple linear relaxation of Equations (2.1)–(2.10) would.[3]

[2]This is possible when loops exist in the guidepath graph, or when agents are allowed to reverse direction. For example, consider the 3-agent problem instance depicted in Figure 1.1. When the single-agent constraints (given by Equations (2.2)–(2.6) and (2.10)) are linearly relaxed, fractional solutions may return a better makespan than what is achievable with integer solutions. For instance, consider the following (fractional) moves for Agent 1:

1. At time $t = 0$, the "mass" of Agent 1 at $s_1$ (i.e., edge 28) is 1 (that is, $x_{1,28,0} = 1$), and the mass placed on every other edge equals zero.
2. At time $t = 1$, the mass of Agent 1 is split; a mass of 0.5 resides on edge 27, and another mass of 0.5 resides on edge 30, with zero mass on every other edge (i.e., $x_{1,27,1} = x_{1,30,1} = 0.5$). Note that edges 27 and 30 are both considered "neighbors" of one another, with $edge\ \ 27 \in (edge\ \ 30)^\bullet$, and $edge\ \ 30 \in (edge\ \ 27)^\bullet$.
3. At time $t = 2$, masses of 0.2 and 0.3 are allocated to edges 27 and 30, respectively, and a mass of 0.5 is placed upon edge 16, which is $d_1$.

Obviously, this occurrence, which we call "teleportation", is not physically realizable. Nor is it permitted within the unrelaxed integer program. However, such solutions do lie within the feasible space of the linear relaxation of Equations (2.2)–(2.6) and (2.10), and, given an objective function that measures makespan, the ability to teleport a fraction of an agent directly into its destination makes it possible to return solutions that are superior to anything that is achievable when Equations (2.2)–(2.6) and (2.10) are not linearly relaxed.

[3]Teleportation can be prevented, however, by modifying the MIP of Chapter 2 in a manner that is reflective of the difference between Equations (A.10) and (A.15); that is, we can replace Equation (2.6) with the following:

$$\forall a \in \mathcal{A},\ \forall e \in E,\ \forall t \in \mathcal{T} \setminus \{T\},\ \ x_{a,e,t} = x_{a,e,t+1} + \sum_{e' \in e^\bullet} x_{a,e',t+1}. \tag{A.21}$$

The equality of Equation (A.21) effectively prevents any fraction of an agent from teleporting across part of the guidepath graph when the integrality constraints are relaxed, resulting in integer-valued extreme points for the linear relaxation of Equations (2.2)–(2.5), (2.10), and (A.21). Furthermore, since the original, unrelaxed MIP of Chapter 2 does not permit teleportation, the substitution of Equation (A.21) for Equation (2.6) does not eliminate any of the integer-valued solutions of the original MIP of Chapter 2. The exchange of Equation (A.21) for Equation (2.6) in the MIP of Chapter 2 therefore results in a new MIP that possesses the integrality property described in [22] and also has the same optimal objective value, $w^*$, as the original MIP of Chapter 2. Together, these properties imply that the linear relaxation of the revised MIP would provide a lower bound on

The MIP of Equations (A.5)–(A.9), Equations (A.11)–(A.14), and Equation (A.15) can be streamlined, much like the MIP of Equations (2.1)–(2.10) was, by using a spatio-temporal graph like the one shown in Figure 4.2, to eliminate extraneous decision variables (i.e., variables that must equal zero). In the case of the MIP of Equations (2.1)–(2.10), non-extraneous decision variables correspond to the nodes of spatio-temporal graphs such as the one shown in Figure 4.2. In the case of the MIP described in this appendix, the non-extraneous variables correspond instead to the *arcs* of such graphs.

In fact, a linear relaxation of the MIP of Equations (A.5)–(A.9), Equations (A.11)–(A.14), and Equation (A.15), streamlined based upon efficient representations such as the one described by Figure 4.2, is effectively described by the LP of Equations (3.74)–(3.76). As section 3.3 demonstrates, the dual of the LP of equations (3.74)–(3.76) can then be used to find Lagrange multipliers corresponding to the "coupling" constraints.

the optimal objective for the original MIP of Chapter 2 that would be identical to the one obtained from the Lagrangian relaxation of Equations (2.1)–(2.10).

## REFERENCES

[1] J. Adams, E. Balas, and D. Zawack, “The Shifting Bottleneck Procedure for Job Shop Scheduling,” *Management Science*, vol. 34, pp. 391–401, 1988.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. 1*. Belmont, MA: Athena Scientific, 1995.

[4] ——, *Nonlinear Programming (2nd ed.)* Belmont, MA: Athena Scientific, 1999.

[5] Y. A. Bozer and M. M. Srinivasan, “Tandem configurations for automated guided vehicle systems and the analysis of single vehicle loops,” *IIE Trans.*, vol. 23, pp. 72–82, 1991.

[6] P. Brucker, B. Jurisch, and B. Sievers, “A branch and bound algorithm for the job-shop scheduling problem,” *Discrete Applied Mathematics*, vol. 49, no. 1, pp. 107 –127, 1994.

[7] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems (2nd ed.)* NY,NY: Springer, 2008.

[8] G. Daugherty, S. Reveliotis, and G. Mohler, “A customized dual ascent algorithm for a class of traffic coordination,” ISyE, Georgia Institute of Technology (submitted for publication), Tech. Rep., 2016.

[9] ——, “Some novel traffic coordination problems and their analytical study based on lagrangian duality theory,” in *Proceedings of the 55th IEEE Conf. on Decision and Control (CDC 2016)*, IEEE, 2016, pp. 1701–1708.

[10] ——, “Optimized multi-agent routing in guidepath networks,” in *Proceedings of the 20th World Congress of the International Federation of Automatic Control (IFAC)*, 2017.

[11] ——, “Optimized multi-agent routing for a class of guidepath-based transport systems,” *IEEE Transactions on Automation Science and Engineering*, (submitted for publication).

[12] G. Desaulniers, A. Langevin, D. Riopel, and B. Villeneuve, "Dispatching and conflict-free routing of automated guided vehicles: An exact approach," *The Intl. Jrnl of Flexible Manufacturing Systems*, vol. 15, pp. 309–331, 2003.

[13] E. W. Dijkstra, "Cooperating sequential processes," Technological University, Eindhoven, Netherlands, Tech. Rep., 1965.

[14] J. Ezpeleta and J. M. Colom, "Automatic synthesis of colored Petri nets for the control of FMS," *IEEE Trans. on R&A*, vol. 13, pp. 327–337, 1997.

[15] J. Ezpeleta, F. Tricas, F. Garcia-Valles, and J. M. Colom, "A Banker's solution for deadlock avoidance in FMS with flexible routing and multi-resource states," *IEEE Trans. on R&A*, vol. 18, pp. 621–625, 2002.

[16] C. I. Fábián and Z. Szőke, "Solving two-stage stochastic programming problems with level decomposition," *Computational Management Science*, vol. 4, no. 4, pp. 313–353, 2007.

[17] M. P. Fanti, "Event-based controller to avoid deadlock and collisions in zone-controlled AGVS," *Inlt. Jrnl Prod. Res.*, vol. 40, pp. 1453–1478, 2002.

[18] M. L. Fisher, "Optimal solutions of scheduling problems using Lagrange multipliers: Part I," *Operations Research*, vol. 21, pp. 1114–1127, 1973.

[19] ——, "The Lagrangian relaxation method for solving integer programming problems," *Management Science*, vol. 27, pp. 1–18, 1981.

[20] T. Ganesharajah, N. G. Hall, and C. Sriskandarajah, "Design and operational issues in AGV-served manufacturing systems," *Annals of OR*, vol. 76, pp. 109–154, 1998.

[21] E. Gawrilow, E. Köhler, R. H. Möhring, and B. Stenzel, "Dynamic routing of automated guided vehicles in real-time," in *Mathematics–Key Technology for the Future*, Springer, 2008, pp. 165–177.

[22] A. M. Geoffrion, "Lagrangian relaxation for integer programming," *Math. Programming Studies*, vol. 2, pp. 82–114, 1974.

[23] A. Giua, M. P. Fanti, and C. Seatzu, "Monitor design for colored Petri nets: An application to deadlock prevention in railway networks," *Control Engineering Practice*, vol. 10, pp. 1231–1247, 2006.

[24] S. S. Heragu, *Facilities Design (3rd ed.)* CRC Press, 2008.

[25] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex analysis and minimization algorithms. I*, ser. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin, 1993, vol. 305.

[26] ——, *Convex analysis and minimization algorithms. II*, ser. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin, 1993, vol. 305.

[27] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati, "A practical approach to job-shop scheduling problems," *IEEE Trans. on Robotics & Automation*, vol. 9, pp. 1–13, 1993.

[28] J. Huang, U. S. Palekar, and S. G. Kapoor, "A labeling algorithm for the navigation of automated guided vehicles," *Journ. of Eng. for Industry*, vol. 115, pp. 315–321, 1993.

[29] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proc. IEEE Symp. Found. Comput. Sci.*, IEEE, 1984, pp. 241–250.

[30] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, "Developing conflict-free routes for automated guided vehicles," *Oper. Res.*, vol. 41, pp. 1077–1090, 1993.

[31] M. Lawley, S. Reveliotis, and P. Ferreira, "The application and evaluation of Banker's algorithm for deadlock-free buffer space allocation in flexible manufacturing systems," *Intl. Jrnl. of Flexible Manufacturing Systems*, vol. 10, pp. 73–100, 1998.

[32] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov, "New variants of bundle methods," *Mathematical Programming*, vol. 69, no. 1, pp. 111–147, 1995.

[33] Z. Li, M. Zhou, and N. Wu, "A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems," *IEEE Trans. Systems, Man and Cybernetics – Part C: Applications and Reviews*, vol. 38, pp. 173–188, 2008.

[34] Z. W. Li and M. C. Zhou, "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems," *IEEE Trans. on SMC – Part A*, vol. 34, pp. 38–51, 2004.

[35] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming (4th ed.)* NY, NY: Springer, 2016.

[36] W. L. Maxwell and J. A. Muckstadt, "Design of automatic guided vehicle systems," *IIE Trans.*, vol. 14, pp. 114–124, 1982.

[37] E. B. Nababan, A. R. Hamdan, S. Abdullah, and M. S. Zakaria, “Branch and bound algorithm in optimizing job shop scheduling problems,” in *Information Technology, 2008. ITSim 2008. International Symposium on*, IEEE, vol. 1, 2008, pp. 1–5.

[38] A. Nazeem and S. Reveliotis, “A practical approach for maximally permissive liveness-enforcing supervision of complex resource allocation systems,” *IEEE Trans. on Automation Science and Engineering*, vol. 8, pp. 766–779, 2011.

[39] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, UK: Cambridge University Press, 2010.

[40] I. M. Ovacik and R. Uzsoy, *Decomposition Methods for Complex Factory Scheduling Problems*. Boston: Kluwer Academic Publ., 1997.

[41] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY: Dover, 1998.

[42] J. Park and S. A. Reveliotis, “Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings,” *IEEE Trans. on Automatic Control*, vol. 46, pp. 1572–1583, 2001.

[43] M. Pinedo, *Scheduling*. Upper Saddle River, NJ: Prentice Hall, 2002.

[44] L. Qui, W.-J. Hsu, S.-Y. Huang, and H. Wang, “Scheduling and routing algorithms for AGVs: A survey,” *IJPR*, vol. 40, pp. 745–760, 2002.

[45] S. Reveliotis, *Logical Control of Complex Resource Allocation Systems*. NOW Series on Foundations, Trends in Systems, and Control, vol. 4 (1–2), 2017.

[46] S. Reveliotis and E. Roszkowska, “On the complexity of maximally permissive deadlock avoidance in multi-vehicle traffic systems,” *IEEE Trans. on Automatic Control*, vol. 55, pp. 1646–1651, 2010.

[47] S. A. Reveliotis, “Conflict resolution in AGV systems,” *IIE Trans.*, vol. 32(7), pp. 647–659, 2000.

[48] ——, *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach*. NY, NY: Springer, 2005.

[49] S. A. Reveliotis and P. M. Ferreira, “Deadlock avoidance policies for automated manufacturing cells,” *IEEE Trans. on Robotics & Automation*, vol. 12, pp. 845–857, 1996.

[50] E. Roszkowska and S. Reveliotis, “On the liveness of guidepath-based, zoned-controlled, dynamically routed, closed traffic systems,” *IEEE Trans. on Automatic Control*, vol. 53, pp. 1689–1695, 2008.

[51] Q. Sajid, R. Luna, and K. E. Bekris, “Multi-agent path finding with simultaneous execution of single-agent primitives,” in *5th Symposium on Combinatorial Search*, Niagara Falls, ON, Canada, 2012.

[52] H. D. Sherali, S. C. Sarin, and R. Desai, “Models and algorithms for job selection, routing and scheduling in a flexible manufacturing system,” *Annals of Operations Research*, vol. 26, pp. 433–453, 1990.

[53] D. Silver, “Cooperative pathfinding,” in *Proceedings of the 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 23–28.

[54] T. Stanley and R. Korf, “Complete algorithms for cooperative pathfinding problems,” in *Proc. 22nd Intl. Joint Conf. Artif. Intell.*, 2011.

[55] P. Surynek, “Towards optimal cooperative path planning in hard setups through satisfiability solving,” in *Proc. 12th Pacific Rim Int. Conf. Artif. Intell.*, 2012, pp. 564–576.

[56] I. F. A. Vis, “Survey of research in the design and control of automated guided vehicle systems,” *European Journal of Operational Research*, vol. 170, pp. 677–709, 2006.

[57] N. Viswanadham, Y. Narahari, and T. L. Johnson, “Deadlock avoidance in flexible manufacturing systems using Petri net models,” *IEEE Trans. on Robotics and Automation*, vol. 6, pp. 713–722, 1990.

[58] R. M. Wilson, “Graph puzzles, homotopy, and the alternating group,” *Journal of Combinatorial Theory, B*, vol. 16, pp. 86–96, 1974.

[59] W. L. Winston, *Introduction To Mathematical Programming: Applications and Algorithms, 2nd ed.* Belmont, CA: Duxbury Press, 1995.

[60] L. A. Wolsey, *Integer Programming*. NY, NY: John Wiley & Sons, 1998.

[61] N. Wu and M. Zhou, “Resource-oriented Petri nets in deadlock avoidance of AGV systems,” in *Proceedings of the ICRA’01*, IEEE, Seoul, Korea, 2001, pp. 64–69.

[62] T. Yamada and R. Nakano, “Genetic algorithms for job-shop scheduling problems,” in *Proceedings of Modern Heuristic for Decision Support*, UNICOM seminar, London, 1997, pp. 67–81.

[63] J. Yu and S. M. LaValle, “Optimal multi-robot path planning on graphs: Structure and computational complexity,” *ArXiv: http://arxiv.org/pdf/1507.03289v1.pdf*, 2015.

[64] ——, “Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics,” *IEEE Trans. on Robotics*, vol. 32, pp. 1163–1177, 2016.

[65] J. Yu and D. Rus, “Pebble motion on graphs with rotations: Efficient feasibility tests and planning,” in *Workshop on the Algorithmic Foundations of Robotics*, Istanbul, Turkey, 2014.

[66] M. Zhou and M. P. Fanti (editors), *Deadlock Resolution in Computer-Integrated Systems*. Singapore: Marcel Dekker, Inc., 2004.