Online Disjoint Vehicle Routing with Application to AGV Routing

vorgelegt von
Dipl.-Math. oec. Björn Stenzel

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Fredi Tröltzsch
Berichter:	Prof. Dr. Rolf H. Möhring
	Prof. Dr. Ekkehard Köhler
Zusätzlicher Berichter:	Dr. Andreas Parra

Tag der wissenschaftlichen Aussprache: 8. Juli 2008

Berlin 2008
D 83

Acknowledgements

I wish to thank many people for their support. In particular, I am grateful to Rolf Möhring for his trust and encouragement, for offering me the possibility to work in a fertile environment, and for supervising my thesis. In addition, I am especially indebted to him for arousing my interest for discrete applied mathematics.

I also want to thank Ekkehard Köhler for his fruitful hints and for taking the second assessment of the thesis. His motivating character helped me a lot.

Furthermore, I thank Andreas Parra from the Hamburger Hafen und Logistik AG (HHLA) for his trust in our ideas and for assessing the thesis from a practical point of view. I also thank Kai-Uwe Riedemann from HHLA and Boris Wulff from Container Terminal Altenwerder (CTA) for interesting discussions on AGV routing.

Moreover, I am greatly indebted to my collaborators on material in this thesis, Ewgenij Gawrilow, Max Klimm, Ekkehard Köhler, Magnus Kühne, and Rolf Möhring. It was a pleasure to work with them. Moreover, I enjoyed the discussions with Ines Spenke on discrete dynamic flows.

Many thanks go to my colleagues from the COGA group for the friendly and motivating environment. In particular, I am grateful to my roommates Felix König, Heiko Schilling, and Gregor Wünsch for an inspiring working atmosphere in our office. I also wish to thank Ewgenij Gawrilow for his help and valuable hints on programming issues. Moreover, I thank Marco Lübbecke, Sebastian Stiller, and Gregor for carefully reading parts of the thesis.

Special thanks go to my family. In particular, I am grateful to my parents for their unlimited support and their believe in me. Finally, I would like to thank Mandy Stuckert for patient listening and for encouraging me in moments of frustration.

My research was funded by the German Federal Ministry of Education and Research (BMBF) under grants no. 03-MONJB1.

Berlin, May 2008 *Björn Stenzel*

Contents

Chapter 1

Introduction

There are several 'Vehicle Routing Problems' that have been intensively studied in the recent years [5, 39]. The common aim of all these problems is the assignment of vehicles to transportation requests (items) and the determination of the order in which these requests should be served. They only vary in the considered objectives and constraints. Since it is assumed that vehicles are small in relation to the given (street) network interdependencies between the vehicles are not taken into account.

In contrast, we investigate a *microscopic vehicle routing model*. Here, 'microscopic' means that the vehicles are rather large-sized in relation to the underlying network. Therefore, we have to take care for conflicts/collisions between the vehicles that serve the transportation requests. In fact, we focus on the determination of paths (routes) such that the vehicles that execute these paths do not conflict with each other. We call such paths *disjoint* and provide two disjointness models. Since the time-dependent behavior of the vehicles have to be taken into account in this context we consider paths over time, which are also called *dynamic paths*.

Furthermore, we are interested in an *online setting* within this model, i.e., we assume that requests appear one-by-one and no information about further requests is known, which is often the case in real-world problems. Therefore, we investigate *online disjoint dynamic paths* in this thesis. More precisely, we introduce two optimization problems: the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP) and the Online Quickest Disjoint Paths Problem (OQDPP). Both problems only differ in their objective. Regarding the OSDDPP we aim at minimizing the sum of durations over all requests, while the objective of the OQDPP is the so-called makespan, which is the time when all requests are served. Note that dispatching of vehicles, i.e., the assignment of transportation requests to vehicles, is not considered.

In contrast to standard disjoint path problems (without a time component) that have been extensively investigated in the offline [33] as well as in the online setting [3, 6, 8], these online disjoint path problems have not been studied before. Krishnamatury, Batta and Karwan [28] discussed the OQDPP in the offline case, where all requests are known right from the beginning. Moreover, Spenke [37] showed that this problem is $\mathcal{NP}$-hard even

in grid graphs and provided an approximation algorithm.

In this work we investigate algorithms for the OSDDPP and the OQDPP. Thereby, we distinguish between dynamic approaches where time dependences are taken into account during the route computation and static approaches where this is done at the execution time of the routes via a reservation procedure.

Besides analyzing these algorithms theoretically we pay special attention to their real-life suitability and performance since the considered problems are of practical relevance. Actually, conflict-free routing of vehicles is a common problem in many applications, when vehicles move on tracks or track like lanes as for instance switching engines at private or public cargo railroads, aircrafts on large-sized airports, or Automated Guided Vehicles (AGVs) in logistic systems like container terminals, production facilities or warehouses. In all these applications the guidance of the vehicles is the key to an efficient transportation system that aims at maximizing its throughput. In particular, automation of large scale logistic systems nowadays is an important method for improving productivity. Vis [40] provides a survey on design and control of such systems.

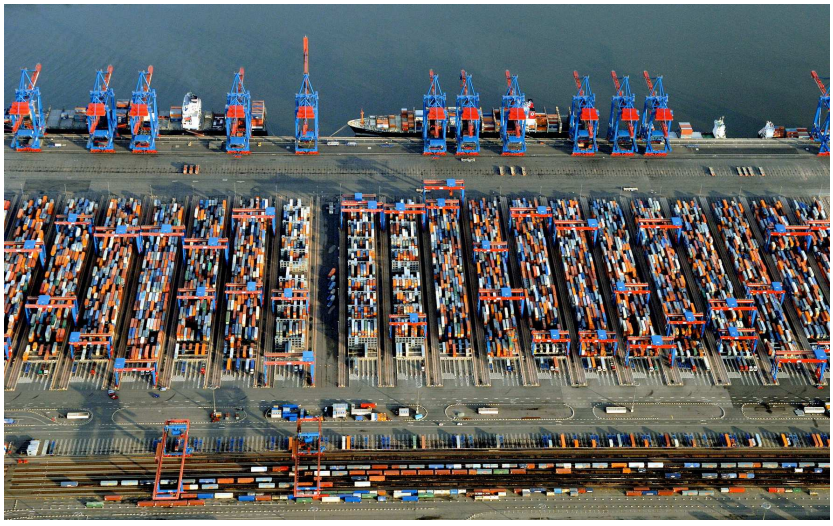


Figure 1.1: The HHLA Container Terminal Altenwerder (CTA). ©HHLA

We focus on the routing of AGVs in an automated logistic system for the evaluation of our algorithms. In fact, we conduct experiments based on a simulation of Container Terminal Altenwerder (Figure 1.1), a container terminal at Hamburg Harbor which is operated by the Hamburger Hafen und Logistik AG (HHLA). There, the AGVs transport containers between the container bridges that load and unload the ships and the storage areas. The

transportation requests usually arrive sequentially (online) and are already assigned to a particular vehicle. Hence, this application is perfectly in line with our model.

Outline of the Thesis

In this work we aim at both, analyzing algorithms for the considered problems, the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP) and the Online Quickest Disjoint Paths Problem (OQDPP), theoretically and providing real-life solutions, namely fast algorithms that perform good in practice. Apart from some preliminaries (*Chapter 2*), including the introduction of our model in Section 2.2, we present three main chapters.

In *Chapter 3* we introduce a dynamic routing algorithm for the OSDDPP and the OQDPP, respectively, i.e., an algorithm that takes the time-dependent behavior of the vehicles into account. For each incoming request our polynomial-time algorithm DYN-ROUTE determines a quickest path that respects the reservations of the already computed paths. This approach is motivated by dynamic flow theory [14, 15, 27] and several papers on the Shortest Path Problem with Time-Windows [11, 12, 13, 32].

Besides the basic routing algorithm we introduce some additional methods for dealing with practical requirements. In particular, we provide rerouting strategies to cope with perturbations and the prioritization of requests. The evaluation with respect to our application, the HHLA Container Terminal Altenwerder (CTA), shows that our approach is suitable for practical use in general, that is, the requests can be served in real-time—each route computation in the underlying graph with about 45,000 edges takes less than a tenth of a second on average—and additional real-life issues can also be modeled within this approach.

In *Chapter 4* we focus on the evaluation of the dynamic routing algorithm DYN-ROUTE presented in Chapter 3 in comparison with optimal solutions as well as with other routing approaches. In particular, we provide another dynamic routing algorithm (DYN-ROUTE-SP) for the purpose of comparison. In addition, we consider the approximation algorithm of Spenke [37] for the offline variant of the OQDPP in grid graphs in this context. For the analysis we focus on both, theoretical worst case performance using competitive analysis, on the one hand, and empirical case studies, on the other hand. We pay special interest to unit grid graphs in both parts since the underlying graph at CTA is grid-like.

On the theoretical side we show that DYN-ROUTE is $\Theta(k)$-competitive with respect to both considered problems in directed as well as in undirected graphs. While the algorithm we introduce in this chapter (DYN-ROUTE-

SP) does not perform better in arbitrary graphs, we are able to show that it is $1 + (5k - 1)/L_{\max}$-competitive with respect to the OQDPP and $1 + (5k + 3)/2L_{\text{avg}}$-competitive with respect to the OSDDPP in grid graphs, where, roughly speaking, $L_{\max}$ and L_{avg} denote the maximum and the average shortest path distance over all requests, respectively. As a byproduct of the result concerning the OQDPP we improve the approximation ratio for the corresponding offline problem on grid graphs given by Spenke for instances with $L_{\max} > (4/3)k + 1$. We will see that this condition often holds in practice.

Concerning the experimental analysis we evaluate several instances in grid graphs. It turns out that DYN-ROUTE is superior to the other mentioned approaches. Moreover, regarding optimal (offline) solutions, we observe an average optimality gap of 1% to 17% for the OSDDPP and of 1% to 25% for the OQDPP—dependent on the size of the grid. The optimal solutions are determined using an integer programming approach. To this end, we formulate both offline problems as multi-commodity flow problems in a time-expanded graph.

In *Chapter 5* we present a different routing approach for the investigated problems. In fact, we ignore the time-dependent behavior of the vehicles during the route computation (static routing). Thus, the collision avoidance has to be considered independently of the routing by a certain reservation procedure. Similar approaches have been investigated in [10, 21, 26, 42]. We provide a two-stage approach that copes with the main problems of this approach—congestion and detours caused by the static route computation, on the one hand, and the risk of deadlocks during the reservation procedure, on the other hand.

Firstly, we give an asymptotically optimal algorithm with respect to a particular load balancing problem, the Online Load Balancing Problem with Bounded Stretch Factor, in order to distribute the vehicles over the network as well as possible while bounding the length of the determined paths. In detail, we compute a shortest path with respect to a specific (load-dependent) cost function for each request.

Secondly, we introduce an algorithm that computes a reservation schedule, which guarantees a deadlock-free execution of the determined routes. Here, we basically focus on the detection of specific cycles in a so-called deadlock detection graph that corresponds to the schedule.

Both algorithms together lead to the first known deadlock-free static routing approach that is suitable for the use in large-scale logistic systems. This is shown by accordant evaluations in the last part of the chapter. There, we also provide an experimental comparison to the dynamic routing algorithm DYN-ROUTE. It turns out that it highly depends on the expected traffic density, which of these approaches should be preferred in practice.

Chapter 2

Preliminaries

In this chapter we prepare the ground for the main part of the thesis. Firstly, in Section 2.1, we provide two basic definitions. In fact, we introduce a special graph class called *grid graphs* and a method that is typically used to analyze online problems, namely the *competitive analysis*. Both are fundamental for the following chapters. Afterwards we describe our *online disjoint vehicle routing model* in Section 2.2. Besides a general description of the considered problems and objectives we introduce two models for the recognition of conflicts, i.e., we define disjointness in different ways. In Section 2.3 we introduce our *application*: the routing of Automated Guided Vehicles at HHLA Container Terminal Altenwerder (CTA).

2.1 Basic Definitions

2.1.1 Grid Graphs

Due to the application we will introduce in Section 2.3 we pay special attention to the class of grid graphs. Although it might be intuitively clear how such a graph looks like we provide a formal definition (Definition 2.1) and illustrate a 10×4 grid graph in Figure 2.1.

Definition 2.1 (Grid graph, Burkard et al. [9])**.** *A graph $G = (V, E)$ is a grid graph if its nodes are in a one-to-one correspondence with the points in a Euclidian $n \times m$ unit grid and if nodes are adjacent if and only if the corresponding points are at a distance of one.*

We call the set of edges whose incident nodes correspond to points in the unit grid with the same first component a *vertical lane*. Accordingly, a *horizontal lane* is the union of all edges between nodes with same second coordinate. Thus, a $n \times m$ grid graph has n vertical and m horizontal lanes. In this work the ratio n/m of the number n of vertical lanes and the number m of horizontal lanes is of special interest. We call this aspect ratio the *narrowness* of a grid graph.

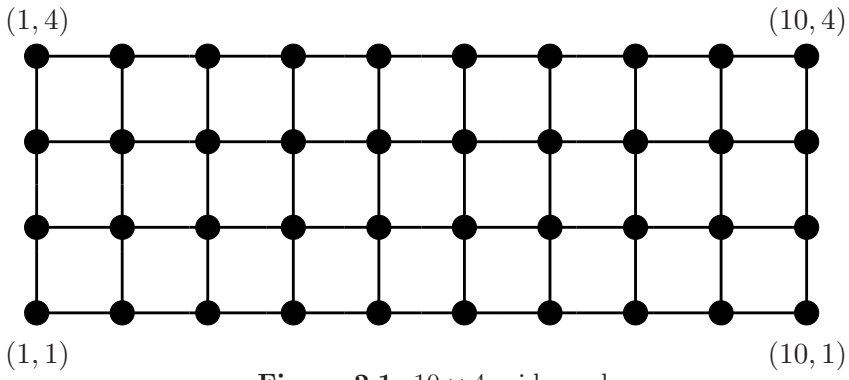


Figure 2.1: 10×4 grid graph

2.1.2 Online Problems and Competitive Analysis

Optimization problems are called online if the problem input becomes known only piecewise. Each request in such a sequence has to be served by an online algorithm. We consider an online model where requests have to be answered without knowledge of further requests. This model is called list model.

Online algorithms are typically analyzed using *competitive analysis*, in which the worst-case performance of the algorithm is compared with an optimal offline solution, i.e., with the best solution that can be obtained if the complete input is known from the beginning. This kind of analysis was introduced by Sleator and Tarjan [36] while Karlin, Manasse, Rudolph, and Sleator [24] were the first who used the term competitive analysis.

Definition 2.2 (Competitive ratio)**.** *An online algorithm* ALG *for a minimization problem is called* c-competitive *if, for any problem instance* $\mathcal{I}$*, it achieves a solution with*

$$\mathrm{ALG}(\mathcal{I}) \leq c \cdot \mathrm{OPT}(\mathcal{I}),$$

where $\mathrm{OPT}(\mathcal{I})$ *denotes the value of the optimal offline solution for that instance.*

The competitive ratio *of* ALG *is the infimum over all* c *such that* ALG *is* c*-competitive.*

Since all problems considered in this thesis are minimization problems we omit a similar definition of the competitive ratio for maximization problems.

Note that the competitive analysis does not say anything about the computational complexity of an online algorithm. Moreover, an c-competitive algorithm does not need to be efficient. However, if this is the case, we can directly conclude that this algorithm is a c-approximation.

2.2 Online Disjoint Vehicle Routing

2.2.1 Problem Description

We consider a graph $G = (V, E)$. The edge set E denotes the lanes of the underlying traffic network. Each edge $e \in E$ has a certain transit time $\tau(e)$ which indicates the time needed to traverse this edge. The node set V models the crossings of the lanes. The graph is either directed (unidirectional lanes) or undirected. In both cases we assume that there exists no (anti-) parallel edges. In order to avoid similar definitions for both graph classes we refer to an undirected edge $\{u, v\}$ as well as to a directed edge (u, v) as uv in this section. In the main part of the thesis we clearly indicate whenever we restrict ourselves to one of these graph classes. Note that grid graphs are undirected by definition.

Transportation tasks are consecutively arriving over time and are modeled by a sequence $\sigma = r_1, \ldots, r_k$ of requests. Each request $r_i = (s_i, t_i, \theta_i)$ consists of a source node s_i, an target node t_i, and a release time θ_i which can be interpreted as earliest possible starting time. The requests do not necessarily arrive chronologically, i.e., an arising request can contain a release time smaller than its predecessor. Note that we do not consider the assignment of vehicles to requests in general. In contrast, we assume that this is done by a higher-level management system in advance. In Section 3.1.2, however, we provide an approach for this task (see Remark 3.4) using the dynamic routing algorithm described in Section 3.1.

Since we use the online list model, cf. Section 2.1.2, each request has to be answered immediately. This is done by a corresponding path over time. According to dynamic flow theory [14, 15, 27], where flows over time are often called dynamic flows, we call such paths *dynamic paths*. The key property of such a path is that waiting on edges is permitted for an arbitrary time while waiting in nodes is forbidden.

Definition 2.3 (Dynamic path)**.** *A* dynamic path *in a graph $G = (V, E)$ with transit times $\tau(e)$ on each edge $e \in E$ is a sequence*

$$P = (\theta_0, (v_1, \theta_1), \ldots, (v_n, \theta_n))$$

of nodes $v_1, \ldots, v_n$ with $v_i v_{i+1} \in E$ for all $i \in \{1, \ldots, n-1\}$ and time stamps $\theta_0, \theta_1, \ldots, \theta_n$.

While θ_0 and $\theta_1 \geq \theta_0$ denote the release time *and the* starting time*, respectively,*

$$\theta_i \geq \theta_{i-1} + \tau(v_{i-1}v_i) \ (2 \leq i \leq n)$$

is the time when the corresponding node v_i is entered.

The duration Δ_P *of a dynamic path* P *is defined as the difference between the* completion time θ_n *and the release time* θ_0.

Remark 2.4 (Static path)**.** *In order to avoid confusion we will use the term* static path *to denote standard paths, i.e., paths without a time component as introduced in [33], whenever the notation would be misleading otherwise. The shortest path distance of a static* s*-*t*-path with respect to a given transit time function* τ *is denoted by* $\text{dist}_\tau(s,t)$*. In graphs with unit transit times we write* $\text{dist}(s,t)$*.*

A dynamic path reserves edges and nodes at certain times. Nodes are reserved at the time they are entered and edges are occupied while the vehicles travels over this edge. These points in time or intervals, respectively, are called reservations.

Definition 2.5 (Reservation of nodes and edges)**.** *Consider a dynamic path* $P = (\theta_0, (v_1, \theta_1), \ldots, (v_n, \theta_n))$ *in a graph* $G = (V, E)$*. For each* $i \in \{2, \ldots, n\}$ *the time interval* (θ_{i-1}, θ_i) *is called a* reservation of edge $v_{i-1}v_i$ *on path* P *and the point in time* θ_i *is called* reservation of node v_i *on path* P*. The set of reservations of an edge* e *(of a node* v*) on a path* P *is denoted by* $\mathcal{R}_P(e)$ *(*$\mathcal{R}_P(v)$*).*

Note that there are no reservations before the starting time since we assume that there are no conflicts at the source node of a request.

Assumption 2.6. *We assume that sources and targets of the transportation requests (depots, pick-up and delivery points) are located such that vehicles that are positioned at these positions do not interfere with other vehicles. Thus, in our model, a vehicle does not enter the graph until the starting time of the corresponding path. We omit the explicit introduction of dummy nodes and edges, but keep in mind that this would be possible.*

Concerning the waiting times we assume, w.l.o.g., that waiting occurs at the end of an edge, i.e., a vehicle traverses the edge first and waits afterwards at the end of the edge. We define waiting intervals accordingly. Moreover, we call the time period from the release time to the starting time a waiting interval, although this interval does not lead to any reservation.

Definition 2.7 (Waiting intervals on a dynamic path)**.** *Consider a dynamic path* $P = (\theta_0, (v_1, \theta_1), \ldots, (v_n, \theta_n))$ *in a graph* $G = (V, E)$*. For each* $i \in \{2, \ldots, n\}$ *the time interval* $(\theta_{i-1} + \tau(v_{i-1}v_i), \theta_i)$ *is called* waiting interval *of path* P *on edge* $v_{i-1}v_i$*.*

Now we are going to introduce two optimization problems in this context. In both formulations we aim at determining *disjoint* dynamic paths. For the definition of disjointness we refer to Section 2.2.2 where we discuss two different models.

The problems only differ in the considered objective. The first one is to minimize the overall duration, i.e., the sum of durations over all requests. The idea behind that objective is to achieve a minimal average operating time of the vehicles in order to make them available for the higher-level management system as quickly as possible. We call the optimization problem the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP).

Online Shortest Dynamic Disjoint Paths Problem (OSDDPP)

Instance:	Graph $G = (V, E)$, sequence of requests $\sigma = r_1, \ldots, r_k$ with $r_i = (s_i, t_i, \theta_i)$.
Task:	Find a corresponding set of disjoint dynamic paths $P_1, \ldots, P_k$ with minimal sum of durations (overall duration), i.e., minimize $\sum_{i=1}^{k} \Delta_{P_i}$.

The second objective is the so-called makespan which is the time when all requests are served. In this case we aim at completing a task that consists of a set of requests as early as possible. We call the corresponding optimization problem the Online Quickest Disjoint Paths Problem (OQDPP).

Online Quickest Disjoint Paths Problem (OQDPP)

Instance:	Graph $G = (V, E)$, sequence of requests $\sigma = r_1, \ldots, r_k$ with $r_i = (s_i, t_i, \theta_i)$.
Task:	Find a corresponding set of disjoint dynamic paths $P_1, \ldots, P_k$ with minimal makespan, i.e., with minimal maximum completion time over all paths.

Table 2.1 illustrates an overview of the considered problems and their offline variants, which are denoted accordingly.

So far only Spenke [37] as well as Krishnamatury, Batta and Karwan [28] considered one of these problems: the QDPP. While Krishnamatury et al. solved a mixed integer program using column generation, Spenke focused on analyzing the complexity of the QDPP. In fact, she concentrated on a multi-commodity flow problem that contains this problem as a special case (demand one for each commodity). She showed that the QDPP is $\mathcal{NP}$-hard

Disjoint Vehicle Routing		
objective	overall duration	makespan
online problem	OSDDPP	OQDPP
offline problem	SDDPP	QDPP

Table 2.1: Considered disjoint vehicle routing problems.

and provided a $4 + (k-4)/L_{\max}$-approximation algorithm for grid graphs, where $L_{\max} := \max_{r_i}\{\theta_i + \text{dist}(s_i, t_i)\}$ denotes a trivial lower bound to the makespan and k is the number of requests.

2.2.2 Disjointness

In the presented problem formulations we require the determined dynamic paths to be disjoint. Since this can be defined in different ways we describe what we mean by disjointness of dynamic paths.

We consider two different models for the definition of disjointness, an (rather theoretical) *idealized model* for graphs with unit edge length and a more precise real-life model, the *polygon model*, that takes arbitrary transit times and the dimensions of the vehicles into account. Both models are formulated independent of the graph class (directed or undirected). In the last part of the section we provide a transformation from undirected graphs to directed graphs that maintains the disjointness condition.

Idealized Model

In this model we assume unit transit times and demand that the waiting times are integral. This is motivated by the idea that the lanes of the network are divided into edges of the same length (for example the vehicle length) and that the vehicles travel with uniform speed. Spenke [37] also uses this model.

Disjointness is defined via the reservations of the dynamic paths. We say that paths P and P' are disjoint if the corresponding reservations on nodes and edges are disjoint.

Definition 2.8 (Disjointness in the idealized model)**.** *Two dynamic paths P and P' are* disjoint *if the corresponding reservations are disjoint, that is,*

$$\mathcal{R}_P(e) \cap \mathcal{R}_{P'}(e) = \emptyset \text{ and } \mathcal{R}_P(v) \cap \mathcal{R}_{P'}(v) = \emptyset$$

for all $e \in E$ and for all $v \in V$.

Note that, due to the integral transit and waiting times in this model, the representation of the reservation intervals can be simplified.

Remark 2.9 (Unit length intervals)**.** *In the idealized model the reservation intervals and therefore the waiting intervals can be subdivided into intervals of unit length.*

Polygon Model

In the idealized model each vehicle is modeled as an infinitesimally small mass point. By contrast, in a real-life model the physical dimensions of the vehicles have to be taken into account. The vehicles usually have to claim several edges of the graph at the same time, i.e., if a vehicle traverses or stands on an edge e, it possibly affects a much larger portion of the network than only edge e.

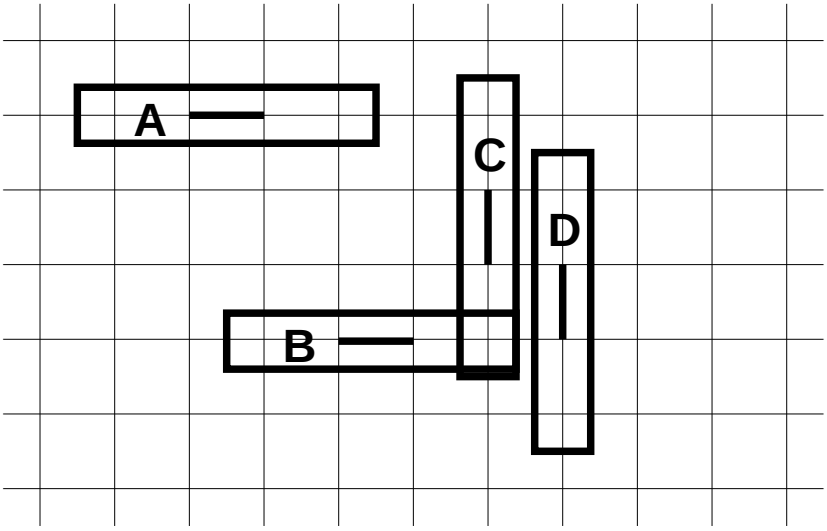


Figure 2.2: Illustration of polygons that are claimed by a vehicle that moves on the indicated edge. Polygon B and C intersect each other while the polygons A and D do not intersect another polygon.

For dealing with the physical dimensions of the vehicles we use polygons $\mathcal{P}(e)$ for each edge e, which describe the blocked area when a vehicle (the center of a vehicle) is located on edge e (Figure 2.2). Thus, it is prohibited to use two edges at the same time if the corresponding polygons intersect. We represent this mutual exclusion by sets $\mathit{confl}(e)$ of so-called *conflicting edges* for each edge e.

Definition 2.10 (Conflicting edges)**.** *We call two edges e and f* conflicting edges *if the corresponding polygons $P(e)$ and $P(f)$ intersect, i.e., if*

$$P(e) \cap P(f) \neq \emptyset.$$

For each edge $e \in E$ we denote the set of conflicting edges by $\mathrm{confl}(e)$*.*

We define the disjointness of dynamic paths accordingly, i.e., for each edge e on a certain dynamic path it must hold that no conflicting edge is used by another path simultaneously. Note that there is no need to take care for node disjointness since the polygons that are associated with an edge contain its adjacent nodes anyway.

Definition 2.11 (Disjointness in the polygon model)**.** *Two dynamic paths P and P' are* disjoint *if*

$$\mathcal{R}_P(e) \cap \left(\bigcup_{f \in \mathrm{confl}(e)} \mathcal{R}_{P'}(f) \right) = \emptyset.$$

for all $e \in E$. Since confl *is symmetric this is obviously the case if and only if*

$$\mathcal{R}_{P'}(e) \cap \left(\bigcup_{f \in \mathrm{confl}(e)} \mathcal{R}_P(f) \right) = \emptyset.$$

for all $e \in E$.

Since one could also think of comparing the edges that are blocked (covered) by the corresponding polygons instead of using polygon intersections and conflicting edges, we conclude the discussion of our polygon model with two remarks concerning this alternative approach.

The first disadvantage would be that polygons often only block a fraction of an edge. Therefore, two polygons may partially cover the same edge without intersecting each other (see polygons C and D in Figure 2.2). Thus, it is not clear how we can define disjointness in this case. Furthermore, we will see that the use of conflicting edges is also of value from the algorithmic point of view. This is due to the fact that only the conflicting edges of one path are considered in Definition 2.11. In particular, we are able to compute routes without taking the physical dimensions of the vehicles into account. For details we refer to Section 3.1.1 , Section 5.2.2, and Section 5.3.4.

Transformation of Undirected Graphs to Directed Graphs

Since some of the algorithms presented in this thesis are based on directed graph formulations we show how an undirected graph is transformed into a directed graph maintaining the disjointness condition.

If we consider the polygon model such a transformation is trivial. We simply replace each undirected edge e by two anti-parallel directed edges e_1 and e_2. The edges e_1 and e_2, by definition, have the same corresponding polygon as e and thus the new set of conflicting edges lead to an equivalent formulation of the problem.

Spenke [37] gave a transformation for the idealized formulation . The disadvantage of her construction is that it allows additional movements. Nevertheless, we will use (and describe) a similar transformation in Section 4.2.1 to construct a time-expanded graph.

Apart from this case, we use an approach that transfers parts of the polygon model to the idealized model. In fact, the idea is to replace an undirected edge in the same way as described above for the polygon model and to use the concept of conflicting edges. In the idealized model each edge has exactly two conflicting edges: the edge itself and the corresponding anti-parallel edge. Node conflicts are considered in the same way as described for the undirected case.

2.3 Application

The HHLA Container Terminal Altenwerder (CTA) at Hamburg Harbor, which is operated by the Hamburger Hafen und Logistik AG (HHLA) is the most modern container terminal worldwide regarding the level of automation. In particular, the containers are transported between ship and storage area using so-called Automated Guided Vehicles (AGVs), see Figure 2.3.

The AGVs navigate through the harbor area using a transponder system and the routes are sent to them from a central control unit. AGVs are symmetric, i.e., they can travel in both of the two driving directions equally well and can also change directions on a route. A key property of the AGVs is that they do not have an on-board collision-control system; i.e., they cannot sense any obstacle on their anticipated route. To this end, the transmitted routes have to be conflict-free.

We apply our disjoint vehicle routing model to that application by introducing a particular ('narrow') *grid-like graph* that represents the AGV street network. This graph consists of roughly 10,000 edges and 5,000 nodes. Since the detailed layout of the CTA is confidential we will show the grid graph illustrated in Figure 2.4 whenever we want to refer to that graph.

For experiments based on the CTA layout we implemented a simula-

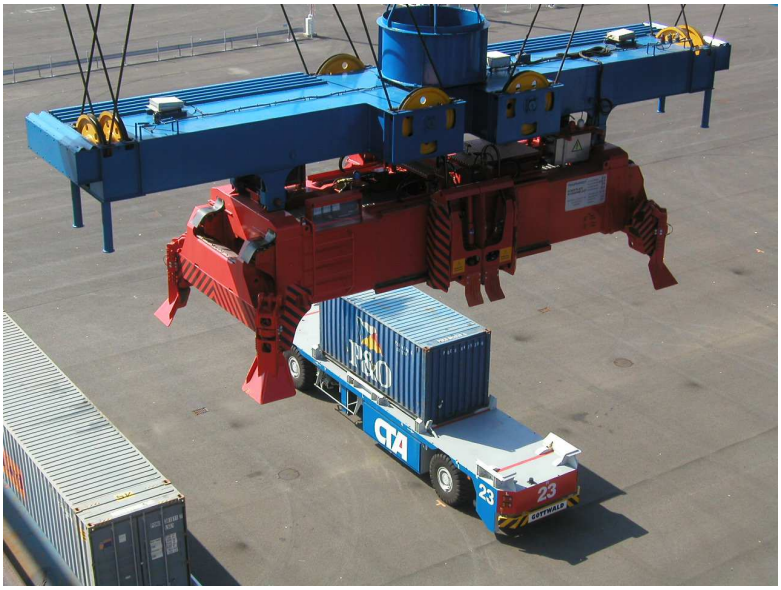


Figure 2.3: An Automated Guided Vehicle (AGV) used at HHLA Container Terminal Altenwerder. ©HHLA

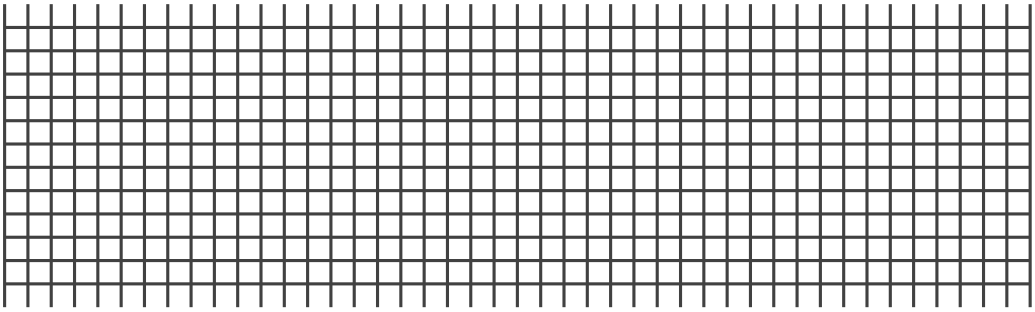


Figure 2.4: Abstract illustration of the underlying grid-like graph at HHLA Container Terminal Altenwerder (CTA). The exact layout is confidential.

tion environment that models the movement of the vehicles in detail, on the one hand, and generates requests, on the other. The latter is done by an integrated management system that assigns vehicles to requests/items by certain (confidential) rules. Note that this management system is not exactly the one used in practice. Therefore, we do not focus on objectives like the throughput or the number of containers transported in our experiments since such an evaluation would also depend on the assignment of the vehicles. In contrast, we evaluate our algorithms with respect to the introduced optimization problems.

Further Applications. The considered disjoint vehicle routing problems are also of interest in other applications. For instance, switching engines at private or public cargo railroads as well as Automated Guided Vehicles in production facilities, warehouses, or distribution centers have to be routed conflict-free. A new problem in this context is the guidance of (automated) aircraft tractors on public airports. Here, one aims at reducing the emissions, on the one hand, and coping with the increasing traffic, on the other.

Chapter 3

Dynamic Routing

In this chapter we present a dynamic routing approach for the online disjoint vehicle routing problems introduced in Section 2.2—the OSDDPP and the OQDPP. In the first part (Section 3.1) we focus on the description of the basic routing algorithm. Afterwards, in Section 3.2, we provide rerouting strategies to cope with perturbations and prioritization of requests. In both parts we consider the more precise disjointness model, namely the polygon model introduced in Section 2.2.2, and describe differences to the idealized setting whenever necessary. In Section 3.3 we evaluate the presented approach with respect to the application given in Section 2.3: the routing of Automated Guided Vehicles at HHLA Container Terminal Altenwerder. Parts of this chapter are published in [19, 20].

3.1 Routing Algorithm

In this section we introduce our *dynamic routing algorithm* for the OSDDPP and the OQDPP. After describing the structure of the iterative approach we focus on the essential parts of the algorithm in Section 3.1.2 and Section 3.1.3. In the final two subsections we focus on practical issues and show how we cope with arising difficulties in this context.

3.1.1 Iterative Routing Scheme

The dynamic routing algorithm is a kind of greedy approach. For each incoming request we compute a shortest path with respect to the elapsed time that respects the reservations of the already served requests, see Figure 3.1.

In fact, in our algorithm, we will not maintain the set of reservations, but the complementary set of free time-intervals $\mathcal{F}(e)$ on each edge $e \in E$, the so-called *time-windows*.

Maintaining these sets of intervals may be seen as a compact representation of the standard time-expanded graph [14, 15], in which there is a copy of each node/edge for each point in time (with respect to some time discretization). In contrast, the set of time-windows of an edge e only models those times, in which there is actually no vehicle on e. Similar compact rep-

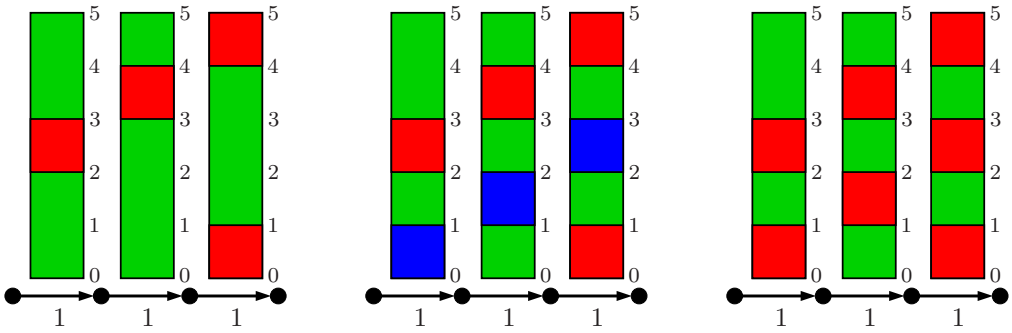


Figure 3.1: Illustration of the iterative routing scheme on three consecutive edges with transit time 1. (a) shows the situation before the new request arrives. There is a graph with some blockings (red) and some time-windows (green) on the time axis (y axis). The task is to compute a quickest path that respects the time-windows. This is illustrated in (b). The chosen path is blocked afterwards (see (c)). In this example we assume that the illustrated edges do not conflict with each other, i.e., the set of conflicting edges only contains the edge itself.

resentations of a time-expanded graph by time intervals have been studied before: While Desrochers et al. [11, 12, 13] as well as Sancho [32] investigate the Shortest Path Problem with Time-Windows, see Section 3.1.2, Kim and Tanchoco [25] consider the routing of Automated Guided Vehicles based on a simple model and provide a polynomial-time algorithm. In fact, they only represent reservations on nodes by time-windows while conflicts on edges have to be respected by additional checks. Moreover, they do not take the physical dimensions of the vehicles into account.

Algorithm 1: DYN-ROUTE

Data: Graph $G = (V, E)$, sequence of requests $\sigma = r_1, \ldots, r_k$.
Result: Sequence of dynamic paths $P_1, \ldots, P_k$.
begin
  foreach *request* r_j **do**
    · compute a dynamic path with minimum completion time that respects the given time-windows
    /∗ execute Algorithm 2 ∗/ ;
    · readjust the time-windows according to the computed path and the corresponding conflicting edges
    /∗ execute Algorithm 3 ∗/ ;
end

In contrast, as mentioned before, we use the polygon model introduced in

Section 2.2.2. Note that it is sufficient to consider the edges of a new path P itself (without the corresponding conflicting edges) to check for disjointness with already computed paths if all reservations (including the ones from conflicting edges) from these paths are represented by the given time-windows, cf. Definition 2.11. Thus, for each incoming request r_j we compute a shortest (w.r.t. the elapsed time) dynamic path that respects the given time-windows and readjust them afterwards according to the set of conflicting edges, see Algorithm 1 (DYN-ROUTE). Here, 'respecting' means that vehicles wait on an edge or traverse an edge e only during one of its 'free' time-windows given by $\mathcal{F}(e)$. For the rest of the thesis we refer to this algorithm as the *dynamic routing algorithm*.

In the following sections we will describe both parts of the algorithm in detail—the route computation and the readjustment of the time-windows.

3.1.2 Route Computation

The interesting part of the presented dynamic routing algorithm is the route computation for each incoming request, i.e., the determination of a shortest path that respects the given time-windows. We call this problem the Quickest Path Problem with Time Windows (QPPTW).

Quickest Path Problem with Time Windows (QPPTW)	
Instance:	Graph G, source node s, destination node t, starting time θ, transit times $\tau(e)$, set of time-windows $\mathcal{F}(e)$ on each edge e.
Task:	Compute a dynamic path from s to t with minimum completion time that respects the given time-windows (w.r.t. $\tau(e)$).

Remark 3.1 (Existence)**.** *A dynamic s-t path that respects the given time-windows exists if and only if there is a static s-t path. This is due to the unbounded time horizon.*

Remark 3.2 (Other applications)**.** *The QPPTW is not only of interest in connection with the dynamic routing approach since one can also assume that the time-windows (or reservations) are given in advance. Consider for example the route assignment for private or public cargo railroads. Here, the task might be to route a single request/train through the network such that a given time table is respected.*

The QPPTW is related to the well-known Shortest Path Problem with

Time Windows (SPPTW) [11, 12, 13, 32] but differs in a subtle point: in the SPPTW there are additional edge costs that can (but do not necessarily) depend on the transit times while in the QPPTW the duration (traveling time including waiting times) is minimized. The duration can be viewed as costs that do not depend on the edge itself but on the routing history.

SHORTEST PATH PROBLEM WITH TIME WINDOWS (SPPTW)	
Instance:	Graph G, source node s, destination node t, starting time θ, transit times $\tau(e)$, costs $c(e)$, set of time-windows $\mathcal{F}(e)$ on each edge e.
Task:	Compute a shortest dynamic path (w.r.t. edge costs $c(e)$) that respects the given time-windows (w.r.t. τ_e).

The SPPTW is $\mathcal{NP}$-hard and therefore there is no polynomial time algorithm that solves the problem optimal, unless $\mathcal{P} = \mathcal{NP}$. The hardness can be shown by reduction from the Constrained Shortest Path Problem (CSPP [7]). The instance of the SPPTW is constructed by placing time-windows $[0, R]$ at each edge while R denotes the resource constraint in the CSPP instance. Then, obviously, a path respects the given time-windows if and only if it is feasible with respect to the resource constraint. Applying the same cost function on edges as in the CSPP instance completes the reduction.

In contrast, the QPPTW can be solved in polynomial time. Our algorithm for this problem is a generalized edge-based label-setting algorithm resembling Dijkstra's algorithm (Algorithm 2).

A *label* $L = (e_L, I_L, pred_L)$ represents a path from the source node s to the tail of e_L, where $pred_L$ is the predecessor of e_L on that path and the label interval $I_L = (a_L, b_L)$ represents an interval of possible arrival times at edge e_L (at the tail of e_L). Therefore, the earliest possible arrival time a_L (the elapsed time at the tail of e_L) is also the *cost value* of label L. We define an ordering for these labels. We say that a label L *dominates* a label L' if and only if

$$I_{L'} \subseteq I_L,$$

which implies $a_{L'} < a_L$.

The labels are stored in a priority queue H (a binary heap for example) that is sorted with respect to the earliest possible arrival time a_L (key). Each label that is extracted from H (line 8 of Algorithm 2) is propagated through the time-windows of the corresponding edge e_L (line 11 to 18). Figure 3.2 illustrates this propagation step. Afterwards the new labels are compared with the already existing labels with respect to the dominance rule (line 19

Algorithm 2: DYN-ROUTE-COMP

Data: Directed graph $G = (V, E)$ with transit times $\tau(e)$ for all $e \in E$, source node s, target node t, edge set $\mathrm{OUT}(e) \neq \emptyset$ $\forall e \in E$, transit time function τ, sorted set of time-windows $\mathcal{F}(e)$ $\forall e \in E$, release time θ.

Result: Dynamic path P with minimum completion time (and starting time $\hat{\theta} \geq \theta$) that respects the given time windows or the message that no such path exists.

begin

1 $H = \emptyset$;
2 $\mathcal{L}(e) = \emptyset$ $\forall e \in E$;
3 **foreach** $e \in \delta^+(s)$ **do**
4 $L = (e, (\theta, \infty), nil)$;
5 H.insert(L, θ);
6 $\mathcal{L}(e)$.insert(L);
7 **while** $H \neq \emptyset$ **do**
8 $L = H$.getMin();
9 **if** *t is tail of e_L* **then**
10 Construct path P based on the found labels and return it afterwards;
11 **foreach** $F^i_{e_L} = [a^i_{e_L}, b^i_{e_L}] \in \mathcal{F}(e_L)$ **do**
/* label expansion */
12 **if** $b_L < a^i_{e_L}$ **then**
13 **goto** 7 /* next label from heap */ ;
14 **if** $a_L > b^i_{e_L}$ **then**
15 **goto** 11 /* next time-window */ ;
16 $\theta_{in} = \max\{a_L, a^i_{e_L}\}$ /* $a^i_{e_L} > a_L \Rightarrow$ waiting */;
17 $\theta_{out} = \theta_{in} + \tau(e_L)$;
18 **if** $\theta_{out} \leq b^i_{e_L}$ **then**
19 **foreach** $f \in \mathrm{OUT}(e_L)$ **do**
/* dominance check */
20 $L' = (f, (\theta_{out}, b^i_{e_L}), L)$;
21 **foreach** $\hat{L} \in \mathcal{L}(f)$ **do**
22 **if** L' dominates $\hat{L}$ **then**
23 H.erase$(\hat{L})$;
24 $\mathcal{L}(f)$.erase$(\hat{L})$;
25 **else if** $\hat{L}$ dominates L' **then**
26 **goto** 19 /* next out-going edge */;
27
28 H.insert$(L', a_{L'})$;
29 $\mathcal{L}(f)$.insert(L') ;
30
31
32
33 **notification:** there is no s-t path ;

end

to 26). Note that we take turning rules into account by introducing an edge set OUT for each edge e. To this end, we need a directed graph representation and use the transformation given in Section 2.2.2 if the given graph is undirected.

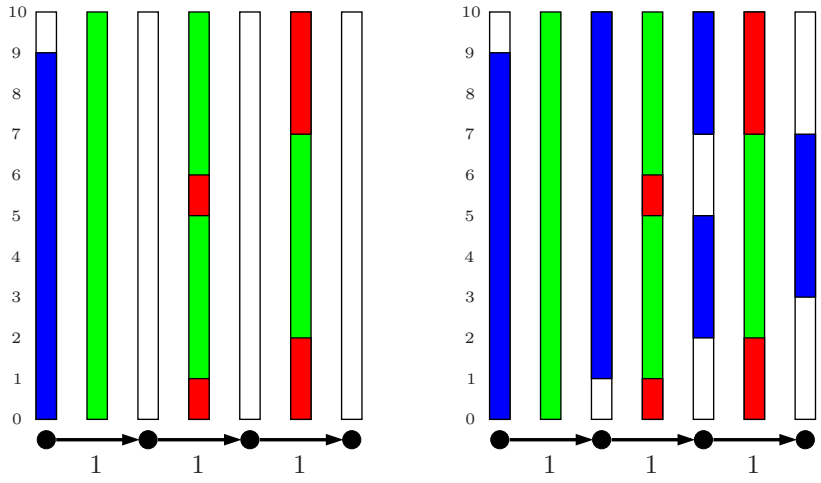


Figure 3.2: Label Expansion on three consecutive edges. The label intervals are represented by blue bars and are placed above the nodes. The blockings are colored red (edges). The green intervals between these blockings are the time-windows. Figure (b) shows the successive expansion of the label interval illustrated in (a).

Theorem 3.3 shows that Algorithm 2 solves the QPPTW in polynomial time (in the number of time-windows). Note that the number of time-windows is obviously linear in the number of reservations.

Theorem 3.3 ([19, 20])**.** *Algorithm 2 solves the QPPTW in polynomial time.*

Proof. The algorithm computes all required paths since the expansion of the label intervals is maximal and no optimal path (label) is dominated. Therefore, on termination the algorithm has computed an optimal path respecting the time-windows. The termination follows from the complexity analysis given below.

Consider the dominance rule: a label is dominated if the corresponding label interval is a subset of an existing label interval. Thus, for any two labels that are expanded w.r.t. the same time-window it holds that one of these labels dominates the other since the upper boundary of the corresponding label intervals is equal in this case.

Therefore, the number of possible labels on an edge e is bounded by the number of time-windows on all in-going edges. We call these time-windows

the in-going time windows $\mathcal{F}^-(e)$ of edge e. As a consequence, the number of iterations (the number of labels taken from the priority queue) is bounded from above by the sum of the number of in-going time-windows over all edges ($\sum_{e\in E}|\mathcal{F}^-(e)|$). In each iteration a label at an edge e is expanded along at most $|\mathcal{F}(e)|$ time-windows and each of the resulting labels is compared with at most $\sum_{f\in OUT(e)}|\mathcal{F}^-(f)|$ existing labels. If the priority queue is implemented as a heap, updating can be done in $O(\log(\sum_{e\in E}|\mathcal{F}^-(e)|))$. This leads to a run time of

$$O\left(\left(\sum_{e\in E}|\mathcal{F}^-(e)|\right)\cdot\left(\max_{e\in E}|\mathcal{F}(e)|\right)\cdot\left(\max_{e\in E}\{\sum_{f\in OUT(e)}|\mathcal{F}^-(f)|\}\right)\cdot\log\left(\sum_{e\in E}|\mathcal{F}^-(e)|\right)\right),$$

which is in $O(|\mathcal{F}|^3\log(|\mathcal{F}|)$, where $\mathcal{F}$ denotes the set of given time-windows.

Hence, the algorithm terminates in polynomial time with an optimal path or the notification that there is no feasible path at all. □

Although the algorithm has a polynomial run time, additional acceleration can be achieved by goal-oriented search [22, 34]. The idea is to add an estimation of the distance to the target to the current cost value of a label in order to direct the search towards the target. Since we use a lower bound for the estimation on the shortest distance to the target that satisfies the consistency assumption [22], the computed path is still an optimal path.

Algorithm 2 can also be used to assign idle vehicles to new requests/items, i.e., to determine the most suitable vehicle for this task.

Remark 3.4 (Assignment of idle vehicles to requests)**.** *Algorithm 2 can also be executed with multiple start labels. The analysis of the run time does not change in this case. Therefore, given the source node of a new request, the algorithm can be used to determine the idle vehicle that would reach that node at first.*

We conclude the description of the route computation with the modifications that have to be made if one considers the idealized disjointness model and an observation concerning the structure of the paths computed by the presented algorithm.

Idealized disjointness model. In the idealized model it is necessary to consider the reservations on nodes during the route computation. More precisely, the computation of θ_{in} in line 16 of Algorithm 2 must include a recognition of these reservations. This can be done by introducing additional time stamps for each time-window that implies the latest possible entry time for this time-window. In fact, the time-windows have to be entered before

that time. In Section 3.1.3 we present the corresponding transformation from node reservations to those time-windows.

The analysis of the algorithm still holds since the consideration of the additional time stamp is just a simple look up in line 16 of the algorithm. Additionally, we will see in Section 3.1.3 that the number of these time-windows is still polynomial in the number of reservations.

Observation 3.5 (Structure of the computed paths)**.** *In a given graph with time-windows on edges there may be several dynamic paths that use the same time-windows and lead to the same duration. These paths just differ in the distribution of the waiting intervals. Algorithm 2 computes a dynamic path such that waiting occurs at the latest possible edge before the respected reservation. All other possible distributions are either not generated or dominated during the algorithm.*

3.1.3 Readjustment of the Time-windows

After routing a request one has to take the new reservations into account, i.e., time-windows have to be readjusted according to the edge usage of the newly found route and their conflicting edges. Note that, as pointed out in Section 3.1.1, taking the conflicting edges into account implies that one does not have to take care of the vehicle dimensions during route computation, since it is already fully represented by readjusting the time-windows accordingly on all affected edges.

Our algorithm for the readjustment of the time-windows (Algorithm 3) works as follows. For each edge e of a computed path we consider the corresponding reservation and verify for all conflicting edges in *confl(e)* whether it intersects with a time-window. If this is the case, the time-window is readjusted (shortened, erased, or split) accordingly. This, obviously, can be done in time proportional to the number of time-windows on conflicting edges of the found path.

Idealized disjointness model. As pointed out in Section 3.1.2 we add a time stamp $\theta(F)$ to each time-window F if we consider the idealized disjointness model. This time stamp indicates that the corresponding time-window has to be entered before that time. The initial value is infinity.

After each route computation the time-windows are at first readjusted according to the edge reservations. This is done in the same way as described for the polygon model in Algorithm 3. The node reservations are considered by transforming the time-windows on all out-going edges of the reserved node v in the following way.

Algorithm 3: DYN-ROUTE-READJUST

Data: Directed graph $G = (V, E)$ with transit times $\tau(e)$ for all $e \in E$, dynamic path P with reservations $(\theta_e^{in}, \theta_e^{out})$ for all $e \in P$, sorted time-windows $\mathcal{F}(e)$ on the edges $e \in E$, set of conflicting edges confl(e) for all $e \in E$.

Result: Sorted set of time-windows $\mathcal{F}(e)$ including the reservations of P.

begin

```
1  | foreach f ∈ P do
2  |   foreach e ∈ confl(f) do
3  |     foreach F_e^i = [a_e^i, b_e^i] ∈ F(e) do
4  |       if θ_f^out ≤ a_e^i then
5  |         goto 3;
6  |       if θ_f^in ≤ a_e^i + τ(e) then
7  |         if θ_f^out ≥ b_e^i − τ(e) then
           |           /* erase time-window */
8  |           F(e).erase(F_e^i);
9  |         else
           |           /* shorten time-window */
10 |           F_e^i = [θ_f^out, b_e^i];
11 |       if θ_f^in < b_e^i then
12 |         if θ_f^out ≥ b_e^i − τ(e) then
           |           /* shorten time-window */
13 |           F_e^i = [a_e^i, θ_f^in];
14 |         else
           |           /* split time-window */
15 |           F_e^i = [a_e^i, θ_f^in] ;
16 |           F(e).insert([θ_f^out, b_e^i]);
17 |           goto 2 /* next conflicting edge */;
18 |       else
19 |         goto 2 /* next conflicting edge */;
20 |
21 |
22 |
```

end

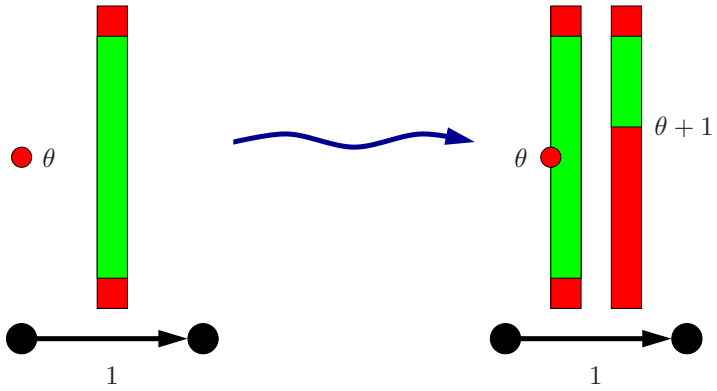


Figure 3.3: Transformation of a node reservation: On the left hand side a node reservation at time θ (red dot) and a time-window (green bar) on the out-going edge of that node is illustrated. We assume that this time-window has no time stamp (infinity). On the right side the resulting time-windows are shown. Here the red dot denotes the time stamp.

Let $a_v \in R_v$ be a reservation on node v and let $F_e = [a_e, b_e]$ be a time-window with time stamp $\theta(F_e)$ on an out-going edge e of node v. Then we distinguish between two cases:

1. $a_e \geq a_v + 1$ or $b_e \leq a_v$: F_e remains unchanged.

2. otherwise ($a_e \leq a_v$ and $b_e \geq a_v + 1$): F_e is transformed into an interval $F_e^1 = [a_e, b_e]$ with time stamp $\theta(F_e^1) = \min\{a_v, \theta(F_e)\}$ and an interval $F_e^2 = [a_v + 1, b_e]$ with time stamp $\theta(F_e)$ (see Figure 3.3).

   Note that empty time-windows (F_e^2 if $b_e = a_v + 1$) as well as time-windows with $\theta(F = [a, b]) \leq a$ are not generated. The latter is the case if $a_v = a_e$ (F_e^1) or if $\theta(F_e) \leq a_v + 1$ (F_e^2).

The described transformation is done for nodes on the determined path and all corresponding out-going edges. It leads to a formulation where only time-windows (with their time stamps) on edges have to be considered. The readjustment can obviously be done in polynomial time (in the total number of new reservations). Moreover, the total number of time-windows is bounded by the number of node reservations times the number of time-windows that result from edge reservations, which is linear in the reservations on edges. Thus, the total number of time-windows is still polynomial in the total number of reservations.

3.1.4 Practical Requirements

To make the algorithm practical, additional ingredients have to be taken into account. This demands for a variety of special features of the model.

Turning behavior

Although each route of a vehicle can be represented in the given graph, not every route in this graph might in fact be executed by a vehicle. The reason for this difficulty can be a complicated turning behavior, which makes it necessary to start turning the wheel already long before the particular intersection is reached. As a consequence, a vehicle needs a sufficiently long straight route segment between two consecutive curves. To cope with such a rather complicated turning behavior we introduce in a preprocessing step a set of *artifical edges* to the network, each representing a possible turn (see Figure 3.4). In addition, at each node of the graph we introduce turning rules by modifying the set of out-going edges $\mathrm{OUT}(e)$ that can be used from a particular in-going edge e. As a result we get a much larger network that captures all possible movements of a vehicle, i.e., each feasible route in this network can be executed by a vehicle. Note that the number of edges in the underlying grid-like graph at HHLA Container Terminal Altenwerder, cf. Section 2.3, increases from about 10,000 to about 45,000.

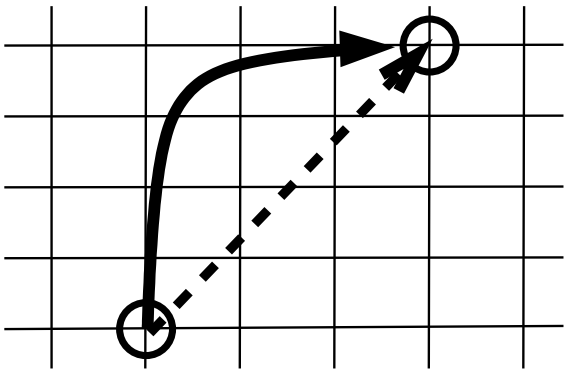
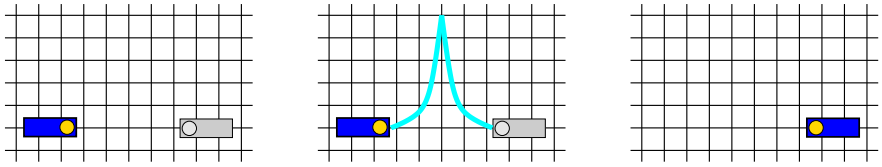


Figure 3.4: The figure illustrates an artificial edge (dotted arrow) that models a curve. This is done for all permitted curves.

Vehicle orientation

Sometimes it might be necessary to give a vehicle an explicit target orientation (see Figure 3.5) since the load of the vehicle has to be delivered in an explicit orientation. Maybe unloading is not possible otherwise.

One way to model this orientation constraint is to add a flag, indicating



(a) (b) (c)

Figure 3.5: Figure (a) shows a vehicle (on the left) with a given target orientation (on the right). To achieve that requirement the vehicle changes its driving direction (b) and reaches the target position in the correct target orientation (c).

whether the vehicle is in the right driving direction to reach the target with the correct orientation. It remains to show that the routing algorithm is capable of keeping track of this direction information during its search. This can be shown by observing that in the proof of Theorem 3.3 one needs only to maintain labels at an edge for each of the two possible directions and define the domination rule accordingly, i.e., such that only labels with the same information about the direction can dominate each other. Using this observation we still obtain a polynomial time algorithm since the maximum number of labels taken from the heap increases only by a factor of 2.

Corollary 3.6. *Algorithm 2 solves the QPPTW in polynomial time (in the number of time-windows) even if the orientation of vehicles is taken into consideration.*

Safety tubes

In spite of the fact that the routes computed by Algorithm 1 (DYN-ROUTE) are conflict-free additional safety is required in practice because the vehicles possibly deviate from the computed routes in time. Additionally, technical problems can occur while traveling through the network. We have implemented two different safety tubes, a distance-dependent and a time-dependent one, to cope with this difficulty.

The *distance-dependent safety tube* blocks an area in front of the vehicle. The length depends on the speed of the vehicle and is at least the distance needed to come to a complete stop (braking distance). This allows the vehicle to stop if something unexpected happens, for example an unscheduled stop of another vehicle, without causing a collision. In Section 3.2 we will discuss rerouting approaches in this context.

To maintain a distance-dependent safety tube the label expansion in Algorithm 2 is modified in the following way. The beginning of the label interval is no longer the time when an edge is entered, but the time when

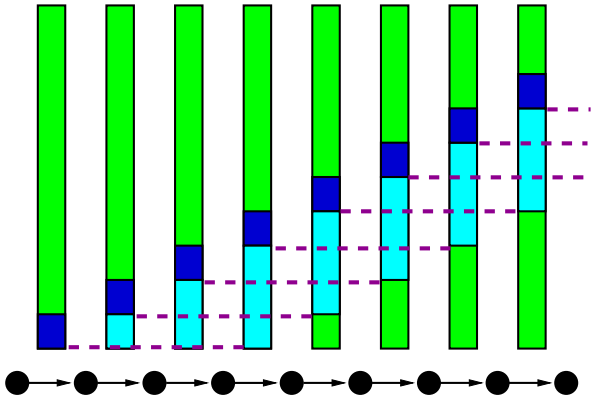


Figure 3.6: Distance-dependent safety tube. The bars in dark blue illustrate the times when the vehicle is on the corresponding edge while the light-blue bars show the intervals resulting from the distance-dependent safety tube which is represented by the dotted magenta lines.

the distance-dependent safety tube blocks that edge for the first time (see Figure 3.6). In the algorithm this time can be determined by regarding the history of the considered label. Since the number of already passed edges is linear in the number of time-windows this can be done efficiently.

The *time-dependent safety tube* allows a little deviation from the computed time, i.e., the expected arrival time at a specific point. This is necessary because there will always be small deviations in time in practice. Note that we will introduce more sophisticated approaches for dealing with large deviations in Section 3.2.

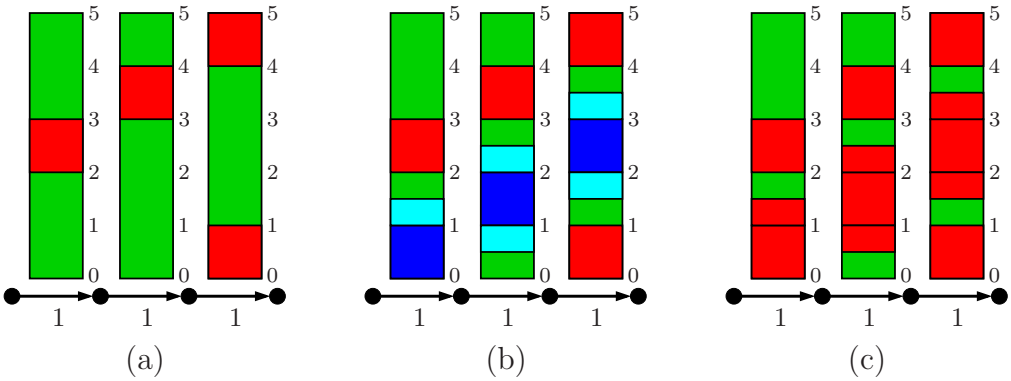


Figure 3.7: Illustration of the iterative routing scheme with time-dependent safety tube on three consecutive edges with transit time 1. In contrast to Fig. 3.1 the computed path in (b) and the resulting reservations in (c) are lengthened by a certain amount.

In the algorithm this safety tube is considered by lengthening the label intervals in the wanted direction by a certain amount. This amount can be made distance-dependent to reflect growing uncertainties along the route. Figure 3.7 illustrates the structure of the dynamic routing algorithm (Algorithm 1) with time-dependent safety tubes. We will evaluate the influence of such a tube on the performance of the dynamic routing algorithm in Section 3.3.2.

Non constant transit times

Instead of constant transit times as described in Section 3.1.2, we take the variable speed of the vehicles into account, i.e., we model the acceleration and deceleration behavior as stepwise linear functions and allow different maximum speeds on each edge of the graph. The maximum speed can depend on the kind of movement (curve or straight section), the weather conditions, and the status of the vehicle.

While the computation of the transit times in the acceleration process is easy to integrate in the presented approach by determining the actual transit time in the propagation step, the deceleration process needs a special handling. The necessity of deceleration cannot be recognized until the algorithm is forced to produce a waiting interval or to reduce speed (lower maximum speed) before entering an edge. Therefore, the transit times on the edges before this edge have to be adapted at this time according to the deceleration function. In case it turns out that deceleration was not possible with respect to the given time-windows on the preceding edges, the considered label is deleted.

Thus, we can conclude that non-constant transit time can be applied to our algorithm, but, obviously, we are not able to guarantee shortest paths anymore. This is basically due to the observation we made concerning the distribution of waiting in our algorithm (waiting is scheduled as late as possible, cf. Observation 3.5). If we consider non-constant transit times it becomes important where we plan waiting, cf. Figure 3.8. But since we are not able to provide waiting at all possible positions for any time segment in polynomial time, we give an heuristic approach to cope with that problem in Section 3.1.5.

3.1.5 Waiting Heuristic

In order to deal with waiting in a more flexible way we developed a heuristic approach that allows to reschedule waiting. The idea is to give the algorithm the possibility to plan waiting earlier than necessary.

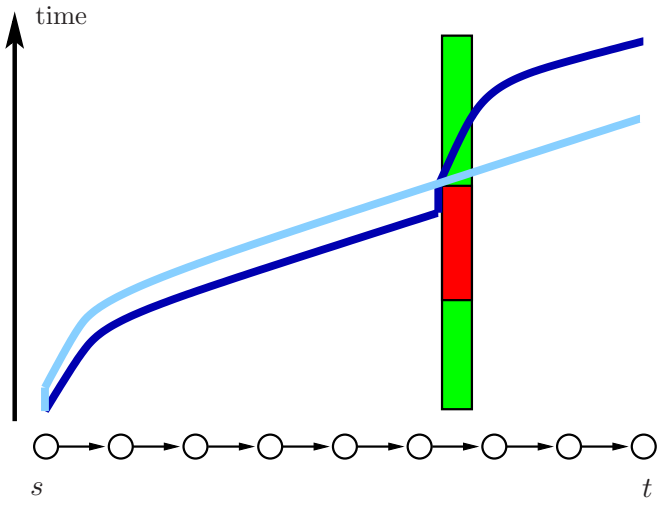


Figure 3.8: Illustration of two dynamic paths from node s to node t. While the dark blue path schedules waiting directly before the shown reservation (red), the light-blue path waits earlier and passes the corresponding edge without waiting. As a consequence, it reaches the target node t at first.

To this end, we introduce a new label value w_L. This value represents the latest possible entry time (on the corresponding edge) that can be achieved by just elongating the waiting time at the last waiting position p_L.

The domination rule is changed accordingly. In contrast to the domination rule formulated in Section 3.1.2, we say that a label L *dominates* a label L' if and only if

$$I_{L'} \subseteq I_L \text{ and } w_{L'} \leq w_L.$$

Algorithm 2 is adapted in the following way. In the propagation step of the algorithm we distinguish between two cases. On the one hand, the case when no waiting on the previous edge is necessary, i.e., if $a_L \geq a_i^{e_L}$ (line 16 of the algorithm), and the waiting case ($a_L < a_i^{e_L}$), on the other hand.

In the non-waiting case we only have to adapt the new label value (which has been initialized with ∞ at the beginning) according to the current time-window, i.e., we set

$$w_{L'} = \min\{w_L + \tau(e_L), b_i^{e_L}\}.$$

In the waiting case we check whether it is possible to avoid waiting on that edge by pushing the waiting interval to the last waiting position (rescheduling of waiting). If this is possible ($w_L \geq a_i^{e_L}$) we set

$$\theta_{in} = a_i^{e_L},$$

and readjust w'_L in the same way as in the non-waiting case. Moreover, we store the length of the rescheduled waiting interval in the label. Otherwise ($w_L < a_i^{e_L}$), the propagation step remains unchanged and $w_{L'}$ is set to $b_i^{e_L}$.

In addition to that new label value, we introduce a fixed parameter C, *the reschedule waiting parameter*, which bounds the number of edges we store the information about the possibility to reschedule an arising waiting interval. This means that we do not move a waiting interval more than C edges along the determined path. In the algorithm this is guaranteed by counting the number of edges since we waited the last time.

Now we show that Algorithm 2 with the described changes and the new domination rule still has a polynomial run time.

Theorem 3.7. *Algorithm 2 with the described waiting heuristic runs in polynomial time (in the number of time-windows).*

Proof. Consider two labels L and L' that produce a waiting interval on the same edge and are afterwards expanded (without waiting) through the same (at most C) time-windows. For these labels it obviously holds that

$$w_L = w_{L'}.$$

With the argumentation of Theorem 3.3 it follows that one of these two labels will dominate the other. Thus, from all labels that wait on the same edge and are expanded through the same time-windows afterwards without waiting, only one label will be extracted from the heap. Since this holds for each combination of at most C time-windows, the number of labels at a each edge is bounded from above by

$$\sum_{i=1}^{C} (\max_{e \in E} |\mathcal{F}^-(e)|)^i < (\max_{e \in E} |\mathcal{F}^-(e)|)^{C+1}, \tag{3.1}$$

where $\mathcal{F}^-(e)$, again, denotes the set of time-windows on all in-going edges of edge e. Note that we assume $\max_{e \in E} |\mathcal{F}^-(e)| \geq 2$ in Equation 3.1. If this is not the case (all edges have only one in-going edge and at most one time-window) the left hand side is bounded by a constant (C^2) anyway.

Thus, the the total number of labels is bounded by

$$|E| \cdot (\max_{e \in E} |\mathcal{F}^-(e)|)^{C+1}$$

and the same arguments as in Theorem 3.3 (label expansion along at most $\max_{e \in E} |\mathcal{F}_e|$ time-windows, dominance check with respect to at most the number of labels stored at the considered edge, size of the heap is bounded

by the total number of labels) lead to a run time of

$$
\begin{aligned}
& O\left(\left(|E| \cdot (\max_{e \in E} |\mathcal{F}^-(e)|)^{C+1}\right) \cdot \left(\max_{e \in E} |\mathcal{F}_e|\right) \cdot \right. \\
& \qquad \left. \left(|E| \cdot (\max_{e \in E} |\mathcal{F}^-(e)|)^{C+1}\right) \cdot \log\left((\max_{e \in E} |\mathcal{F}^-(e)|)^{C+1}\right)\right) \\
= \; & O\left(\left(|E| \cdot (\max_{e \in E} |\mathcal{F}^-(e)|)^{2C+2}\right) \cdot \left(\max_{e \in E} |\mathcal{F}_e|\right) \cdot \log\left(|E| \cdot (\max_{e \in E} |\mathcal{F}^-(e)|)^{C+1}\right)\right)
\end{aligned}
$$

if the priority queue is implemented as a heap. □

Note that the rescheduling of waiting does not change the completion time of a computed path if we consider constant transit times, see also Observation 3.5. But in the case of non-constant transit times it might avoid some deceleration and acceleration processes. We are going to evaluate this heuristic approach in Section 3.3.2.

3.2 Rerouting Strategies

Until now we assumed that all computed routes are executed almost exact, i.e., only small deviations are considered in the model by introducing time-dependent safety tubes in Section 3.1.4. In this section we present a rerouting approach (Section 3.2.3) that is able to deal with various, much more challenging, perturbations, which are introduced in Section 3.2.1.

As a welcome side effect this rerouting approach can be used in connection with priorities (Section 3.2.2) that might be associated with the vehicles or certain requests, respectively.

3.2.1 Perturbations

In this section we describe the six kinds of perturbations we take into consideration: namely blocked areas, broken-down vehicles, route cancellation, slow driving vehicles, late starting vehicles, and reservation conflicts.

Blocked areas. We call a connected zone of the given street network that cannot be used for a certain time period a blocked area. In our application, the HHLA Container Terminal Altenwerder, the declaration of such areas might be necessary for maintenance work at container cranes or at the sensors in the ground since in general the area where the AGVs operate is not allowed to be entered for safety reasons. Another reason might be a defective vehicle that has to be recovered.

We investigate two kinds of blocked areas: so-called hard blocked areas may not be entered from the time they are known on while soft blocked areas are only not usable for requests that appear after the area was set.

Broken-down vehicles. If a vehicle brakes down it probably makes some other, already computed, routes impossible. We assume that a broken-down vehicle informs the router, or at least the higher-level management system, about its status. Otherwise a reservation conflict (see below) would be the consequence and we have to react accordingly. Note that a broken-down vehicle is a special kind of a hard blocked area.

Route cancellation. Route cancellations occur if the higher-level management system decides to change the target of a certain request. If the corresponding path is already computed it has to be canceled and a new dynamic path has to be determined. Note that the vehicle is possibly already executing the original route when such a cancellation appears.

Slow driving vehicles. We call a vehicle that does not travel with the expected maximum speed a slow driving vehicle. This might be caused by problems with the engine, with the tires, or with the load.

Late starting vehicles. A vehicle that is late because it has not started traveling at the scheduled starting time is called a late starting vehicle. A reason for such a perturbation might be that the engine of the vehicle has not start immediately.

Remark 3.8 (Early vehicles)**.** *Note that in contrast to late vehicles (slow driving or late starting vehicles), dealing with vehicles that reach an edge of the corresponding dynamic path too early is easy. One only has to take care that the vehicle slows down in this case. A possible approach for achieving this is to assign earliest possible entry times to each edge of the path. There might be, however, small deviations in time, but this is unproblematic if one uses a time-dependent safety tube, cf. Section 3.1.4.*

Reservation conflicts. A reservation conflict occurs when a vehicle wants to pass an edge that is already occupied by another vehicle. The only reason for such a conflict is a perturbation that was not recognized in time since the dynamic paths computed by Algorithm 1 (DYN-ROUTE) are conflict-free.

3.2.2 Priorities

We discuss the prioritization of requests in order to take into account that requests might be of different importance/urgency which is possibly rooted in a given order of the transported objects at the delivery point. Another reason might be time limits for certain requests/items.

In our dynamic routing algorithm DYN-ROUTE (Algorithm 1) it is possible to take priorities of requests into consideration if we permit the rerouting of vehicles with lower priority. Then, these vehicles can be ignored during the route computation whenever they would force a prioritized vehicle to wait.

To this end, we introduce a procedure that checks—by iterating over the reservations on a certain edge e—whether the reservations in a given time interval I on edge e can be ignored. If this is the case, i.e., if the reservations that intersect with the interval I correspond to vehicles/requests of lower priority that can be stopped in time (before the conflict point is reached), a set of ignored requests is returned. Otherwise, a negative notification is provided.

We apply this procedure to Algorithm 2 in three phases during the propagation step. First of all, without considering any time-window, we try to provide a straight movement, i.e., we apply the procedure to the interval $[a_L, a_L + \tau(e_L)]$. If the answer is positive we create an new label L' with label interval $[a_L + \tau(e_L), a_L + \tau(e_L)]$.

Secondly, we use the introduced procedure if in line 16 of Algorithm 2 it holds that $a_L < a^i_{e_L}$. In this case we check interval $[a_L, a^i_{e_L}]$, i.e., we try to avoid waiting. If the affected requests/vehicles can be ignored we set $\theta_{in} = a_L$ and propagate without waiting. Otherwise, the propagation step remains unchanged.

The last adaption of the standard algorithm originates in line 18 of the algorithm, where we evaluate whether the chosen time-window can be left in time ($\theta_{out} \leq b^i_{e_L}$). If this is not the case we check the interval $[b^i_{e_L}, \theta_{out}]$ using the introduced check. If the answer is positive we proceed in the same way as if the if-statement had returned true.

In each of these cases the ignored vehicles/requests are stored in the label and after the route computation these vehicles are rerouted immediately (see Section 3.2.3).

3.2.3 Rerouting Approach

The first step in any rerouting approach is the detection of perturbations and the determination of the vehicles that have to be rerouted. Therefore, we describe how we achieve this before we introduce the rerouting strategies.

Blocked areas, route cancellations and reservation conflicts are easy to detect since they are reported by the higher-level management system. For the other incidents we use so-called position reports of the vehicles.

Such a report consists of the position of the vehicle, the time that corresponds to the position, and the status of the vehicle. Since each position corresponds to a certain edge we can compare the edge-time pair of the position report with the information of the computed path to check whether the vehicle is in time or too late. Hence, late starting and slow driving vehicles can be detected this way. Slow driving vehicles additionally report their current maximum speed (status). Moreover, broken-down vehicles can be identified by their status.

Based on the kind of perturbation we determine the vehicles that have to be rerouted. If a reservation conflict or a route cancellation occurs we only have to reroute the perturbed vehicle. The same is done in the case of late vehicles (slow driving vehicles and late starting vehicles). If other vehicles are already affected by such a perturbation a reservation conflict would have occurred and a rerouting is triggered that way.

In contrast, if permanent blockings, namely broken-down vehicles or hard blocked areas, appear we have to determine those vehicles that are not able to fulfill its request on the scheduled path. This is the case, if their path contains edges that are blocked by the new permanent blocking. We determine these vehicles by checking all reservations on the edges that are affected by the permanent blocking. Then, any vehicle that has a reservation with a larger entry time than the time of the appearance of the perturbation has to be rerouted. Additionally, for all new requests the permanent blocked edges are removed from the graph. This is also done in the case of soft blocked areas where no rerouting is required since the already routed requests can pass the area.

Now we describe how we proceed with the vehicles that have to be rerouted. In fact, we provide three different strategies.

Determining New Routes

Request that are affected by perturbations or are interrupted due to a prioritization of another request have to be rerouted. To this end, we developed the following strategies:

- the ‘Adapting Schedule’ strategy,
- the ‘Adapting Route’ strategy, and
- the ‘New Route after Stop’ strategy.

All these strategies are based on Algorithm 2. We adapt the algorithm according to the requirements of the particular strategy. Before we introduce the rerouting strategies in detail we provide some general remarks. Firstly, note that all strategies have in common that the original path is ignored during the computation of the new path. Moreover, the original path is removed after a successful determination of a new one. For the readjustment of time-windows we use Algorithm 3. Furthermore, since we want to permit reroutings without an intermediate stop before the new route is entered, we apply an additional feature to Algorithm 2. We add the starting speed to the initial label (in line 4 of the algorithm). This feature is used in the 'Adapting Schedule' strategy as well as in the 'Adapting Route' strategy.

The *'Adapting Schedule'* strategy is used for delayed vehicles (slow driving and late starting vehicles). It is based on the idea that the geographical part of the computed route is kept unchanged and only the schedule is adapted according to the new circumstances. Therefore, we restrict the search domain to the edges of the already computed path, i.e., in line 19 of Algorithm 2 we do not investigate all out-going edges of edge e_L but the edge that succeeds e_L on the original path. To ensure that the execution of the new path is possible we use a node that will not be reached before the new path is computed as source node. The speed at this node is approximated. By Theorem 3.3 it holds that this routing strategy is polynomial in the number of time-windows on the original path.

The *'Adapting Route'* strategy is applied to vehicles that are affected by blocked areas, broken-down vehicles or route cancellations. Here, we first determine all labels of the original path between the current position and the position of the conflict or the end of the path in the case of a route cancellation, respectively. Then, all these labels are added to the set of start labels. The outcome of the algorithm is a new path, on the one hand, and the point where the old path is left, on the other. The latter depends on the start label that has produced the optimal path.

Since both strategies can fail we need the third strategy which first stops the vehicle and then determines a new path by the standard version of Algorithm 2. Note that we also use this strategy after a reservation conflict. This *'New Route after Stop'* strategy may cause new affected vehicles due to the unexpected and unscheduled stop. These vehicles also have to be rerouted.

Moreover, we have to keep in mind that the stopped vehicles may block other vehicles such that they are no longer able to reach their target. In the next part we describe how we deal with such situations and with unrealizable requests in general.

Unrealizable Requests and Parking Zones

So-called *permanent reservations*, namely blocked areas, broken-down vehicles, or stopped vehicles can block other vehicles. In contrast to temporary reservations generated by a dynamic path, such reservations possibly lead to a failure of the route computation, i.e., Algorithm 2 is not able to provide a dynamic path for a certain request. Hence, this request cannot be fulfilled.

For each such *unrealizable request* there is at least one *responsible* set of permanent reservations. Here, 'responsible' means that there will be a dynamic path that fulfills the corresponding request if all permanent blockings in that set are removed.

Since the knowledge of these responsible sets is useful for anticipating whether it makes sense to retry the route computation we introduce Algorithm 4 which computes all inclusion-wise minimal responsible sets of permanent reservations between the desired source and target node. Note that these permanent reservations are represented by their conflicting edges, which are determined according to the procedure used for reservations of a dynamic path, i.e., via polygon intersections with the polygons that corresponds to the edges of the graph..

In that labeling algorithm (Algorithm 4), each label L consists of the corresponding edge e_L and a set B_L of permanent reservations. The dominance rule is defined as follows. A label L *dominates* a label L' if and only if

$$B_L \subseteq B_{L'}.$$

Note that Algorithm 4 does not terminate when one path is found but proceeds until all paths (all sets) are found. Obviously, this cannot be done efficiently since the output of the algorithm is exponential in the input size.

Theorem 3.9. *Algorithm 4 computes all inclusion-wise minimal sets of permanent reservations between a source node s and a target node t in a graph $G = (V, E)$ in*

$$O(4^p \cdot |E|),$$

where p denotes the number of permanent reservations in the graph.

Proof. Since we take those labels from the heap that have the least number of permanent reservations stored and additionally dominate all labels that are not inclusion minimal, the number of labels that are expanded along a certain edge is bounded by the largest number of inclusion minimal sets with at most p elements. Sperner showed that this number is $\binom{p}{\lfloor p/2 \rfloor}$ [31, 38].

From Stirling's formula we get that $p!$ is in $\Theta(\sqrt{p} \cdot (\frac{p}{e})^p)$. Thus, for even p

Algorithm 4: FIND-PERM-RESERVATIONS

Data: Directed graph $G = (V, E)$ with permanent reservations on edges, source node s, target node t, edge set $\mathrm{OUT}(e) \neq \emptyset$ $\forall e \in E$, empty heap H

Result: Set $\mathcal{B}$ of inclusion-wise minimal responsible sets of permanent reservations between s and t.

```
begin
1    H.insert(e_L, ∅);
2    while H ≠ ∅ do
3        extract a label L = (e_L, B_L) with |B_L| minimal from the
         heap H;
4        if t is tail of e_L then
5            add B_L to B;
6        else
7            if there is a permanent reservation b on e_L and b ∉ B_L
             then
8                B_L' = B_L ∪ {b};
9            else
10               B_L' = B_L ;
11           foreach f ∈ OUT(e_L) do
12               L' = (f, B_L');
13               foreach label L̂ at f do
14                   if L' dominates L̂ then
15                       remove L̂ from the heap H;
16                       remove L̂ from edge f ;
17                   if L̂ dominates L' then
18                       goto 11;
19
20               add L' to the heap H;
21               store L' add edge f;
22   return B;
end
```

it holds that $\binom{p}{\lfloor p/2 \rfloor}$ is in

$$\Theta\left(\frac{p!}{((p/2)!)^2}\right) = \Theta\left(\frac{\sqrt{p} \cdot (\frac{p}{e})^p}{(\sqrt{p/2} \cdot (\frac{p}{2e})^{p/2})^2}\right) = \Theta\left(\frac{1}{\sqrt{p}} \cdot 2^p\right). \tag{3.2}$$

Moreover, for odd p the binomial coefficient can be written as

$$\binom{p}{(p-1)/2} = \frac{p!}{((p+1)/2)! \cdot ((p-1)/2)!} = \frac{2p}{p+1} \frac{(p-1)!}{(((p-1)/2)!)^2}$$

which is, due to (3.2), also in $\Theta(\frac{1}{\sqrt{p}} \cdot 2^p)$. Therefore, $\binom{p}{\lfloor p/2 \rfloor}$ is in $O\left(\frac{2^p}{\sqrt{p}}\right)$ in general.

Since this bounds the number of label expansions at a single edge, the total number of label expansions is in

$$O\left(\frac{2^p}{\sqrt{p}} \cdot |E|\right).$$

A label is expanded by adding the permanent reservation on the considered edge if there is a reservation that is not already contained in the label. This can be done in linear time. After each label expansion we have to check for other labels that may dominate the new one or vice versa. Since each check requires at most linear time ($2p$) and updating the heap takes $O(\log \frac{2^p}{\sqrt{p}})$, each label expansion can be done in

$$O\left(p + \frac{2^p}{\sqrt{p}} \left(p + \log \frac{2^p}{\sqrt{p}}\right)\right) = O\left(\frac{2^p}{\sqrt{p}} \cdot p\right),$$

which leads to the claimed overall running time of

$$O\left(\frac{2^p}{\sqrt{p}} \cdot |E| \cdot \frac{2^p}{\sqrt{p}} \cdot p\right) = O(4^p \cdot |E|).$$

□

Remark 3.10. *It is also possible to start the computation with the labels from the original route computation that have been deleted because of a permanent reservation.*

Using the presented algorithm, permanent reservations are taken into account in the following way. Firstly, if a route computation fails the (inclusion-wise minimal) responsible sets are computed. Then, whenever a permanent reservation is removed the corresponding sets are adjusted. If a set becomes empty the unrealizable requests that correspond to this set are routed again.

However, there are situations in which additional actions are necessary. In fact, this is the case if several unrealizable requests block each other, i.e., if a group of vehicles is not able to reach its target due to a permanent reservation caused by another vehicle of the group. In such a kind of deadlock situation we make use of so-called parking positions. Note that these areas are also useful to keep the waiting vehicles away from areas with high traffic density.

Parking zones. We try to route the vehicles that cannot be routed to their target positions to so-called *parking positions* that are located such that the parked vehicles do not interfere with other (traveling) vehicles. This is done to avoid perturbations caused by vehicles that wait for a new route, on the one hand, and to dissolve the described cyclic dependencies of stopped vehicles, on the other hand. If the routing to parking positions also fails we determine the inclusion-wise minimal sets (Algorithm 4) also for these requests.

Note that the parking zones are located such that always at least one vehicle is able to reach such a position. More precisely, the parking positions are placed on the borders of the underlying graph and, therefore, at least the vehicle that is next to the border is able to move to such a position.

3.3 Computational Results

In this section we are going to evaluate the presented dynamic routing approach in certain real-life scenarios using the simulation environment mentioned in Section 2.3 (simulation of HHLA Container Terminal Altenwerder). Firstly, after an introduction of the considered test instances and objectives, we investigate the dynamic routing algorithm DYN-ROUTE under different parameter settings. More precisely, we vary the rescheduling waiting parameter C introduced in Section 3.1.5 and the time-dependent safety tube described in Section 3.1.4.

Moreover, we aim at investigating the quality of the solutions provided by DYN-ROUTE. Unfortunately, the results of the currently used routing approach at CTA are confidential and optimal solutions (or good lower bounds) cannot be determined due to the problem size. Therefore, we refer to Section 4.2, where we provide optimal solutions in smaller test instances, and to Section 5.4, where we consider a so-called static routing approach in our simulation environment, for detailed evaluations of DYN-ROUTE with respect to optimal solutions and other routing approaches. In Section 3.3.3, however, we provide simple lower bounds to get a first impression of the performance rating.

In Section 3.3.4 we focus on the introduced rerouting strategies. To this end, we consider test instances that require rerouting, namely scenarios with perturbations or priorities, respectively.

3.3.1 Test Instances and Objective

We consider two scenarios of practical relevance provided by HHLA. For the evaluation of our routing approach we use the simulation environment

introduced in Section 2.3. Recall that it simulates the AGVs at HHLA Container Terminal Altenwerder (CTA) and provides the underlying grid-like graph.

The scenarios differ in the number and position of the pick-up and delivery points. In the first scenario (SCEN-A), which is illustrated in Figure 3.9, there are 22 pick-up points on the bottom and 12 delivery points on the top of the grid-like graph. In contrast, the second scenario (SCEN-B, Fig. 3.10) has 14 delivery points while pick-up points are arranged in the same way as in SCEN-A.

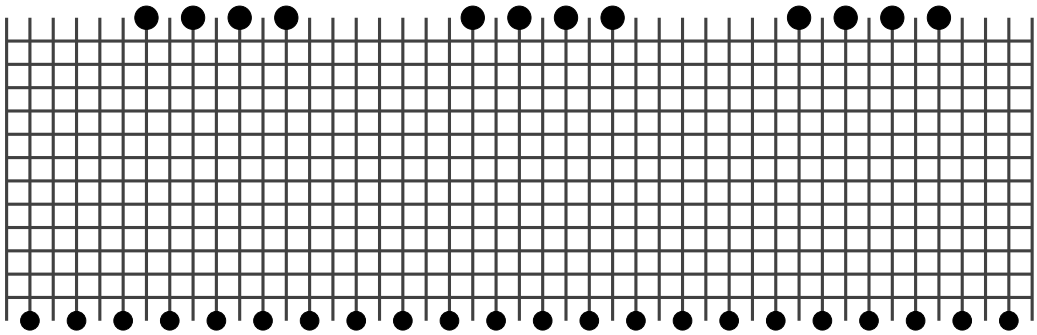


Figure 3.9: Scenario SCEN-A

In each six-hour simulation run we evaluate about 6000 requests. Request are assigned to vehicles by the integrated management system immediately whenever a vehicle becomes idle. SCEN-A uses 72 vehicles and in SCEN-B there are 79 vehicles that fulfill the arising requests. The results are determined by averaging over three test runs for each test case.

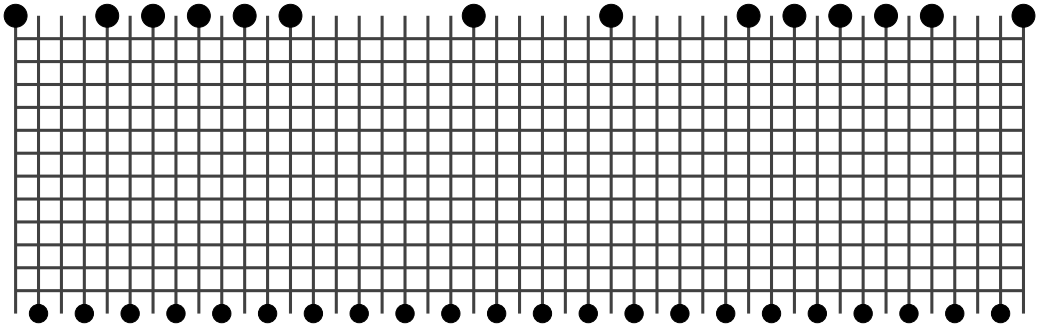


Figure 3.10: Scenario SCEN-B

In order to avoid redundancy we only consider one objective as a measure of the performance, the *average duration* over all requests. Beyond the equivalence to the objective of the OSDDPP, the overall duration, the makespan

(objective of the OQDPP) is approximated very well in our setting. Due to the nonexistence of idle times and the large number of requests the makespan is almost the average number of requests per vehicle (# requests/ # vehicles) times the average duration. Hence, we are able to measure the performance of our algorithm concerning both problems with the considered objective avoiding to provide (almost similar) results for both problems.

In addition to the performance evaluation with respect to the considered online disjoint vehicle routing problems we also focus on the real-time suitability of our approach. To this end, we determine the average as well as the maximum *computation times*. Note that we ran the simulation on an AMD Athlon 64 Dual Core 2.2 GHz with 4 GB RAM.

Moreover, for the evaluation of the rerouting strategies, we will introduce additional indicators in Section 3.3.4.

3.3.2 Variation of the Introduced Parameters

We introduced two parameters in connection with the dynamic routing algorithm DYN-ROUTE (Algorithm 1), the rescheduling waiting parameter used in the waiting heuristic (Section 3.1.5) and the length of the time-dependent safety tube (see Section 3.1.4). We first evaluate the waiting heuristic and consider different safety tubes afterwards.

Waiting Heuristic

The waiting heuristic introduced in Section 3.1.5 permits the rescheduling of waiting, i.e., the movement of waiting intervals to positions that are already passed in the route computation. Using this heuristic, the so-called rescheduling waiting parameter C bounds the number of edges the waiting intervals are moved. In order to determine the best choice for that parameter we evaluate several settings. Note that the second parameter, the length of the safety tube, is set to 1.0 seconds for these experiments.

Table 3.1 illustrate the effect of the considered parameter C on the performance and the computation times in scenario SCEN-A and SCEN-B, respectively. The results are almost the same in both scenarios.

The evaluation shows that the worst average duration is generated when the heuristic is not used ($C = 0$). The best results are achieved if parameter C is set to 20. An interesting observation is that increasing the rescheduling waiting parameter does not lead to a better performance in general. On the contrary, the average duration increases for $C = 40$ in SCEN-A and $C = 30$ in SCEN-B, respectively, and does not change significantly for larger values of C. The stagnation is due to the fact that rescheduling waiting

C	average duration (in sec.)		average comp. time (in sec.)		maximum comp. time (in sec.)	
	SCEN-A	SCEN-B	SCEN-A	SCEN-B	SCEN-A	SCEN-B
0	185.23	188.01	0.08	0.10	0.97	1.08
10	184.56	186.87	0.08	0.10	0.81	0.98
20	182.26	183.90	0.08	0.10	0.83	1.02
30	182.59	185.66	0.08	0.10	0.88	1.09
40	184.79	186.02	0.09	0.10	1.07	1.01
50	184.27	186.17	0.08	0.10	0.90	0.99
100	184.89	186.33	0.08	0.10	0.98	1.19

Table 3.1: Performance of DYN-ROUTE with different settings of the reschedule waiting parameter C in SCEN-A and SCEN-B.

over large distances is often not possible, i.e., larger bounds do not lead to different paths. The reason for the increase of the duration from $C = 20$ to $C = 40$ or $C = 30$, respectively, is not clear.

The computation times are almost similar for all parameter settings. They only increase a bit in SCEN-A for $C = 40$.

We conclude that $C = 20$ is the best choice for the rescheduling parameter in the considered scenarios. To this end, we will use this setting from now on.

Time-dependent Safety Tube

In Section 3.1.4 we introduced a time-dependent safety tube. Such a safety tube, of course, has an influence on the performance of the dynamic routing algorithm DYN-ROUTE. Thus, we investigate this effect in both scenarios using constant (independent of the covered distance) safety tubes of 1.0, 1.5, 2.0, and 2.5 seconds. In fact, we lengthen the reservations equally in both directions by half the amount of the tube. Note that smaller values than 1.0 seconds are not suitable for coping with deviations in practice (details are confidential).

Table 3.2 and Table 3.3 show the evaluation with respect to the average duration (in sec.). Absolute values as well as the differences between the certain time steps are illustrated.

The main observation is that the average duration decreases in value when smaller safety tubes are chosen. This is not surprising. Moreover, the

safety tube	2.5 s		2.0 s		1.5 s		1.0 s
average duration	198.62		192.81		187.25		182.26
difference		5.81		5.56		4.99	

Table 3.2: Variation of the time-dependent safety tube in SCEN-A.

safety tube	2.5 s		2.0 s		1.5 s		1.0 s
average duration	199.49		194.37		188.82		183.90
difference		5.12		5.55		4.92	

Table 3.3: Variation of the time-dependent safety tube in SCEN-B.

dependence between the length of the safety tube and the benefit of performance is approximately linear. The loss of performance is 5 to 6 seconds (about 3 percent) per time step of 0.5 seconds. Regarding the computation times we observe that the variation of the time-dependent safety tube does not lead to significant deviations of the computation times. Therefore, we omit the corresponding figures.

Standard parameter setting. For all further evaluations we set the reschedule waiting parameter to 20 and use a time-dependent safety tube of 1.0 seconds. In this standard setting we obtain an average duration of 182.26 seconds in SCEN-A and 183.90 seconds in SCEN-B, respectively. The average computation time is 0.08 seconds in SCEN-A and 0.10 seconds in SCEN-B while the maximum computation times are 0.83 (SCEN-A) and 1.02 (SCEN-B), respectively.

3.3.3 Trivial Lower Bound

In order to obtain a first indication of the performance of the dynamic routing algorithm DYN-ROUTE we determine trivial lower bounds by computing an average static shortest path distance (w.r.t. the transit time τ) for each instance; i.e., we relax the collision constraint. The results of the comparison are illustrated in Table 3.4.

In both test instances, SCEN-A and SCEN-B, we get a gap of about 50 percent. If we take into account that the considered lower bound is pretty poor, this provides a strong informative basis for the assumption that the dynamic routing algorithm performs well. As mentioned before, we will

	lower bound	DYN-ROUTE	gap
SCEN-A	119.46	182.26	52.57 %
SCEN-B	119.69	183.90	53.65 %

Table 3.4: Comparison with a trivial lower bound. We compare the average duration achieved by DYN-ROUTE with the lower bound we get from the average static shortest path distances.

have a closer look at the performance of the dynamic routing algorithm DYN-ROUTE in Section 4.2 and Section 5.4.

3.3.4 Evaluation of the Rerouting Strategies

In order to evaluate the performance of our rerouting strategies introduced in Section 3.2.3 we investigate them under the influence of perturbations and in instances with priorities. Since the results of the previous sections showed that the algorithm behaves similarly in both considered scenarios we focus on scenario SCEN-A in this section.

Besides the average duration we focus on two additional performance measures. In fact, we evaluate the number of reroutings per request and the delay of the vehicles in comparison with the expected completion time, i.e., the determined completion time for the original request.

Moreover, instead of comparing the plain computation times, we consider response times, i.e., the time period between the release time and the termination of the route computation. This time only differs from the computation time if several requests arise almost at the same time, which only plays a role in scenarios with massive reroutings. The plain computation times are omitted since they do not differ from those in the scenario without perturbations.

Perturbations

We investigate seven scenarios, based on the standard scenario SCEN-A, with different kinds of perturbations.

BL-A: We consider two soft blocked areas that cover essential parts of the grid such that there are only one third of the horizontal lanes left in these parts (see Figure 3.11).

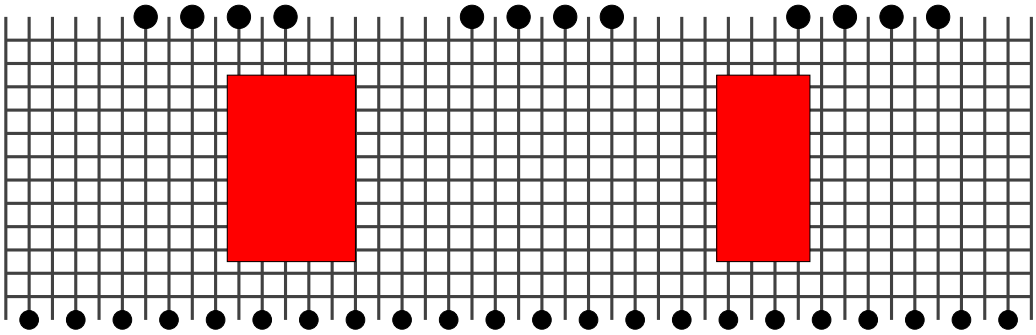


Figure 3.11: Scenario BL-A

BD-V-10: In every tenth request the corresponding vehicle brakes down. It recovers after a time-out of 20 seconds. Then, the request is served as required.

BD-V-50: In every fiftieth request the corresponding vehicle brakes down. It recovers after a time-out of 20 seconds. Then, the request is served as required.

CANCEL: Every tenth request is canceled by assigning a new target position while the request is served. The cancellation occurs 60 seconds after the release time of the original request.

LATE-10: In every tenth request the corresponding vehicle starts with a delay of 60 seconds.

LATE-50: In every fiftieth request the corresponding vehicle starts with a delay of 60 seconds.

SLOW-10: In every tenth request the corresponding vehicle moves at half speed.

SLOW-50: In every fiftieth request the corresponding vehicle moves at half speed.

The results of the experiments are illustrated in Table 3.5. The evaluation shows that the scenarios with late vehicles, in particular LATE-10, lead to the worst average duration. Here, the loss of performance, in terms of the average duration of the requests, is up to 23 percent. In contrast, in scenario BD-V-50 we observe just a little deviation from the setting without perturbations

(SCEN-A). Note that the durations of the perturbed requests are also taken into account.

The average response times do not change significantly. They just increase a bit in scenario LATE-10 and LATE-50, but if we compare the maximum values we observe large differences. While we get similar values as in the unperturbed setting for scenario CANCEL, the scenarios with broken-down vehicles show maximum response times of 10.80 and 10.70, respectively. This is a strong indicator for the conjecture that the hardness of the rerouting problem highly depends on the location where a vehicle brakes down. Since the average response time increases only by one hundredth of a second such worst cases seem to be very unusual. Moreover, these values are still suitable for a real-time computation since we consider response times and not pure computation times. As pointed out, the computation times remain unchanged. Therefore, we can conclude that large response times indicate that there are a lot of reroutings 'at the same time', possibly combined with above-average computation times.

	average duration (in sec.)	average resp. time (in sec.)	maximum resp. time (in sec.)	reroutings per request	average delay (in sec.)	maximum delay (in sec.)
SCEN-A	182.26	0.08	0.83	0.00	0.00	0.0
BL-A	212.31	0.08	1.14	0.00	0.00	0.0
BD-V-10	207.86	0.09	10.80	0.26	8.94	289.4
BD-V-50	186.58	0.09	10.70	0.06	1.30	225.2
CANCEL	194.60	0.08	0.83	0.22	0.12	41.0
LATE-10	223.91	0.11	4.50	0.26	14.51	290.3
LATE-50	206.67	0.10	2.40	0.07	4.23	293.6
SLOW-10	202.39	0.09	3.14	0.20	10.96	545.8
SLOW-50	188.96	0.08	2.20	0.06	2.91	428.4

Table 3.5: Evaluation of the scenarios with perturbations. Besides the average duration we consider the average and maximum response time, the number of reroutings per request and the resulting delay.

However, there is no direct correlation between the number of reroutings and the (maximum) response times. While the response times in scenario BD-V-10 and scenario BD-V-50 are almost similar, the number of reroutings is significantly different. Another argument for this observation is the maximum response time in scenario LATE-10, which is much smaller than in scenario BD-V-10, while both scenarios show a similar number of

reroutings per request. Note that there are no reroutings in scenario BL-A since we consider soft blocked areas.

Therefore, there are no delays in this scenario, too. For all other test cases we get average delays between 0.12 seconds (CANCEL) and 14.51 seconds (LATE-10). Note that in the CANCEL scenario we do not take delays into account that occur due to a changed target, i.e., the delays are measured with respect to the first route that was computed to the real target. Therefore, in this scenario delays only arise if a route cancellation produces reroutings of other requests. This is the reason why the delays are rather small in this case. In the other test cases, it turns out that the more challenging instances ('-10') produce much larger delays than the '-50'-variants which is not surprising. Besides this, there is no significant correlation between the delays and the average duration. Note that the average duration must not be the sum of the average duration in the non-perturbed case plus the average delay since the original routes might be already different in the perturbed case, usually they are longer. The comparison of the maximum delays is only of low significance if one aims at suggesting the hardness of an instance since these values are almost random. However, they give an idea what effects the considered perturbations might have.

Priorities

In order to evaluate the prioritization of requests we consider accordant scenarios based on scenario SCEN-A. In each scenario we choose a set of pick-up points and prioritize the requests that correspond to this pick-up point, i.e., the requests that have the pick-up point as source or target.

For the choice of these pick-up points we focus on two properties. The first is the average duration of the corresponding requests in SCEN-A (without priorities) and the second is the distance between the chosen pick-up points. For the latter we consider so-called *clusters*. We call a set of pick-up points a cluster if the pick-up points are located 'closely' to each other, i.e., such that the corresponding request interfere each other with high probability. SCEN-A provides three such clusters, each consisting of four pick-up points (see Figure 3.12). Note that the exact distances between the pick-up points in SCEN-A are confidential.

For our experiments we consider four scenarios that differ concerning these properties.

PRIO-1: Choose the three pick-up points from the same cluster that show the longest average duration of the corresponding requests in SCEN-A and prioritize those requests.

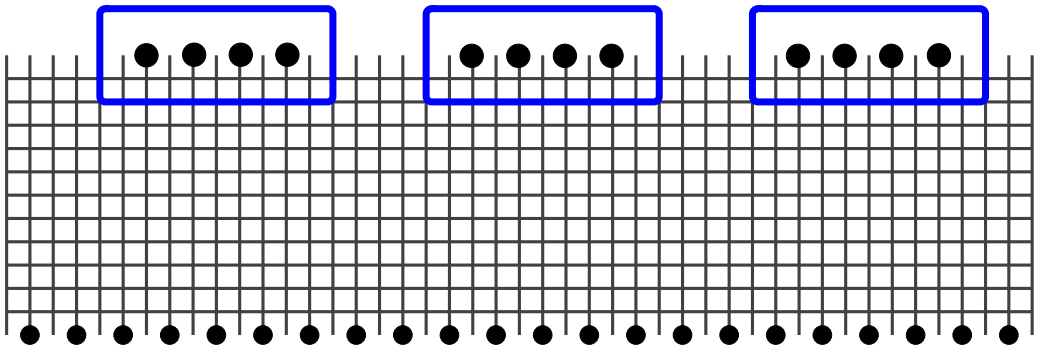


Figure 3.12: Clusters in SCEN-A

PRIO-2: Choose the three pick-up points from the same cluster that show the least average duration of the corresponding requests in SCEN-A and prioritize those requests.

PRIO-3: Choose from each cluster the pick-up point with the longest average duration of the corresponding requests in SCEN-A and prioritize those requests.

PRIO-4: Choose from each cluster the pick-up point with the least average duration of the corresponding requests in SCEN-A and prioritize those requests.

Due to the construction of the scenarios we prioritize about 1500 requests (one fourth of the overall requests). Besides the evaluation of the performance with and without prioritization we again analyze the response times, the number of reroutings and the delays at the target location.

In Table 3.6 and Table 3.7 we illustrate the comparison of the average duration over all requests and the average duration of the prioritized requests in the unprioritized setting with the prioritized setting. This is done for all considered scenarios. Here, unprioritized setting (no prio) means that the values are determined in the standard scenario SCEN-A.

The evaluation shows that the prioritization leads to the best results, in terms of the benefit for the prioritized requests, if these requests had a long average duration before and do not perturb each other too much (PRIO-3). Consequently, the prioritization of clustered requests that have already had a good performance, i.e., a low average duration, before only has small effects (PRIO-2). The results of the other instances are in between since there the requests either have a bad initial performance (PRIO-1) or do not influence

	PRIO-1		PRIO-2	
	no prio	prio	no prio	prio
average duration (all requests)	182.26	185.90	182.26	186.24
difference		**2.0** %		**2.2** %
average duration prioritized requests	201.25	184.23	162.50	158.98
difference		**-8.5** %		**-2.3** %

Table 3.6: Prioritization of requests that correspond to pick-up points in a cluster. PRIO-1 prioritizes requests with long average duration in the base scenario SCEN-A while PRIO-2 prioritizes those with short average duration. The table shows the influence of prioritization on the average duration of all requests and the prioritized requests, respectively. Both is measured in seconds.

	PRIO-3		PRIO-4	
	no prio	prio	no prio	prio
average duration (all requests)	182.26	189.65	182.26	189.30
difference		**4.1** %		**3.9** %
average duration prioritized requests	198.62	168.72	171.66	151.02
difference		**-15.1** %		**-12.0** %

Table 3.7: Prioritization of requests that correspond to pick-up points from different clusters. PRIO-3 prioritizes requests with long average duration in the base scenario SCEN-A while PRIO-4 prioritizes those with short average duration. The table shows the influence of prioritization on the average duration of all requests and the prioritized requests, respectively. Both is measured in seconds.

each other too much (PRIO-4). Here, the latter seems to be the stronger criterion. Figure 3.13 illustrates this observation.

As an additional performance measure we consider the loss of overall performance if we prioritize certain requests. Here, we observe that the overall average duration increases more in the test cases with non-clustered requests which is not surprising since these scenarios also lead to larger benefits of the prioritized requests. Note that the loss of overall performance is always smaller than the gain of performance that is achieved by the prioritized requests.

Table 3.8 shows that the number of reroutings correlate with the described results concerning the average duration, i.e., a large number of reroutings lead to a stronger prioritization. The number of reroutings, in turn,

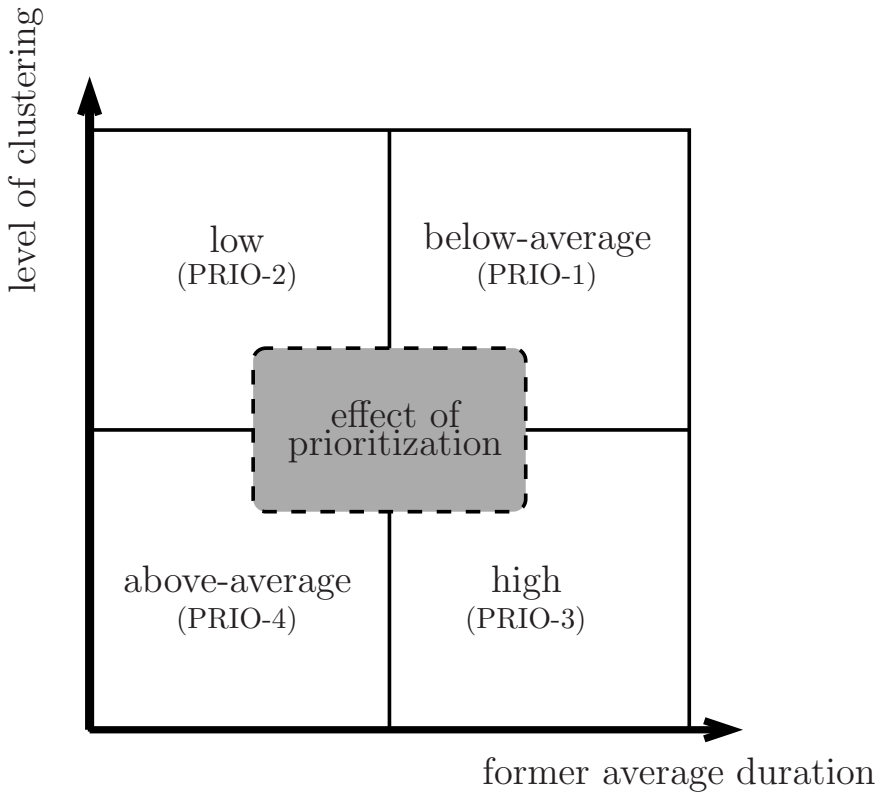


Figure 3.13: Illustration of the observed effect of prioritization with respect to the average duration without prioritization (former average duration) and the level of clustering of the considered requests.

highly depend on the number of vehicles that can be ignored by a prioritized vehicle. This number is larger if the prioritized requests are not too much clustered. This might be the reason for the greater effect of prioritization in the scenarios PRIO-3 and PRIO-4.

The delays and response times are in line with the determined number of reroutings. The largest delays and response time occur in scenario PRIO-3, where we observe 0.91 reroutings per request, while these values are small in scenario PRIO-2 with only 0.26 reroutings per request.

Note that, in contrast to the test cases with perturbations, the sum of average delay and average duration without priorities exceeds the overall average delay with priorities. This is due to the fact that prioritized requests benefit from the rerouting of the other requests. This reduces the average duration but not the average delay. In the perturbation case the vehicle that causes the reroutings does not benefit and is possibly delayed itself.

	re-routings per request	average delay (in sec.)	maximum delay (in sec.)	average resp. time (in sec.)	maximum resp. time (in sec.)
PRIO-1	0.65	12.3	146.3	0.13	3.54
PRIO-2	0.26	5.0	181.7	0.10	1.77
PRIO-3	0.91	17.6	345.9	0.20	5.11
PRIO-4	0.64	13.7	145.8	0.13	4.23

Table 3.8: Evaluation of the number of reroutings, the delays and the response times in the scenarios with priorities.

3.4 Conclusions

In this chapter we presented an online dynamic routing algorithm for the considered disjoint vehicle routing problems, namely for the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP) and the Online Quickest Disjoint Paths Problem (OQDPP). The algorithm iteratively assigns dynamic paths to the incoming (transportation) requests. For each request the computation consists of the determination of a dynamic path with minimal duration that respects the reservations of the already routed requests and a subsequent reservation of the found path. Both can be done in polynomial time.

Based on this basic approach we also investigated additional, practical requirements like the physical dimensions of the vehicles, the acceleration and deceleration behavior, and certain safety aspects.

Since we aim at providing an approach that is suitable for the use in practice three questions certainly arise:

- Is the algorithm able to ensure a fast real-time computation?
- Is the performance of the algorithm satisfactory?
- Is the algorithm able to deal with perturbations?

To answer these questions we consider the routing of Automated Guided Vehicles (AGVs) at HHLA Container Terminal Altenwerder (CTA). More precisely, we evaluate our approach in a simulation environment that models CTA.

The first question can be answered in the affirmative. The computation times presented in Section 3.3 showed that the algorithm is able to provide

fast answers. On the average the computation requires not more than some hundredths of a second.

For a concluding answer to the second question we refer to the next chapter where we will analyze the performance of the dynamic routing approach theoretically via competitive analysis and empirically in comparison to other routing approaches and optimal solutions. In this chapter we provided a comparison to a simple lower bound in Section 3.3.3. Although this evaluation is of limited significance, it is already a strong indicator of the good performance of the algorithm.

In order to discuss the last question we implemented a rerouting environment based on the dynamic routing approach. Using the strategies introduced in Section 3.2 we are able to show that our approach can cope with various kinds of perturbations without losing too much performance. Moreover, the computation times remain suitable for a real-time use. As a welcome side effect we are able to prioritize requests using this rerouting approach. The results are impressive. Prioritization of one fourth of the requests lead to a gain of performance of up to 15 percent on average for the prioritized requests. Additionally, the interference of the other requests is acceptable, i.e., the overall performance decreases only slightly.

Chapter 4

Performance Analysis of the Dynamic Routing Approach

In this chapter we analyze the performance of the dynamic routing algorithm DYN-ROUTE (Algorithm 1) introduced in Chapter 3.

In Section 4.1 we use competitive analysis to determine the theoretical worst case performance of $\Theta(k)$, where k denotes the number of requests, with respect to both considered problems: the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP) and the Online Quickest Disjoint Paths Problem (OQDPP). Note that these are the first results concerning the approximation behavior of these problems. Additionally, we introduce and analyze another dynamic routing algorithm in Section 4.1.2. Using this approach we are able to improve the approximation ratio given by Spenke [37] for the QDPP in grid graphs for instances with a specific property.

Moreover, the quality of both approaches is measured experimentally in Section 4.2. To this end, we consider optimal solutions as well as the approximation algorithm of Spenke for the purpose of comparison. Some of these results can be found in [29].

For both parts of the analysis we use the idealized disjointness model (see Section 2.2.2) and pay special attention to grid graphs and their narrowness.

4.1 Competitive Analysis

In Section 2.1.2 we introduced competitive analysis as a measure for the worst case performance of an online algorithm. We focus on an analysis such as this for the dynamic routing algorithm DYN-ROUTE (Algorithm 1) and another dynamic routing approach, which we present in Section 4.1.2. In order to simplify the argumentation we consider unit length reservations and waiting intervals (see Remark 2.9).

4.1.1 Dynamic Routing Algorithm

In order to analyze the performance of DYN-ROUTE we firstly bound the competitive ratio from above. In fact, we prove that our dynamic routing

algorithm is k-competitive with respect to the OQDPP and the OSDDPP, where k denotes the number of requests. Afterwards we show that this upper bound is asymptotically tight in undirected graphs as well as in directed graphs.

Performance Guarantee

To show a performance guarantee of k for both considered problems we only have to argue that the dynamic routing algorithm performs at least as well as an approach that routes the requests one after another on shortest static paths. Although this is easy to see, even for the precise model, we give a formal proof of this observation.

Theorem 4.1. *The dynamic routing algorithm* DYN-ROUTE *(Algorithm 1) is k-competitive with respect to the Online Quickest Disjoint Paths Problem (OQDPP) and the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP), where k denotes the number of requests.*

Proof. Consider a sequence of requests $\sigma = r_1, \ldots, r_k$ and let T_i denote the time when the first i requests are completed. Moreover, recall that we refer to the shortest static path distance between two nodes s_i and t_i as $\text{dist}(s_i, t_i)$.

Since routing each request over a shortest static path after all preceding requests are served is a feasible solution for both considered problems, it is easy to see that DYN-ROUTE, which routes the requests over shortest dynamic paths, determines a path P_i for the i-th request $r_i = (s_i, t_i, \theta_i)$ with completion time not greater than $\max\{\theta_i, T_{i-1}\} + \text{dist}(s_i, t_i)$. Thus, for the duration Δ_{P_i} of the i-th request it holds that

$$\Delta_{P_i} \leq \sum_{j=1}^{i} \text{dist}(s_j, t_j). \tag{4.1}$$

This leads to an upper bound of

$$\sum_{i=1}^{k} \sum_{j=1}^{i} \text{dist}(s_j, t_j) < k \cdot \sum_{j=1}^{k} \text{dist}(s_j, t_j)$$

to the overall duration, which shows the claimed ratio for the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP) since for any instance $\mathcal{I}$ of that problems it holds that

$$\text{OPT}(\mathcal{I}) \geq \sum_{j=1}^{k} \text{dist}(s_j, t_j).$$

To see that ratio also for the Quickest Disjoint Paths Problem (OQDPP) we consider the latest request $r_\ell = (s_\ell, t_\ell, \theta_\ell)$ with the property

$$\theta_\ell > \theta_1 + \sum_{j=1}^{\ell-1} \text{dist}(s_j, t_j)$$

(or r_1 ($\ell = 1$) if no such request exists). Then, for any instance $\mathcal{I}$ of the OQDPP it holds that

$$\text{OPT}(\mathcal{I}) \geq \max_{i \geq \ell} \{\theta_i + \text{dist}(s_i, t_i)\}.$$

Since, due to (4.1), the solution of our dynamic routing algorithm DYN-ROUTE is bounded from above by

$$\theta_\ell + \sum_{i=\ell}^{k} \text{dist}(s_i, t_i) \leq \theta_\ell + k \cdot \max_{i \geq \ell} \{\text{dist}(s_i, t_i)\},$$

we get an upper bound of k for the competitive ratio of DYN-ROUTE. □

Remark 4.2 (Polygon model)**.** *Theorem 4.1 also holds if one considers the polygon model introduced in Section 2.2.2 since the arguments used in the proof also hold in this case.*

In the following sections we will show, by introducing two instances on the line, that this bound is tight for arbitrary undirected and directed graphs. Moreover, we are going to provide lower bounds for grid graphs.

Example Undirected Graphs

Example 4.3 shows that the worst case performance of the dynamic routing approach DYN-ROUTE on the (undirected) line is in $\Omega(k)$.

Example 4.3. *We consider k request on the line with $2k$ edges, cf. Figure 4.1.1. The requests appear in alternating directions. More precisely, they are organized as follows:*

$$r_i = \begin{cases} (v_i, v_{k+i}, 0) & \text{for} \quad i = 1, 3, 5, \ldots, k-1 \\ (v_{k+i}, v_i, 0) & \text{for} \quad i = 2, 4, 6, \ldots, k \end{cases}.$$

DYN-ROUTE serves the requests in the order they appear. Therefore, each request has to wait until all preceding requests are finished, i.e., by

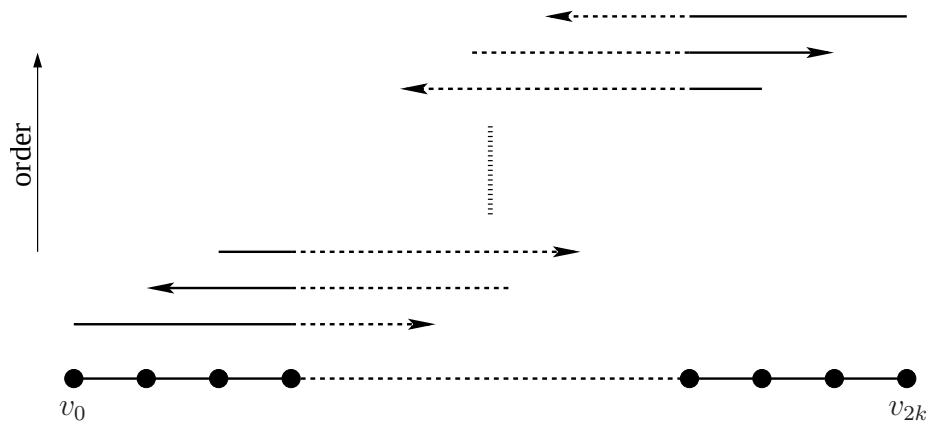


Figure 4.1: Illustration of the instance considered in Example 4.3.

construction of the requests, the starting time of a request corresponds to the completion time of the preceding request. Since each request has a travel time (without waiting) of k, the completion time as well as the duration of request r_i is $i \cdot k$. This leads to a makespan of $k \cdot k = k^2$ and an overall duration of

$$\sum_{i=1}^{k} ik = \frac{(k+1)k^2}{2}. \tag{4.2}$$

An optimal algorithm for the QDPP as well as an optimal algorithm for the SDDPP routes all requests in one of the two directions first and then it serves all requests in the opposite direction.

Therefore, all requests of the first group finish at time k. Since, again by construction of the requests, the second group starts at this time, we get a makespan of $2k$ and an overall duration of

$$\frac{k}{2} \cdot k + \frac{k}{2} \cdot 2k = \frac{3k^2}{2}.$$

This leads to a competitive ratio of $k/2$ with respect to the OQDPP (makespan) and of $(k+1)/3$ with respect to the OSDDPP (overall duration), respectively. Thus, we conclude that the upper bound shown in Theorem 4.1 for the dynamic routing approach DYN-ROUTE is (asymptotically) tight.

Theorem 4.4. DYN-ROUTE *(Algorithm 1) is* $\Theta(k)$*-competitive with respect to the OQDPP and the OSDDPP in undirected graphs.*

Example Directed Graph

In order to show that the claimed lower bound also holds for directed graphs we consider an example on the directed line.

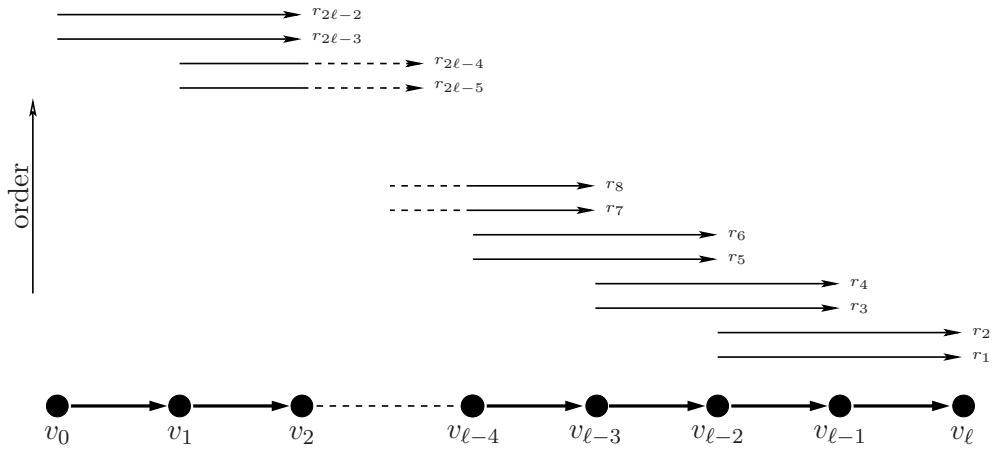


Figure 4.2: Illustration of the instance considered in Example 4.5.

Example 4.5. *Consider a directed line with $\ell > 2$ edges and a sequence $\sigma = r_1, \ldots, r_{2\cdot(\ell-1)}$ of $k = 2(\ell-1)$ requests that appear in the following order (cf. Figure 4.2):*

$$\begin{aligned} r_{2i+1} &= (v_{\ell-(i+2)}, v_{\ell-i}, 0) \text{ for } i = 0, \ldots, \ell-2, \\ r_{2i+2} &= (v_{\ell-(i+2)}, v_{\ell-i}, 0) \text{ for } i = 0, \ldots, \ell-2. \end{aligned}$$

The dynamic routing algorithm DYN-ROUTE serves the requests in the order they appear and computes the dynamic paths such that waiting occurs as late as possible (Observation 3.5). This leads to the paths illustrated in Figure 4.3, i.e., for $i = 0, \ldots, \ell-2$ we get:

$$\begin{aligned} P_{2i+1} &= (0, (v_{\ell-(i+2)}, 0), (v_{\ell-(i+1)}, i+1), (v_{\ell-i}, i+2)), \\ P_{2i+2} &= (0, (v_{\ell-(i+2)}, i+1), (v_{\ell-(i+1)}, i+2), (v_{\ell-i}, i+3)). \end{aligned}$$

This leads to a makespan of $\ell + 1$ and a overall duration of

$$\begin{aligned} \textstyle\sum_{i=0}^{\ell-2}(i+2) + \sum_{i=0}^{\ell-2}(i+3) &= 2 \cdot \textstyle\sum_{i=0}^{\ell-2} i + \sum_{i=0}^{\ell-2} 5 \\ &= 2(\tfrac{(\ell-2)(\ell-1)}{2}) + 5(\ell-1) \\ &= \ell^2 + 2\ell - 3. \end{aligned}$$

In contrast, an optimal algorithm for the SDDPP and the QDPP, respectively, routes the requests with odd index, i.e., the requests that appear first on a certain node, simultaneously and serves the second group of requests afterwards. In doing so one gets the following optimal solution for both

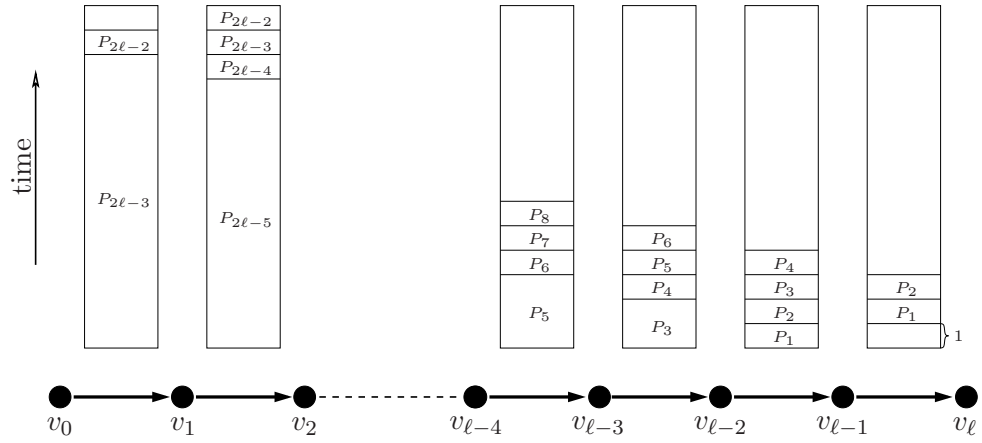


Figure 4.3: Solution of DYN-ROUTE for the instance described in Example 4.5.

problems ($i = 0, \ldots, \ell - 2$):

$$\begin{aligned} P_{2i+1} &= (0, (v_{\ell-(i+2)}, 0), (v_{\ell-(i+1)}, 1), (v_{\ell-i}, 2)), \\ P_{2i+2} &= (0, (v_{\ell-(i+2)}, 2), (v_{\ell-(i+1)}, 3), (v_{\ell-i}, 4)). \end{aligned}$$

Figure 4.4 illustrates the arrangement of the dynamic paths. It is easy to see that a makespan of 4 is optimal with respect to the QDPP, since this is exactly the number of paths that have to pass each edge (except the first and the last). Moreover, we get a overall duration of

$$(l-1) \cdot 2 + (l-1) \cdot 4 = 6(l-1),$$

which is optimal with respect to the SDDPP since whenever we start a request of the second group earlier, at least one request of the first group will have a longer duration.

Comparing both solutions, the one generated by DYN-ROUTE and the optimal solution, leads to the claimed competitive ratio of $\Omega(k)$ for directed graphs. More precisely, taking into account that $\ell = (k+2)/2$, we get a ratio of

$$\frac{\ell+1}{4} = \frac{k+4}{8} > \frac{k}{8}$$

for the OQDPP and of

$$\frac{\ell^2 + 2\ell - 3}{6(l-1)} > \frac{\ell^2 - \ell}{6(l-1)} = \frac{\ell}{6} > \frac{k}{12}.$$

for the OSDDPP.

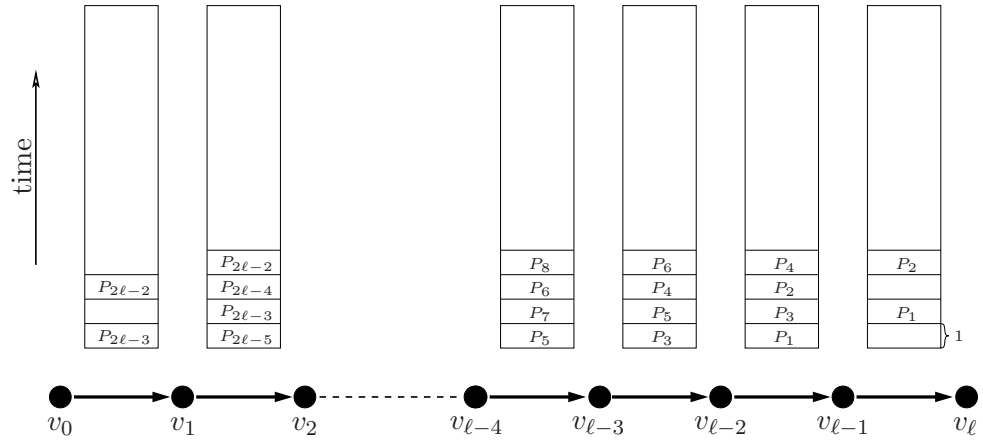


Figure 4.4: Optimal solution for the instance described in Example 4.5.

Theorem 4.6. DYN-ROUTE *(Algorithm 1) is* $\Theta(k)$*-competitive with respect to the OQDPP and the OSDDPP in directed graphs.*

Remark 4.7 (Grid graphs)**.** *The presented results also hold if we consider the class of grid graphs, because the line is a grid graph by definition. Moreover, the presented example for the directed line is also an example for the undirected case. Thus, we can adapt this example such that it holds for an arbitrary number of (undirected) horizontal lanes. This is done by duplicating the line example. If we then order the requests such that they appear in the different copies in a round robin fashion, i.e., we consider the sequence*

$$r_{1,1}, r_{1,2}, \ldots, r_{1,m}, \ldots, r_{i,1}, \ldots, r_{i,j}, \ldots, r_{i,m}, \ldots, r_{k,m},$$

where $r_{i,j}$ *denotes the* i*-th request in the* j*-th copy,* DYN-ROUTE *as well as an optimal algorithm would route the requests in the same way as illustrated in Example 4.5 in each copy. In addition, adding edges between these copies in order to obtain an undirected grid graph does neither change the optimal solution nor the solution of* DYN-ROUTE*. This leads to a lower bound of* $\Omega(k/m)$ *for grid graphs. Moreover, since we are able to increase the number of requests per horizontal lane by increasing the number of vertical lanes, we get a lower bound of* $\Omega(\mathcal{N})$*, where* $\mathcal{N}$ *denotes the narrowness of the grid.*

A Closer View on Directed Graphs

Although the given examples suggest that the performance of DYN-ROUTE is almost the same in directed and undirected graphs, there is a subtle dif-

ference: While the waiting times in Example 4.3 (undirected line) are independent of the number of requests already served, they are bounded from above by this number in Example 4.5 (directed line).

Lemma 4.8 shows that this holds on directed lines in general.

Lemma 4.8. *Consider a sequence of requests $\sigma = r_1, \ldots, r_k$ on the directed line served by the dynamic routing algorithm* DYN-ROUTE *(Algorithm 1). Then, for each determined dynamic path P_i that routes the i-th request $r_i = (s_i, t_i, \theta_i)$ it holds that it contains at most $i-1$ unit length waiting intervals. Hence, the duration Δ_{P_i} is bounded from above by*

$$\mathrm{dist}(s_i, t_i) + i - 1.$$

Proof. Since the static length of the computed path is fixed in this kind of graph we only have to concentrate on the generated waiting intervals. Firstly, note that each waiting interval of a dynamic path that is computed by DYN-ROUTE is caused by a reservation of another dynamic path. If this reservation is also a waiting interval, we can iterate that observation and end up with a reservation that belongs to a non-waiting interval. We call such a reservation the origin of the considered waiting and the corresponding path the causing path. Obviously, waiting originates in crossings of routes or in source nodes of requests (if a path enters the graph), which can also be interpreted as a kind of crossing.

On the directed line waiting always originates in source nodes of requests since there are no crossings. More precisely, waiting of a path P_i is either caused by a path P_ℓ that enters the line on a node v_ℓ that succeeds the source node v_i of P_i on the directed line or by a path P_j that crosses the source node of P_i (v_j does not succeed v_i on the directed line). Obviously, each path that is computed before P_i can either cause the first or the second kind of waiting since the corresponding source node either succeeds the source node of P_i or not. Therefore, the number of (unit length) waiting intervals on a dynamic path computed by DYN-ROUTE is bounded by the number of requests already served, which completes the proof. □

Lemma 4.9 describes the connection between such a bound to the duration and the competitive ratio of the corresponding algorithm. Applying this result to Lemma 4.8 directly leads to Theorem 4.10. Note that the showed competitive ratio is (asymptotically) tight (due to Example 4.5).

Lemma 4.9. *Consider a sequence $\sigma = r_1, \ldots, r_k$ of requests $r_i = (s_i, t_i, \theta_i)$. Then, an algorithm* ALG *that determines dynamic paths $P_1, \ldots, P_k$ such that*

for some $h \geq 1$ and for some $\ell \geq -1$ it holds that

$$\Delta_{P_i} \leq \operatorname{dist}(s_i, t_i) + hi + \ell \text{ for all } i \in \{1, \ldots, k\}$$

is $1+(hk+\ell)/L_{max}$-competitive with respect to the OQDPP and $1+(hk+2\ell+h)/2L_{avg}$-competitive with respect to the OSDDPP, where $L_{max} := \max_i\{\theta_i + \operatorname{dist}(s_i, t_i)\}$ and $L_{avg} := \sum_i \operatorname{dist}(s_i, t_i)/k$.

Proof. Firstly, we consider the OQDPP. Due to the given bound to the duration Δ_{P_i} of a path P_i the makespan is bounded by

$$\begin{aligned} \max_i\{\theta_i + \operatorname{dist}(s_i, t_i) + hi + \ell\} &= L_{\max} + hk + \ell \\ &= (1 + \tfrac{hk+\ell}{L_{\max}})L_{\max}. \end{aligned}$$

Since in any solution the makespan is at least $L_{\max}$ this leads to the claimed competitive ratio, i.e.,

$$\operatorname{ALG}(\mathcal{I}) \leq \left(1 + \frac{hk + \ell}{L_{\max}}\right) \cdot \operatorname{OPT}(\mathcal{I})$$

for any instance $\mathcal{I}$ of the OQDPP.

Regarding the OSDDPP we observe, again due to the given bound, that the overall duration is not greater than

$$\begin{aligned} \textstyle\sum_{i=1}^{k} \operatorname{dist}(s_i, t_i) + hi + \ell &= \textstyle\sum_{i=1}^{k} \operatorname{dist}(s_i, t_i) + \frac{h(k+1)k}{2} + k\ell \\ &= \textstyle\sum_{i=1}^{k} \operatorname{dist}(s_i, t_i) + \frac{(hk+2\ell+h)k}{2} \\ &= (1 + \tfrac{hk+2\ell+h}{2L_{\text{avg}}}) \textstyle\sum_{i=1}^{k} \operatorname{dist}(s_i, t_i). \end{aligned}$$

Obviously, the optimal value in any instance of the OSDDPP is bounded from below by $\sum_{i=1}^{k} \operatorname{dist}(s_i, t_i)$, i.e.,

$$\operatorname{ALG}(\mathcal{I}) \leq \left(1 + \frac{hk + 2\ell + h}{2L_{\text{avg}}}\right) \cdot \operatorname{OPT}(\mathcal{I})$$

for any instance $\mathcal{I}$, which completes the proof. □

Theorem 4.10 (Competitive ratio directed line)**.** *Consider a sequence of requests $\sigma = r_1, \ldots, r_k$ with $r_i = (s_i, t_i, \theta_i)$. Then,* DYN-ROUTE *(Algorithm 1) is $1 + (k-1)/L_{max}$-competitive with respect to the OQDPP and $1 + (k-1)/2L_{avg}$-competitive with respect to the OSDDPP on the directed line, where $L_{max} := \max_i\{\theta_i + \operatorname{dist}(s_i, t_i)\}$ and $L_{avg} := \sum_i \operatorname{dist}(s_i, t_i)/k$.*

4.1.2 Dynamic Routing Algorithm without Waiting

In this section we present a modified dynamic routing algorithm. As in the presented dynamic routing approach DYN-ROUTE we iteratively compute routes for each incoming request. But, instead of computing a shortest dynamic path, we determine a *shortest distance dynamic path*, i.e., a dynamic path that is based on a shortest static path. Moreover, we only permit waiting at the source node. We use the term *'without waiting'* to denote this property of the computed paths.

Before we describe the algorithm in detail we introduce conflicts of static paths (with a certain time stamp θ) as illustrated in Figure 4.5.

Definition 4.11 (Conflict of a static path)**.** *A static path $\bar{P} = (v_1, \ldots, v_n)$ has a* conflict with time stamp θ (or a θ-conflict) *w.r.t. a dynamic path P if there is, for some $\theta \in \mathbb{N}$, either a node $v_j \in \bar{P}$ with*

$$\theta + \text{dist}(v_1, v_j) \in \mathcal{R}_P(v_j) \qquad \text{(node conflict in } v_j\text{)}$$

or an edge $v_j v_{j+1}$ (edge conflict on $v_j v_{j+1}$) *with*

$$(\theta + \text{dist}(v_1, v_j), \theta + \text{dist}(v_1, v_j) + 1) \in \mathcal{R}_P(v_j v_{j+1}).$$

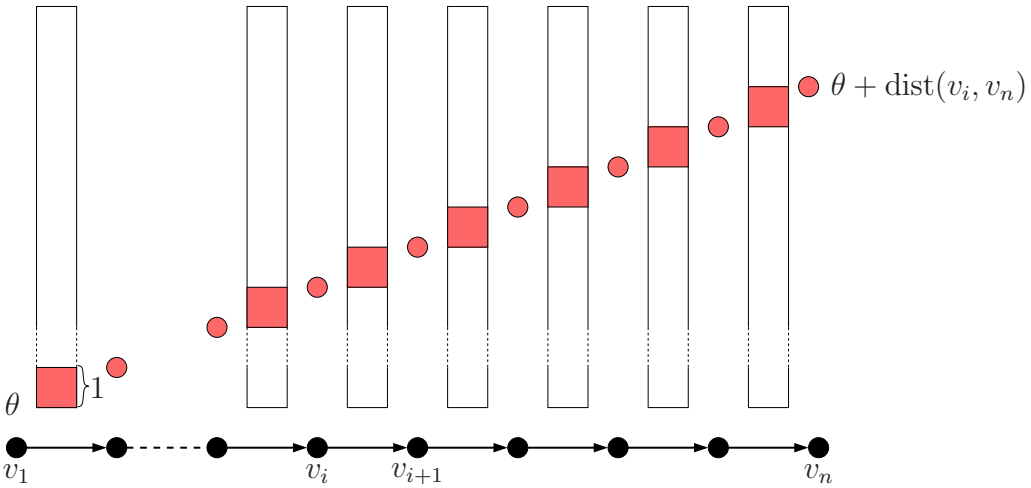


Figure 4.5: Illustration of a (shortest) static path with a set of potential reservations (light red) of other dynamic paths. Each of these reservations would lead to a θ-conflict with respect to a certain $\theta \in \mathbb{N}$.

Obviously, there is a shortest distance dynamic path without waiting with starting time θ if and only if the corresponding static shortest path has no θ-conflict. To this end, our dynamic routing algorithm without waiting

Algorithm 5: DYN-ROUTE-SP

```
Data: Graph G = (V, E), sequence of requests σ = r_1, ..., r_k.
Result: Sequence of dynamic paths P_1, ..., P_k.
begin
    foreach request r_i = (s_i, t_i, θ_i) do
        compute a shortest static s_i-t_i-path P̄_i = (v_1, ..., v_n) ;
        j = 0 ;
L       while there is a (θ_i + j)-conflict do j = j + 1 ;
        return the dynamic path
        P_i = (θ_i, (v_1, θ_i + j), ..., (v_m, θ_m + j + dist(v_1, v_m))) ;
        readjust the set of reservations accordingly ;
end
```

DYN-ROUTE-SP (Algorithm 5) determines, based on a shortest static path, the minimum starting time that does not imply such a conflict with respect to the given reservations. Recall that the resulting waiting times at the source node do not lead to any reservations since we, as explained in Assumption 2.6, assume that the source nodes of the requests (the pick-up and delivery points) can be used simultaneously.

The run time of the algorithm and the worst case performance depend on the maximum number of conflicts with distinct time stamps. Since the argumentation of Theorem 4.1 also holds for DYN-ROUTE-SP we observe that the algorithm is k-competitive with respect to the OQDPP and the OSDDPP. Moreover, due to the bound to the makespan, the number of loops in line L of the algorithm is bounded by $k \cdot \sum_i \text{dist}(s_i, t_i) \leq k \cdot |E|$. Since the shortest path computation as well as the check for conflicts can be done in polynomial time DYN-ROUTE-SP is efficient.

Corollary 4.12. *The dynamic routing algorithm* DYN-ROUTE-SP *(Algorithm 5) runs in polynomial time and is k-competitive with respect to the Online Quickest Disjoint Paths Problem (OQDPP) and the Online Shortest Dynamic Disjoint Paths Problem (OSDDPP), where k denotes the number of requests.*

Concerning the lower bounds for undirected graphs given in Section 4.1.1 for DYN-ROUTE we observe that DYN-ROUTE-SP routes the requests given in Example 4.3 in the same way. Thus, in undirected graphs DYN-ROUTE-SP is also $\Theta(k)$-competitive. In contrast, it is not clear if this also holds for directed graphs.

On the directed line each conflict of a shortest static path with a dynamic

path computed by DYN-ROUTE-SP has the same time stamp per definition, cf. Figure 4.5. Thus, for the duration of each path P_i determined by DYN-ROUTE-SP it holds that

$$\Delta_{P_i} \leq \text{dist}(s_i, t_i) + i - 1.$$

Applying Theorem 4.9 this leads to the same competitive ratio as shown for DYN-ROUTE.

Corollary 4.13. *Consider a sequence $r_1, \ldots, r_k$ of requests $r_i = (s_i, t_i, \theta_i)$. Then,* DYN-ROUTE-SP *(Algorithm 5) is $1 + (k-1)/L_{max}$-competitive with respect to the OQDPP and $1 + (k-1)/2L_{avg}$-competitive with respect to the OSDDPP on the directed line, where $L_{max} := \max_i\{\theta_i + \text{dist}(s_i, t_i)\}$ and $L_{avg} := \sum_i \text{dist}(s_i, t_i)/k$.*

So far we obtained similar results for DYN-ROUTE-SP as for DYN-ROUTE. Now we are going to investigate grid graphs. For this graph class we were not able to give a better competitive ratio than for arbitrary (undirected) graphs in Section 4.1.1.

In order to show a better worst case performance for DYN-ROUTE-SP we use a simple transformation: We construct a directed grid graph by assigning directions to the horizontal and the vertical edges alternatingly (see Figure 4.6). Note that two edges can be used in both directions (anti-parallel edges that can not be used at the same time) to provide a strongly connected graph.

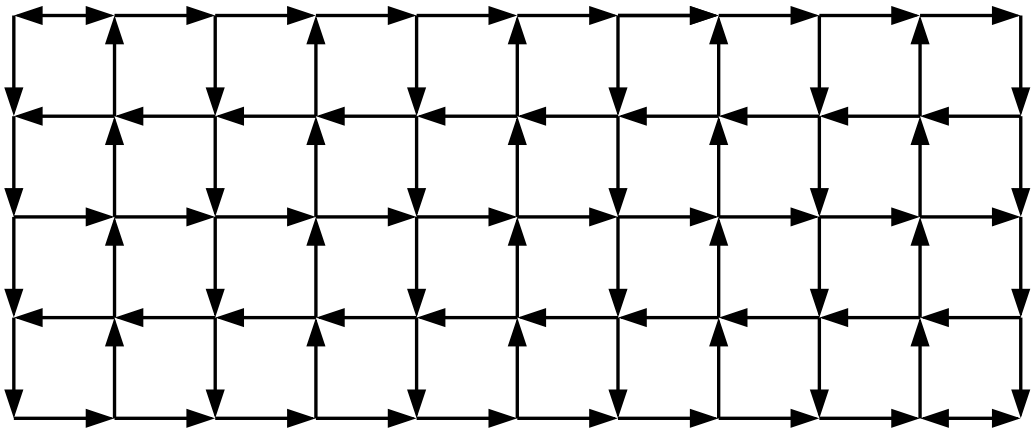


Figure 4.6: Directed grid graph

In such a directed grid graph the length of a shortest static path between two nodes s and t increases by at most 4 time units. This can also be

guaranteed if we restrict the number of bends of that path to 4. Figure 4.7 illustrates a worst case example.

We refer to DYN-ROUTE-SP in the directed grid graphs with the additional restriction that the determined shortest path contains at most 4 bends as DYN-ROUTE-SP *for grid graphs*.

In order to analyze the performance of DYN-ROUTE-SP for grid graphs we aim at bounding the number of conflicts of a static path with distinct time stamps. To this end, we introduce two terms in connection with static paths in a (directed) grid graph: Firstly, we call the union of edges and nodes between two bends a *straight*. Secondly, consider two static paths P and P' in a (directed) grid graph. We say that path P *crosses* path P' in node v if both paths contain this node and the preceding edges on the paths are different. Each such a situation is called a *crossing* of the paths P and P'.

We obtain Lemma 4.14 by bounding the number of crossings and the number of conflicts with distinct time stamps on a straight.

Lemma 4.14. *For each path P_i computed for a certain request $r_i = (s_i, t_i, \theta_i)$ by* DYN-ROUTE-SP *for grid graphs it holds that the duration Δ_{P_i} is bounded from above by*

$$\operatorname{dist}(s_i, t_i) + 5i - 1.$$

Note that $\operatorname{dist}(s_i, t_i)$ *denotes the shortest static path distance in the underlying undirected grid graph.*

Proof. First of all, note that all conflicts of a static path with a dynamic path (computed by DYN-ROUTE-SP) on consecutive edges of a straight have the same time stamp by definition. This also holds for the bidirectional edges since the straight in the opposite direction has length one.

Thus, the number of conflicts of a static path P with a dynamic path P' is bounded by the number of crossings of P and the static path $\hat{P}$ that corresponds to P'. Obviously, there are at least 2 bends needed between two crossings; either one of each considered path or two of one path. Thus, if b denotes the number of permitted bends, the number of crossings is bounded by $2b/2 + 1 = b + 1$. Hence, since we consider paths with at most 4 bends, each shortest static path has at most 5 conflicts with distinct time stamps with the same dynamic path computed by DYN-ROUTE-SP for grid graphs. This leads to a waiting time at the start node of at most $5(i-1)$ time units for the i-th request r_i.

Additionally, we have to take into account that the computed shortest path is at most 4 time units longer than a shortest path in the underlying (undirected) grid graph.

Both arguments together lead to the claimed bound of

$$\text{dist}(s_i, t_i) + 5(i-1) + 4 = \text{dist}(s_i, t_i) + 5i - 1$$

for path P_i. □

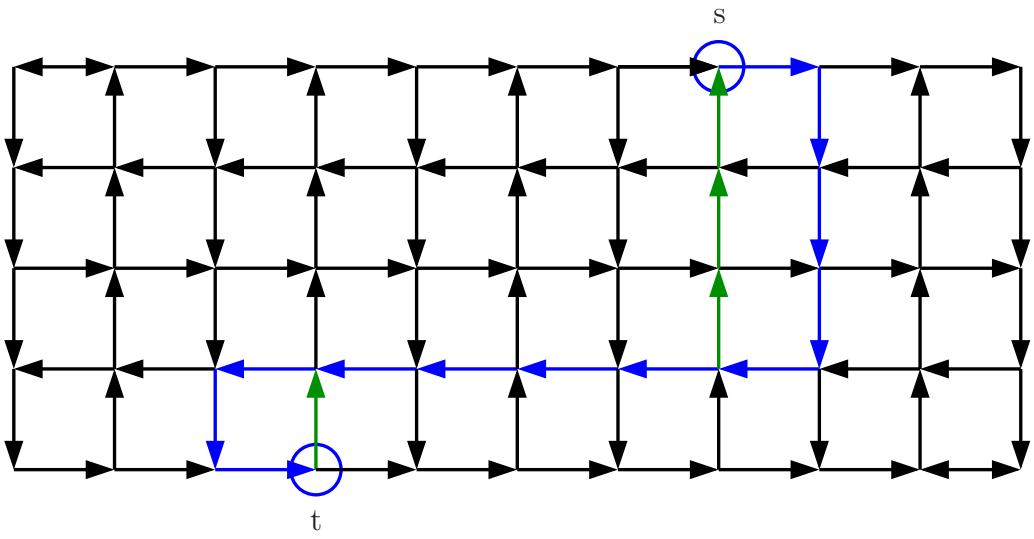


Figure 4.7: Illustration of a shortest static path between s and t in the directed grid (blue). The green edges show possible shortcuts if one considers the underlying undirected grid.

Lemma 4.9 and the upper bound for arbitrary undirected graphs (Corollary 4.12) lead to the competitive ratio given in Theorem 4.15.

Theorem 4.15. *Consider a sequence* $\sigma = r_1, \ldots, r_k$ *of requests* $r_i = (s_i, t_i, \theta_i)$. *Then,* DYN-ROUTE-SP *is* $\min\{k, 1+(5k-1)/L_{max}\}$*-competitive with respect to the OQDPP and* $\min\{k, 1+(5k+3)/2L_{avg}\}$*-competitive with respect to the OSDDPP in (undirected) grid graphs, where* $L_{max} := \max_i\{\theta_i + \text{dist}(s_i, t_i)\}$ *and* $L_{avg} := \sum_i \text{dist}(s_i, t_i)/k$.

Since our algorithm runs in polynomial time, this also improves the result of Spenke [37], who provided an (offline) $4 + (k-4)/L_{\max}$-approximation algorithm for the QDPP, if the maximum distance between source and target and/or the maximum release time of at least one request is large in compar-

ison to the number of requests ($L_{\max} > \frac{4}{3}k + 1$):

$$\begin{aligned} & 1 + \tfrac{5k-1}{L_{\max}} < 4 + \tfrac{k-4}{L_{\max}} \\ \Leftrightarrow \quad & \tfrac{4k+3}{L_{\max}} < 3 \\ \Leftrightarrow \quad & \tfrac{4}{3}k + 1 < L_{\max}. \end{aligned}$$

It is not unusual that such a condition holds in real-world instances. On the one hand, transportation requests appear over time and therefore the last request in a sequence might have a large release time. On the other hand, the distances that have to be covered in large scale logistic systems are enormous.

4.2 Experimental Performance Analysis

In order to measure the performance of our dynamic routing algorithm DYN-ROUTE with respect to the OSDDPP and the OQDPP empirically we consider optimal solutions as well as other dynamic routing approaches, namely the dynamic routing algorithm without waiting DYN-ROUTE-SP presented in Section 4.1.2 and the approximation algorithm of Spenke [37], for the purpose of comparison. The experiments are conducted in instances based on grid graphs.

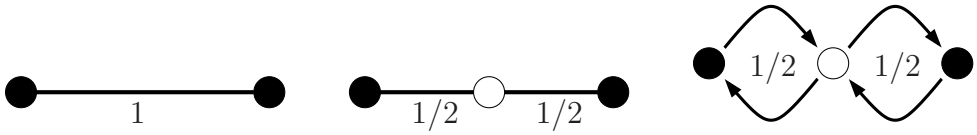
For the determination of optimal solutions we introduce a special (explicit) time expanded formulation of our problem and solve the corresponding offline problems (the SDDPP and the QDPP) via a static integer multicommodity flow approach. This requires a slight modification of our model. In fact, we relax the definition of a dynamic path a little. Note that the results of Section 4.1 also hold in this model. Moreover, of course, we use that slight relaxation not only for the determination of optimal solutions, but also for the routing approaches we are going to evaluate.

The section is organized as follows. After describing the mentioned modification, the time expansion, and the IP formulation in Section 4.2.1 we provide the test instances in Section 4.2.2. The computational results are presented in Section 4.2.3.

4.2.1 Optimal Solutions in a Slightly Modified Model

Modification of the Model

We slightly modify our model concerning the definition of a dynamic path in undirected graphs. More precisely, we permit that vehicles enter and leave an edge via the same node, i.e., vehicles can turn back in the middle of an edge. Formally, we change the definition of a dynamic path (Definition 2.3) such



(a) undirected edge (b) edge split (c) directed subgraph

Figure 4.8: Transformation of an undirected edge.

that for two consecutive nodes v_i and v_{i+1} it must hold that there is either an edge $e = v_i v_{i+1}$ (standard requirement) *or* $v_i = v_{i+1}$. In the latter case the edge that is used for the newly introduced movement have to be specified. We call such a path a *relaxed dynamic path*. Beyond the permission of these additional movements the model remains unchanged.

Although that modification does not exactly correspond to a split of each edge, it can be viewed as an increase of the level of discretization. Note that the relaxation does not change the analysis of Section 4.1. In particular, neither an optimal algorithm nor the dynamic routing approaches would benefit in the worst case examples given in Section 4.1.1.

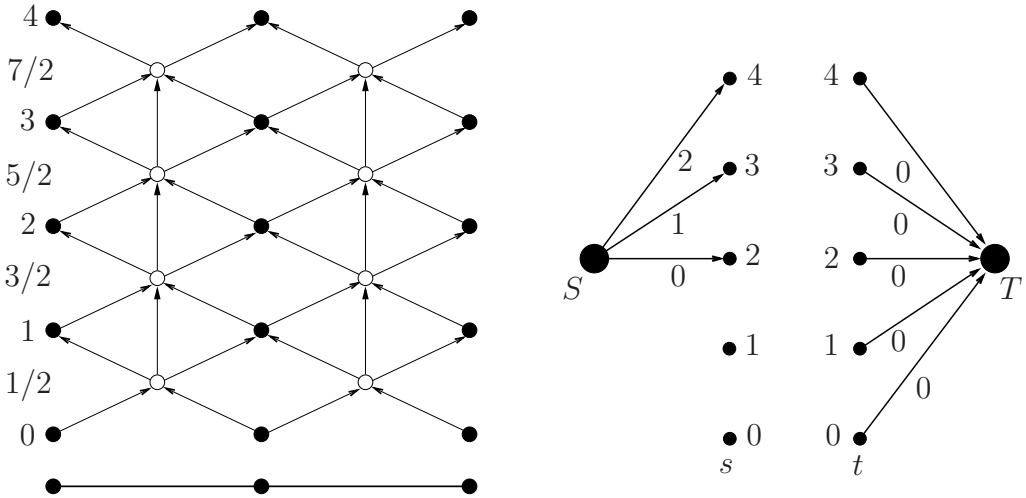
The advantage of that modification of the model is that we are able to give a time expanded graph formulation. Therefore, optimal solutions can be computed via a standard static multi-commodity flow approach. Both is described in the following sections.

Time Expansion

In this part we aim at constructing a time-expanded graph $G_T = (V_T, E_T)$ with time horizon T from a given undirected graph $G = (V, E)$ such that each relaxed dynamic path in G is represented by a static path in G_T and vice versa. To this end, we transform each undirected edge into a tiny directed graph. This transformation is illustrated in Figure 4.8. Firstly, we introduce artificial nodes on edges. More precisely, each edge $e \in E$ is split into two edges and each of these new edges has transit time $1/2$ (see Figure 4.8(b)). We call the new node an *artificial node*. Afterwards each undirected edge is replaced by two anti-parallel directed edges, cf. Figure 4.8(c).

Similar to a related transformation given by Spenke [37], this method leads to a directed graph that is obviously not equivalent to the original undirected graph since it provides additional paths, namely the paths that contain a subsequence v, w, v of nodes, where v is an original node and w is an artificial node. However, this is exactly what we permit by our relaxation.

Now we construct a time-expanded graph from that directed graph, see Figure 4.9. In each time step we provide a copy of each node and link these nodes according to the given transit time. Since waiting is permitted only



(a) Time expansion

(b) Super source S and super target T for a request $r = (s, t, 2)$.

Figure 4.9: Illustration of the time expansion and the introduced super node.

on edges we provide waiting edges between the artificial nodes that represent the original edges and not between original nodes.

Additionally, we add super sources and super targets for each request. While super sources are introduced to model waiting on the source node, super targets have to be considered to make all possible completion times available. Therefore, the costs (transit times) on the ingoing edges of the super target are set to 0 and the outgoing edges of the super source has transit times according to the elapsed time (from the release time on).

Obviously, if we consider the idealized model with the described relaxation, each dynamic path can be represented by a static path in such a time-expanded graph and vice versa. Moreover, these static paths are node-disjoint if and only if the corresponding relaxed dynamic paths are disjoint. In particular, due to the super sources, waiting at the source node does not create reservations in the static graph. Thus, the constructed time expanded graph provides a static characterization of our disjoint vehicle routing problem with the mentioned relaxation. Therefore, optimal solutions can be determined using standard integer multi-commodity flow formulations.

Remark 4.16 (Directed graphs)**.** *Since we focus on grid graphs in this section we introduce the time expansion for undirected graphs. It is easy to see that this would also be possible if one considers directed graphs. In addition, one does not have to use a relaxed model in this case.*

IP Formulation

To obtain optimal offline solutions for the Shortest Dynamic Disjoint Paths Problem (SDDPP) and the Quickest Disjoint Paths Problem (QDPP) we use standard edge-based multi-commodity flow formulations with binary variables x_{ei} that indicate whether the i-th request $r_i = (s_i, t_i, \theta_i)$ from a sequence $\sigma = r_1, \ldots, r_k$ uses edge e or not. Flow conservation constraints and setting the capacity on each node to one then leads to formulations that guarantee node disjointness of the determined paths. The objective functions are chosen according to the considered problems.

For the SDDPP, we get the following integer program:

$$
\begin{aligned}
\min \quad & \sum_{r_i \in \sigma} \sum_{e \in E} \tau(e) x_{ei} \\
\text{s.t.} \quad & \sum_{e \in \delta^+(v)} x_{ei} - \sum_{e \in \delta^-(v)} x_{ei} = \begin{cases} -1 & : \ v = T_i \\ 1 & : \ v = S_i \\ 0 & : \ otherwise \end{cases} && \forall v \in V_T, r_i \in \sigma \\
& \sum_{r_i \in \sigma} \sum_{e \in \delta^-(v)} x_{ei} \leq 1 && \forall v \in V_T \\
& x_{ei} \in \{0,1\} && \forall e \in E_T, r_i \in \sigma.
\end{aligned}
$$

A similar formulation is used for the QDPP:

$$
\begin{aligned}
\min \quad & C_{\max} \\
\text{s.t.} \quad & C_{\max} - \sum_{e \in E} \tau(e) x_{ei} \geq \theta_i && \forall r_i \in \sigma \\
& \sum_{e \in \delta^+(v)} x_{ei} - \sum_{e \in \delta^-(v)} x_{ei} = \begin{cases} -1 & : \ v = T_i \\ 1 & : \ v = S_i \\ 0 & : \ otherwise \end{cases} && \forall v \in V_T, r_i \in \sigma \\
& \sum_{r_i \in \sigma} \sum_{e \in \delta^-(v)} x_{ei} \leq 1 && \forall v \in V_T \\
& x_{ei} \in \{0,1\} && \forall e \in E_T, r_i \in \sigma.
\end{aligned}
$$

Both problems can be solved by standard IP solvers. We use CPLEX [23] for this purpose. Note that we are not interested in computation times and the evaluation of different CPLEX settings since this is a standard task for IP solvers. In contrast, our goal is to evaluate the performance of our dynamic routing approach.

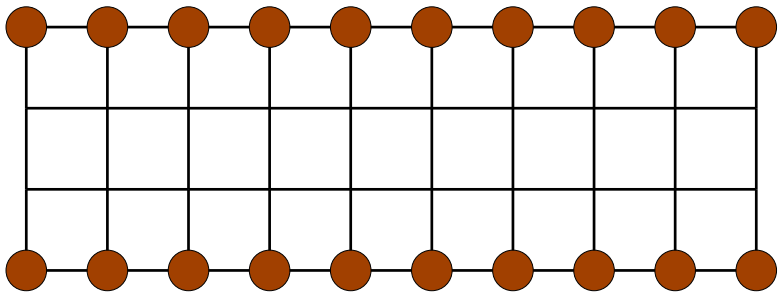
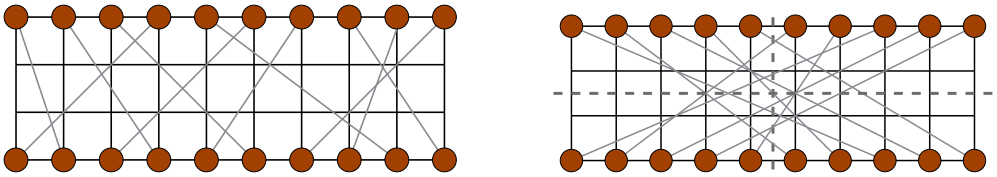


Figure 4.10: A grid graph with pick-up and delivery points on the upper and lower boundary.

4.2.2 Test Instances

For the experiments we consider grid graphs with pick-up and delivery points on the upper and lower boundary (Fig. 4.10). The graphs have 2, 4, or 6 horizontal lanes and 6, 10 or 20 vertical lanes, respectively. Among the variation of the dimension of the grid we distinguish three different request patterns, namely the BASE, the CROSSING, and the CROSSING-2 setting.



(a) BASE (b) CROSSING

Figure 4.11: Illustration of the BASE and the CROSSING setting.

BASE: Source and target of the requests are chosen randomly such that each pick-up and each delivery point is exactly once start or destination (Figure 4.11(a)). Thus, the number of requests depends on the grid size. It equals the number of vertical lanes. Moreover, we require that start and destination of each request lie on opposite sides of the grid, i.e., if the source node is positioned on the upper bound, the target node must be located on the lower bound and vice versa. The release time is randomly set to a value in $\{0, 1, 2, 3, 4\}$.

CROSSING: We extend the BASE setting by the restriction that the source node and the target node of requests have to be in opposite quadrants of the grid, i.e., we require that each request crosses the 'center' of the grid. Figure 4.11(b) illustrates a set of requests that fulfill this condition.

CROSSING–2: In this setting we generate a CROSSING instance and double the requests by repeating each request 4 time units later. We assume that both requests—the original one and the repeated one—do not interfere each other in the source and in the target node, respectively.

In each of the resulting test cases we consider 100 randomly chosen instances for the evaluations presented in the next section.

4.2.3 Computational Results

In this section we compare the dynamic routing approach DYN-ROUTE (Algorithm 1) with optimal offline solutions, on the one hand, and other routing approaches, namely the dynamic routing approach without waiting DYN-ROUTE-SP (Algorithm 5) introduced in Section 4.1.2 and the approximation algorithm of Spenke [37], on the other hand. This is done with respect to both objectives, the overall duration (OSDDPP) and the makespan (OQDPP).

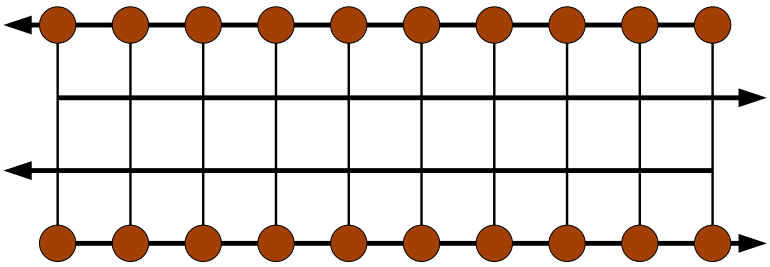


Figure 4.12: A grid graph with pick-up and delivery points on the upper and lower boundary and directed horizontal lanes.

Additionally, we consider a restricted variant of both introduced dynamic routing algorithms (DYN-ROUTE and DYN-ROUTE-SP) by applying directions to the horizontal lanes alternatingly (cf. Figure 4.12). This is due to the observations of Section 4.1. There we showed that the dynamic routing algorithm without waiting DYN-ROUTE-SP performs better if we assign directions to the edges of the grid graph (cf. Theorem 4.15). Note that we do not apply directions to the vertical lanes since we focus on grids with a small number of edges on a vertical lane. These variants are compared with the standard variants of the algorithms as well as with optimal solutions which are, of course, determined in the given undirected grid graph (without any restriction).

Remark 4.17 (Restricted variant of DYN-ROUTE and DYN-ROUTE-SP)**.** *We consider a restricted variant of* DYN-ROUTE *(Algorithm 1) and* DYN-ROUTE-SP *(Algorithm 5) in grid graphs. The algorithms are adapted such that horizontal lanes can only be used in one direction. The directions are applied alternatingly. We refer to these variants as the restricted* DYN-ROUTE *and the restricted* DYN-ROUTE-SP*, respectively.*

The section is organized as follows. At first, we compare the standard (unrestricted) dynamic routing algorithm DYN-ROUTE with optimal solutions. Afterwards we switch to the restricted variant of DYN-ROUTE and at last we consider the other mentioned routing approaches, the dynamic routing algorithm without waiting DYN-ROUTE-SP (with and without the restriction) as well as the approximation algorithm of Spenke and set them in contrast to DYN-ROUTE.

Comparison with Optimal Solutions

We consider the Online Shortest Dynamic Disjoint Paths Problem (OSD-DPP) first. Table 4.1 illustrates the performance of the dynamic routing approach with respect to the overall duration. We evaluate the optimality gap, which is defined as

$$\frac{\text{DYN-ROUTE}(\mathcal{I}) - \text{OPT}(\mathcal{I})}{\text{OPT}(\mathcal{I})}$$

for each instance $\mathcal{I}$, as well as the number of requests that are solved optimal by the dynamic routing algorithm DYN-ROUTE.

The general observation is that the optimality gap depends on three criteria: the structure of the requests, the narrowness of the grid, and the number of requests.

Concerning the *structure of the requests* we present two variants: the BASE instances with completely random requests and the CROSSING instances, where interferences between the requests are forced. The evaluation shows that, if we compare grids with the same dimension, the dynamic routing approach performs better in the completely random test scenarios (BASE). This is most significant in the scenarios with two horizontal lanes, which are the only instances in the BASE and CROSSING scenario, where our algorithm does not perform almost optimal. However, there is also an observable difference between the instances with four and six horizontal lanes, respectively. In general, the main observation concerning the *narrowness* is that the optimality gap increases with a decreasing number of horizontal lanes, if we compare scenarios with an identical number of vertical lanes.

	optimality gap (in %)			gap=0
	average	maximum	standard deviation	(in %)
BASE				
6×2	3.46	23.53	4.74	54
6×4	2.07	8.70	2.31	42
6×6	0.81	5.36	1.40	70
10×2	5.11	17.86	3.93	13
10×4	1.84	9.18	1.65	24
10×6	0.77	4.26	0.88	46
20×2	7.58	22.79	3.82	0
20×4	1.75	4.78	1.05	3
CROSSING				
6×2	5.01	21.62	4.50	26
6×4	2.74	10.42	2.36	26
6×6	1.06	5.00	1.36	54
10×2	10.62	28.57	6.62	3
10×4	2.54	9.00	2.03	14
10×6	0.90	4.17	0.97	40
20×2	15.77	31.67	8.22	0
20×4	3.68	6.33	1.33	0
CROSSING–2				
6×2	12.85	42.11	7.53	2
6×4	3.79	10.42	2.32	6
8×2	16.97	42.50	8.72	0
8×4	5.25	11.72	2.46	0

Table 4.1: Evaluation of the dynamic routing approach DYN-ROUTE in comparison with optimal solutions with respect to the OSDDPP. We consider the optimality gap (average, maximum, standard deviation) as well as the number of instances solved optimal.

Note that the percentage of optimal solutions behaves accordingly. Moreover, if we conversely fix the number of horizontal lanes, the optimality gap increases with the number of vertical lanes. The latter can also be viewed as an effect of an increasing *number of requests*. We observe the same in the CROSSING–2 setting, where the doubling of the number of requests leads to a noticeable performance loss in comparison to identical instances in the CROSSING setting. Thus, we conclude that all mentioned criteria have an influence independently of each other.

Apart from this general result we now turn our attention to some specific instances. In the smallest instances (6×2) in the BASE setting we observe an exceptionally large standard deviation compared with the average optimality gap, which is in line with the large number of optimal solutions and a large maximum optimality gap. In our opinion this is due to the small optimal values in the corresponding instances. Another interesting point is that the solution values in the instances based on the 6×6 and the 10×6 grid in the BASE setting as well as in the CROSSING setting are on average less than one percent away from the values of the corresponding optimal solutions. Moreover, the fraction of optimal solutions generated by the dynamic routing approach in these instances is between 40 and 70 percent, while in the instances with 20 vertical lanes in the CROSSING setting and in the instances with 8 vertical lanes in the CROSSING–2 setting no optimal solutions are determined.

The worst case instances are those based on grids with 2 horizontal lanes in the CROSSING–2 setting, which is not very surprising. It is interesting, however, that the solution value of DYN-ROUTE is at most about 42 percent away from the optimum in these instance. Moreover, the average deviation is far below that value.

The evaluation of the dynamic routing algorithm with respect to the makespan (OQDPP), which is showed in Table 4.2, is basically in line with the analysis presented for the OSDDPP. The variation of the narrowness, the request structure, and the number of requests lead to results of the same quality. The only difference is that we observe larger deviations between the instances of the same scenario. In fact, there are larger standard deviations, maximum gaps and also many more optimal solutions in relation to the evaluation with respect to the overall duration. One reason for that are the smaller values of the computed solutions compared with the overall duration case. Moreover, if we consider the makespan, we, in a way, only evaluate one request – the one with the latest completion time – and disregard the others. This can obviously lead to effects such as these. In particular, there are up to 92 percent optimal solutions.

	optimality gap (in %)			gap=0
	average	maximum	standard deviation	(in %)
BASE				
6×2	4.96	37.50	7.56	62
6×4	1.61	10.00	3.45	82
6×6	0.63	8.33	2.16	92
10×2	6.94	40.00	9.36	52
10×4	1.39	27.27	3.91	85
10×6	0.54	11.76	1.93	92
20×2	9.08	43.48	9.80	35
20×4	1.14	18.75	3.15	84
CROSSING				
6×2	5.93	35.27	6.30	30
6×4	2.22	16.67	4.43	78
6×6	1.28	21.34	3.95	88
10×2	15.05	66.67	13.48	26
10×4	1.76	18.75	3.96	80
10×6	0.81	11.76	2.18	87
20×2	25.61	50.00	11.35	4
20×4	7.38	36.36	8.70	38
CROSSING–2				
6×2	14.68	46.15	10.11	16
6×4	3.15	26.67	5.09	65
8×2	24.32	60.00	13.15	4
8×4	4.19	33.33	5.52	51

Table 4.2: Evaluation of the dynamic routing approach DYN-ROUTE in comparison with optimal solutions with respect to the OQDPP. We consider the optimality gap (average, maximum, standard deviation) as well as the number of instances solved optimal.

Restricted Dynamic Routing Algorithm

Now we consider the dynamic routing algorithm DYN-ROUTE (Algorithm 1) with the additional restriction that the edges on the horizontal lanes are directed alternatingly, cf. Figure 4.12. The results with respect to both objectives, the overall duration and the makespan, are illustrated in Table 4.3.

The main observation is that the performance of this approach is much better in instances with only 2 horizontal lanes. Moreover, if we consider the overall duration, the optimality gap does not increase with the number of vertical lanes and the values in the CROSSING setting do not differ significantly from those in the BASE setting. Only the doubling of requests in the CROSSING–2 setting leads to larger optimality gaps, but they are still much smaller than those determined by the standard dynamic routing approach. In addition, the number of optimally solved instances in these most narrow setting increases considerably. In the CROSSING-2 instances, this value is even larger than in instances with 4 horizontal lanes.

In the instances with 4 or 6 horizontal lanes we get almost the same results as in the case without the restriction to the direction of the lanes (with the exception of the 20×4 grid in the CROSSING setting in the makespan case). This observation holds for both objectives, while comparing the results among each other leads to the same observation as given for the standard case, namely a larger standard deviation if we consider the makespan. Thus, the dynamic routing approach with the presented restriction to the direction of the edges does not seem to be superior to the standard dynamic routing approach in general, but is so in extremely narrow grids.

Motivated by these results we also evaluated this restricted approach in the simulation environment introduced in Section 3.3.1, i.e., we applied the restriction to the scenarios SCEN-A and SCEN-B. These simulations lead to contrary results.

Remark 4.18 (Evaluation of the restricted DYN-ROUTE in SCEN-A and SCEN-B from Section 3.3.1)**.** *While we achieve an average duration of* 182.26 *sec. in* SCEN-A *and* 183.90 *sec. in* SCEN-B *without the restriction, the results in the restricted version are* 189.31 *sec. and* 190.69 *sec., respectively, which is a loss of performance of almost* 4 *percent in both cases.*

There are three reasons for that apparent discrepancy: Firstly, the underlying grid-like graph in the simulation environment has more than two horizontal lanes. In this case the test instances in this section do not show an advantage of the restricted approach either. The second reason is that in a simulation over several hours there might be sequences with many requests in the same direction in a particular time period and, at a later date, there

	overall duration optimality gap (in %)			gap=0	makespan optimality gap (in %)			gap=0
	avg	max	dev	(in %)	avg	max	dev	(in %)
BASE								
6×2	3.79	16.67	4.85	50	6.05	33.33	8.66	60
6×4	1.82	9.76	2.35	50	1.68	22.22	4.87	88
6×6	1.23	6.00	1.64	56	0.64	16.67	2.75	94
10×2	4.12	17.72	3.81	19	5.16	33.33	7.14	56
10×4	1.74	5.71	1.55	27	0.64	14.29	2.29	92
10×6	0.87	4.81	1.09	50	0.77	13.33	2.35	89
20×2	3.49	9.46	2.17	2	4.54	23.81	6.07	49
20×4	1.79	6.12	1.07	3	0.70	12.50	2.00	87
CROSSING								
6×2	3.53	16.22	4.32	46	4.43	30.00	7.36	68
6×4	2.28	8.33	2.00	30	1.47	9.09	3.14	82
6×6	0.60	3.33	1.09	74	0.71	7.69	2.12	90
10×2	3.78	14.81	3.69	25	7.68	41.67	9.40	41
10×4	2.37	7.84	1.73	11	1.25	13.33	2.88	83
10×6	1.09	4.17	1.04	30	0.98	11.76	2.59	86
20×2	3.65	6.50	2.98	1	9.44	36.36	10.43	34
20×4	2.51	4.32	0.90	0	0.71	7.69	1.80	85
CROSSING–2								
6×2	8.85	38.89	8.50	8	11.86	58.33	11.00	25
6×4	4.61	16.67	2.61	1	3.85	20.00	5.33	58
8×2	8.68	37.60	7.72	1	16.91	64.29	12.74	8
8×4	5.31	12.16	2.40	0	4.02	25.00	5.08	50

Table 4.3: Evaluation of the restricted dynamic routing approach in comparison with optimal solutions with respect to the OSDDPP (overall duration) and the OQDPP (makespan). We consider the optimality gap (average, maximum, standard deviation) as well as the number of instances solved optimal.

might be a sequence in the opposite direction. In such cases the possibility of a flexible use of the lanes is of value. Moreover, as a further explanation, the grid-like graph has many more vertical lanes and the requests usually do not stretch across the hole grid. Therefore, lanes might be used in different directions in different parts of the grid.

Thus, we conclude that the question which approach should be preferred, the one with directed horizontal lanes or the unrestricted one, cannot be answered generally. It highly depends on the considered scenario/application.

Comparison with Other Routing Approaches

After analyzing the performance of the dynamic routing algorithm DYN-ROUTE with respect to optimal solutions, we now aim at comparing it with other routing approaches. In fact, we consider the dynamic routing algorithm without waiting DYN-ROUTE-SP (Algorithm 5) presented in Section 4.1.2 and the approximation algorithm of Spenke [37] for the QDPP. Both algorithms determine shortest static paths and, on that basis, construct dynamic paths such that waiting only occurs at the source node. While DYN-ROUTE-SP is an online algorithm that, similar to DYN-ROUTE, iteratively computes routes when the corresponding requests arrive, the algorithm of Spenke is an offline approximation algorithm that first organizes the request in four groups and then processes the groups one after another. Therefore, we refer to this algorithm as FOUR-GROUPS.

Table 4.4 illustrates the average deviation of the solution values of these approaches from our dynamic routing algorithm DYN-ROUTEwith respect to the makespan (OQDPP) and the overall duration (OSDDPP). This gap is defined as

$$\frac{\text{ALG}(\mathcal{I}) - \text{DYN-ROUTE}(\mathcal{I})}{\text{DYN-ROUTE}(\mathcal{I})}$$

for each instance $\mathcal{I}$, where ALG denotes the investigated routing algorithm.

Concerning DYN-ROUTE-SP we also compare both algorithms with the introduced restriction to the directions of the lanes (cf. Remark 4.17). Note that we do not compare the restricted version of one algorithm with the unrestricted of the other since we already analyzed the influence of the restriction on the performance before. Moreover, algorithm FOUR-GROUPS is only faced with the unrestricted dynamic routing approach, but we will see that this does not play an important role since the differences between the restricted and the unrestricted version are small in comparison with the deviation we observe regarding the results of algorithm FOUR-GROUPS.

The evaluation shows that both approaches, the dynamic routing algorithm DYN-ROUTE and algorithm DYN-ROUTE-SP, lead to very similar

	overall duration		makespan		
	DYN-ROUTE-SP				FOUR-
	UNDIR	DIR	UNDIR	DIR	GROUPS
BASE					
6×2	1.19	1.98	1.27	2.47	98.85
6×4	0.01	0.68	0.56	0.55	130.98
6×6	0.12	0.01	0.33	0.48	138.11
10×2	2.08	1.19	2.91	1.43	132.98
10×4	0.38	1.03	0.14	0.96	155.31
10×6	0.21	0.54	0.16	0.39	173.36
20×2	2.01	0.70	2.79	0.57	167.51
20×4	0.53	0.49	0.10	0.27	198.80
CROSSING					
6×2	2.31	1.30	1.71	0.62	107.80
6×4	0.16	0.68	0.00	1.18	141.01
6×6	0.08	0.56	0.24	0.93	160.93
10×2	2.87	1.18	3.75	1.24	136.84
10×4	0.61	0.75	0.74	0.88	184.34
10×6	0.16	0.56	0.47	0.93	194.03
20×2	1.57	1.42	1.26	1.17	145.31
20×4	0.27	0.60	0.20	0.83	209.05
CROSSING–2					
6×2	1.41	2.52	1.19	2.45	48.62
6×4	2.23	1.92	2.57	1.91	85.00
8×2	2.75	3.16	2.77	2.75	62.11
8×4	1.62	1.28	1.38	2.18	111.08

Table 4.4: Analysis of the performance of the dynamic routing algorithm DYN-ROUTE in comparison with other routing approaches, namely the dynamic routing algorithm without waiting DYN-ROUTE-SP and the approximation algorithm FOUR-GROUPS of Spenke. We illustrate the average difference of these approaches to DYN-ROUTE (in %) with respect to the overall duration and the makespan, respectively. Regarding DYN-ROUTE and DYN-ROUTE-SP we distinguish between the case with directed horizontal lanes ('DIR') and the unrestricted variant ('UNDIR').

	overall duration				makespan			
	UNDIR		DIR		UNDIR		DIR	
	DYN	DYNS	DYN	DYNS	DYN	DYNS	DYN	DYNS
BASE								
6×2	32	16	32	4	20	8	16	0
6×4	32	26	32	16	10	4	18	8
6×6	20	16	18	22	12	4	4	0
10×2	52	27	39	5	29	15	17	2
10×4	43	29	51	12	11	7	13	1
10×6	47	19	39	13	9	4	9	2
20×2	63	32	64	13	40	15	11	4
20×4	60	29	62	25	13	10	9	2
CROSSING								
6×2	42	10	26	2	18	4	4	0
6×4	32	32	38	18	16	14	18	4
6×6	32	24	22	12	4	6	4	6
10×2	60	36	45	8	45	25	19	6
10×4	44	33	44	20	18	13	15	4
10×6	44	18	48	18	9	2	13	1
20×2	52	48	52	11	48	41	22	4
20×4	50	44	59	18	26	12	18	0
CROSSING–2								
6×2	58	33	54	5	37	24	34	7
6×4	71	20	63	25	35	11	33	10
8×2	61	33	72	6	51	28	38	4
8×4	69	22	69	24	34	19	36	15

Table 4.5: Analysis of the performance of the dynamic routing algorithm DYN-ROUTE in comparison with DYN-ROUTE-SP: For each setting it has been evaluated how often (in %) the corresponding approach lead to a better result with respect to the considered objective. Again, we distinguish between the restricted case ('DIR': directed horizontal lanes) and the unrestricted variant ('UNDIR'). Moreover, we abbreviate DYN-ROUTE by DYN and DYN-ROUTE-SP by DYNS, respectively.

results. However, there is a slight difference, in particular in those scenarios that turned out to be the most difficult ones, namely the instances with only 2 horizontal lanes. There, DYN-ROUTE has a noticeable advantage. CROSSING setting with respect to the makespan (restricted and unrestricted).

Concerning the evaluation of algorithm FOUR-GROUPS, again regarding Table 4.4, we observe that this approach is not able to keep up with the presented online algorithms. This does not mean that the approach of Spenke is unsuitable in general, but it should be used for other applications, namely for vehicle routing problems, where a huge number of request, say for a whole day, is known in advance and their release times are not fixed (or are set to 0, cf. [37]). While the advantage of the latter is obvious the results in the CROSSING-2 setting with twice as much requests as in the CROSSING instances support the first part of that conjecture since we observe smaller gaps in this case. Furthermore, note that small numbers of horizontal lanes as well as small numbers of vertical lanes lead to a better performance of the FOUR-GROUPS approach in comparison to DYN-ROUTE. While the latter is due to the short distances in the given graph, which reduces the waiting for the termination of a preceeding group of requests in this approach, the results in the grid graphs with 2 horizontal lanes show again that these instances are the most challenging for DYN-ROUTE.

4.3 CONCLUSIONS

In Chapter 3 we showed that the dynamic routing algorithm DYN-ROUTE (Algorithm 1) is suitable for real-time disjoint vehicle routing in practice. More precisely, we concluded that the algorithm computes the routes fast enough and is able to deal with several kinds of perturbations. Although we also gave a comparison to a trivial lower bound, we postponed the performance analysis to this chapter.

The analysis was divided into two parts. On the one hand, we investigated the theoretical performance of DYN-ROUTE and, on the other hand, we conducted experiments in order to get empirical results.

Regarding the theoretical side we used competitive analysis to determine the worst case performance of DYN-ROUTE. We showed that DYN-ROUTE is $\Theta(k)$-competitive with respect to the Online Shortest Dynamic Disjoint Path Problem (OSDDPP) and the Online Quickest Disjoint Path Problem (OQDPP), where k is the number of requests. In addition, we investigated another dynamic routing approach. More precisely, we introduced algorithm DYN-ROUTE-SP which computes a dynamic path based on a shortest static path such that waiting only occurs at the source node.

Besides providing almost similar results as for DYN-ROUTE we were able to show that this approach is $1+(5k-1)/L_{\max}$-competitive with respect to the OQDPP and $1+(5k+3)/2L_{\text{avg}}$-competitive with respect to the OSDDPP in grid graphs, where, roughly speaking, $L_{\max}$ and L_{avg} denote the maximum and average static distance between source and target of the considered requests. As a consequence, since the algorithm runs in polynomial time, this improves the (offline) approximation ratio for the QDPP in grid graphs of Spenke [37], who gave the first approximation algorithm for that problem, for instances with $L_{\max} > (4/3)k + 1$.

In the experimental part we compare DYN-ROUTE to optimal solutions, on the one hand, and to DYN-ROUTE-SP as well as to the approximation algorithm of Spenke, on the other hand. For the evaluation we use instances based on grid graphs of different dimensions with pick-up and delivery points on the upper and lower boundary. It turned out that DYN-ROUTE is superior to DYN-ROUTE-SP and to the approximation algorithm of Spenke. Moreover, we determined an average optimality gap between 1% and 17% (dependent on the dimensions of the grid) if we consider the OSDDPP. For the OQDPP we observed an average gap between 1% and 25%. The respective optimal solutions are determined using a integer programming approach based on a time-expanded formulation of the given graph.

One question certainly arises regarding these results: What kind of theoretical analysis reflects the 'real' performance of the dynamic routing approach DYN-ROUTE? Unfortunately, we were not able to provide an average case analysis that answers this question. The difficulty was to find enough instances (classes of instances) in which the algorithm provably performs well (better than in the order of k). Therefore, this problem remains open.

Chapter 5

Static Routing

In this chapter we present a different kind of routing approach for solving the introduced disjoint vehicle routing problems. While the routes, again, are computed iteratively, the main difference to the dynamic routing algorithm presented in Chapter 3 (Algorithm 1) is that we do not take time dependences into account during the route computation. In contrast, the idea is to compute a static shortest path with respect to a particular cost function. Therefore, we refer to this heuristic approach as the static routing approach. Collisions are avoided by a reservation procedure which can be regarded as the construction of a dynamic path at the execution time of the route.

We will see that, due to the absence of time-dependence, this routing approach has to be two-stage. Besides the route computation, which we consider in Section 5.2, we have to take care for deadlock situations. To this end, we give a deadlock prevention algorithm in Section 5.3. The resulting algorithm is then evaluated with respect to its real-life suitability, especially in contrast to the dynamic routing approach.

Parts of this chapter are published in [18].

5.1 Introduction

5.1.1 Static Routing

In a static approach for online disjoint vehicle routing problems one computes static paths in the given (street) network, ignoring their time-dependent nature. More precisely, one computes a standard shortest path, e.g., using Dijkstra's algorithm, with respect to edge costs $c(e)$ for each edge e consisting of the transit times $\tau(e)$ plus a load-dependent penalty cost which is a function of the number of routes that are already using this edge. So far only Guan and Moorthy [21] have investigated different kinds of penalty costs for that problem. In fact, they considered constant additive penalty costs, on the one hand, and distance-dependent costs, on the other hand. Besides not giving any theoretical performance guarantee for these approaches, the experimental evaluation does not show any major effect. We introduce a particular cost function that guarantees short routes while avoiding too much

congestion. Moreover, the practical benefit is shown in Section 5.4.1.

In such a routing approach the computed paths are, of course, not conflict-free. Hence, one needs an additional conflict management that, at execution time of the routes, guarantees that no collisions occur. This can be done by iteratively allocating to a vehicle the next part of its route (the *claim*) and block it for all other vehicles, see Figure 5.1.

Remark 5.1 (Minimum claim)**.** *For safety reasons the claim must block at least the distance needed to come to a complete stop, cf. Section 3.1.4 (distance-dependent safety tube).*

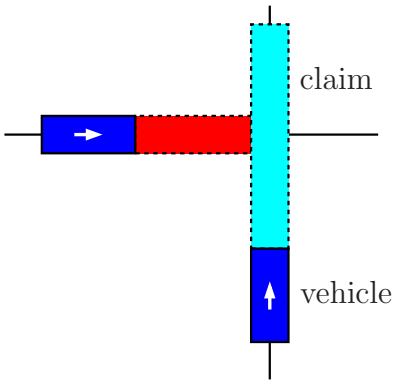


Figure 5.1: Reservation procedure. Each vehicle reserves the next part of the route. Mutual exclusive reservations guarantee a collision-free execution of computed (static) paths.

This *reservation procedure* can be viewed as construction of a dynamic path at execution time of the computed static path. Note that we do not concentrate on different kinds of reservation rules in this context. We assume a 'first come first served' strategy.

5.1.2 Drawbacks of Static Routing

In comparison to the dynamic routing approach presented in Chapter 3, the static routing approach a priori shows various drawbacks. The most alarming one is caused by the collision avoidance at execution time. The reservation procedure might cause *deadlocks*, as illustrated in Figure 5.2, which have a deteriorating effect on the system performance. Deadlocks appear if a group of vehicles wish to reserve a set of edges which are already occupied by another vehicle in this group such that none of them is able to continue its route and thus the system is blocked. Note that the dynamic routing approach already provides deadlock-free routes at the time of the route computation.

In addition to deadlocks, *detours* and *high congestion* may occur in the

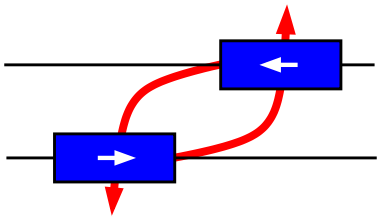


Figure 5.2: Simplified deadlock situation. Both vehicles are trying to occupy the same portion/edges of the network, thereby blocking each other.

static setting since time-dependent behavior is not modeled. This results in traveling times that can be far away from the shortest possible traveling time.

Moreover, a variety of other inbuilt drawbacks are due to the reservation at the execution time of the routes. For instance, actual arrival times of the vehicles at their destinations are completely random and cannot be predicted at the time of route computation (*unpredictable completion times*). This is a major drawback for other planning steps in the logistic chain that depend on the knowledge of these completion times. A special case in this context is the appearance of so-called *livelocks* which is the generic term for situations where a vehicle is blocked repeatedly by other vehicles without having the possibility of reserving the area which is next on its route. Note that, in the dynamic routing approach, the completion time is known immediately after route computation since the time-dependent behavior is fully modeled.

For the same reason the conflict times are unpredictable, i.e., it is not known at the time of the route computation whether or when two vehicles try to allocate the same portion of the network. As a consequence, *potential targets* can not be used for *transit purposes* since, if the transiting vehicle reaches the position later, it has to wait for an incalculable time (or a rerouting is required). We excluded such situations in our model (Assumption 2.6), but in practice it can be of value to have the flexibility of an unrestricted route computation.

Furthermore, it is *not clear how* a (efficient) reservation procedure that takes *priorities* into account could look like. Note that reserving the whole route in advance would lead to massive performance loss of the unprioritized requests.

The last three disadvantages (unpredictable completion time, no transit of potential target nodes, not clear how to prioritize requests) are inevitable consequences of the static approach. Therefore, we focus on the avoidance of deadlocks, congestion and detours in our implementation of the static routing approach.

5.1.3 Our Approach

In order to cope with the problem of congestion and detours we use a sophisticated routing approach that balances the load on the edges of the graph while the resulting paths remain short with respect to the transit times. More precisely, in Section 5.2, we present an online load-balancing algorithm that guarantees a minimal load under a given length constraint to the chosen paths.

In a second step we construct so-called reservation schedules to avoid potential deadlock situations. This is done by a deadlock detection and prevention algorithm introduced in Section 5.3.

Algorithm 6: STAT-ROUTE

Data: Graph $G = (V, E)$, sequence of requests $\sigma = r_1, \ldots, r_k$.
Result: Sequence of static paths $P_1, \ldots, P_k$ with corresponding deadlock-free reservation schedules.
begin
  foreach *request* r_j **do**
    · compute a shortest path w.r.t. a certain load-dependent cost function /∗ execute Algorithm 7 (Section 5.2.2) ∗/ ;
    · compute a deadlock-free reservation schedule /∗ execute Algorithm 10 (Section 5.3.4)∗/;
end

We refer to this two-stage approach as the static routing algorithm STAT-ROUTE (Algorithm 6). As in Chapter 3 we describe all algorithms based on the precise disjointness model (polygon model) introduced in Section 2.2.2 and give modifications for the idealized model whenever necessary. Moreover, note that the turning behavior and the orientation of the vehicles can be taken into account in the same way as for the dynamic approach, see Section 3.1.4.

We will not discuss the deletion of served routes in detail. We just assume that paths (and therefore the corresponding loads and reservation schedules) are removed when the corresponding vehicle has reached its target.

5.2 Online Load Balancing with Bounded Stretch Factor

5.2.1 Introduction

In the recent years many people have conducted research into the Online Load Balancing Problem [2, 8]: Consider a directed graph $G = (V, E)$ and

a sequence of requests $\sigma = (r_1 = (s_1, t_1), \ldots, r_k = (s_k, t_k))$, where s_i and t_i denote the source and target node of the request i, respectively. Sometimes a certain bandwidth b_i (resources needed by request r_i) is assigned to each request. We will focus on the case where this bandwidth is equal to one. In this case, the load on an edge e after the i-th request ($\text{load}_i(e)$) is defined as the number of requests already routed over e. The task is to minimize the maximum load over all edges, i.e., $\min \max_{e \in E} \text{load}_k(e)$, where k again denotes the number of requests.

Online Load Balancing Problem

Instance: Directed graph $G = (V, E)$, sequence $\sigma = (r_1, \ldots, r_k)$ of requests $r_i = (s_i, t_i)$.

Task: Choose a static path P_i for each request r_i such that the maximum load over all edges $e \in E$ is minimized, i.e., $\min \max_{e \in E} \text{load}_k(e)$

Aspnes et al. [2] gave an $O(\log(|E|))$-competitive algorithm for that problem and showed, using the lower bound of Azar, Naor and Rom [4] for online assignment, that their approach is optimal for online load balancing.

This standard load balancing problem has been extended to the case with transit times $\tau(e)$ on edges and a certain constraint to the length of a chosen path by Gao and Zhang [17]. In fact, they introduced a so-called stretch factor $B > 1$ that bounds the length of a chosen s_i-t_i-path. In fact, each s_i-t_i-path has to be shorter than B times the length of a shortest path between s_i and t_i. We call this problem the Online Load Balancing with Bounded Stretch Factor.

Online Load Balancing Problem with Bounded Stretch Factor

Instance: Directed graph $G = (V, E)$, transit times $\tau : E \to \mathbb{R}$, stretch factor B, sequence of requests $\sigma = (r_1, \ldots, r_k)$.

Task: Choose a static path P_i with $\text{length}_\tau(P_i) < B \cdot \text{dist}(s_i, t_i)$ for each request r_i such that the maximum load over all edges $e \in E$ is minimized.

Gao and Zhang modified the routing approach of Aspnes et al. and obtained similar results concerning the competitive ratio for this problem. In particular, they also provide a $O(\log(|E|))$-competitive algorithm. Since they compute resource-constrained shortest paths for each request their algorithm has an exponential run time .

We also consider the Online Load Balancing Problem with Bounded

Stretch Factor, but we focus on a different kind of analysis. Instead of comparing the solution of a particular online algorithm with the optimal solution for that problem (competitive analysis), we consider the optimal solution of the standard load balancing problem, i.e., without the given stretch factor constraint. We are interested in this ratio since an optimal load balancing solution can be seen as the best choice with respect to generated congestion in our static routing algorithm STAT-ROUTE (Algorithm 6). We call it the *stretch factor restricted (sfr) competitive ratio* and transfer the notation from the standard competitive analysis introduced in Section 2.1.2.

Definition 5.2 (Stretch factor restricted competitive ratio)**.** *An online algorithm* ALG *for the Online Load Balancing Problem with Bounded Stretch Factor is called* c-stretch-factor-restricted-competitive *or* c-sfr-competitive *if, for any problem instance* $\mathcal{I}$*, it achieves a solution with*

$$\mathrm{ALG}(\mathcal{I}) \leq c \cdot \mathrm{OPT}(\mathcal{I}),$$

where $\mathrm{OPT}(\mathcal{I})$ *denotes the value of the optimal offline solution for the Online Load Balancing Problem.*

The stretch factor restricted (sfr) competitive ratio *of* ALG *is the infimum over all* c *such that* ALG *is* c*-competitive.*

In addition, we assume that the number of requests k, or at least a good upper bound, is given in advance. Seiden, Sgall and Woeginger [35] call such approaches semi-online. In our application, the disjoint vehicle routing, a good upper bound might be the number of vehicles since there cannot be more 'active' requests than vehicles. Moreover, since we consider the polygon model, we take conflicting edges into account whenever we determine the load on the edges.

In the next section we provide a semi-online algorithm that turns out to be optimal with respect to the sfr competitive ratio.

5.2.2 Algorithm

We present an

$$O\left(\max\left(k, \log_{\sqrt[k]{B}} |E|, \log_{\sqrt[k]{B}} \frac{\max_{e\in E} \tau(e)}{\min_{e\in E} \tau(e)}\right)\right)\text{-sfr-competitive}$$

algorithm (Algorithm 7) for Online Load Balancing with Bounded Stretch Factor. The algorithm works in phases. In each phase we consider a certain upper bound UB to the optimal load with respect to the already routed requests in this phase. This upper bound is adjusted dependent on the

current maximum load produced by the algorithm, i.e., we enter a new phase (and double the bound) whenever it cannot be guaranteed that UB is still an upper bound. This is verified in line UB of the algorithm. We show the correctness of this check later (see Theorem 5.4).

Algorithm 7: BAL-BOUND

Data: Directed graph $G = (V, E)$, transit times $\tau : E \to \mathbb{R}$,
requests $\sigma = r_1, \ldots, r_k$, stretch factor B.
Result: Sequence of static paths $P_1, \ldots, P_k$.
begin
  $\text{load}(e) = 0 \ \forall e \in E$;
  $\text{UB} = 1$;
  $b = \sqrt[k]{B}$;
  foreach $r_i = (s_i, t_i)$ **do**
SP compute a shortest s_i-t_i path P_i w.r.t. the cost function
    $c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$;
UB **if** $\exists e \in \mathit{confl}(P_i)$: $\text{load}(e) + 1 > \log_b(b^{k+1} \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)}) \cdot \text{UB}$
    then
      /* new phase */
      $\text{UB} = 2 \cdot \text{UB}$;
      $\text{load}(e) = 0 \ \forall e \in E$;
      goto line SP /* repeat shortest path computation */;
    else
      $\text{load}(e) = \text{load}(e) + 1 \ \forall e \in \mathit{confl}(P_i)$;
      assign path P_i to request r_i;
end

For each request the algorithm computes a shortest path with respect to the cost function $c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$, where b is the k-th root of the given stretch factor B and load(e) is the current load on edge e (in the considered phase). Afterwards either a new phase is entered or the load on all edges that conflict with an edge of the selected path P, namely the edges in $\mathit{confl}(P) := \bigcup_{e \in P} \mathit{confl}(e)$, is increased by one. Note that, due to Definition 2.11 (to check for disjointness only the conflicting edges of one path have to be considered), the sets of conflicting edges are not taken into account during the route computation.

Obviously, each route computation is done at most two times since the first route computation in a new phase cannot violate the upper bound constraint (line UB of the algorithm). Thus, the algorithm runs in polynomial time.

Theorem 5.3. BAL-BOUND *(Algorithm 7) runs in polynomial time.*

In order to show the claimed stretch factor restricted competitive ratio of BAL-BOUND (Algorithm 7) we introduce SUB-BAL-BOUND (Algorithm 8) which can be viewed as subroutine (phase) of BAL-BOUND since there we assume that an upper bound UB is given in advance.

Algorithm 8: SUB-BAL-BOUND

Data: Directed graph $G = (V, E)$, transit times $\tau : E \to \mathbb{R}$, requests $\sigma = r_1, \ldots, r_k$, stretch factor B, upper bound UB $\geq$ OPT to the optimum.
Result: Sequence of static paths $P_1, \ldots, P_k$.
begin
  $\text{load}(e) = 0 \ \forall e \in E$;
  $b = \sqrt[k]{B}$;
  foreach $r_i = (s_i, t_i)$ **do**
    compute a shortest s_i-t_i path P_i w.r.t. the cost function $c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$;
    $\text{load}(e) = \text{load}(e) + 1 \ \forall e \in \mathit{confl}(P_i)$;
    assign path P_i to request r_i;
end

In Theorem 5.4 we provide a performance guarantee for Algorithm 8 that depends on the given upper bound UB.

Theorem 5.4. *For a fixed upper bound $UB \geq OPT$ and for any problem instance $\mathcal{I}$ it holds that*

$$\text{SUB-BAL-BOUND}(\mathcal{I}) \leq \log_b \left(b^{k+1} \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)} \right) \cdot \text{UB}.$$

Proof. Let P_i^* and P_i be an optimal path and the path selected by algorithm SUB-BAL-BOUND, respectively. Moreover, recall that $\text{load}_i(e)$ denotes the load generated by SUB-BAL-BOUND on edge e after i requests. Since the algorithm chooses the shortest path with respect to the considered costs $c(e)$, it holds that

$$\begin{array}{rlcl} & \sum_{e \in P_i} c(e) & \leq & \sum_{e \in P_i^*} c(e) \\ \Leftrightarrow & \sum_{e \in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} & \leq & \sum_{e \in P_i^*} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} \\ \Rightarrow & \sum_{e \in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} & \leq & \sum_{e \in P_i^*} \tau(e) b^{k} \end{array}$$

for all i since the load on each edge is bounded by k (by definition a static path contains no cycle). Summing up over all requests leads to the following inequality:

$$\begin{aligned} & \textstyle\sum_{i=1}^{k} \sum_{e\in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} \leq \sum_{i=1}^{k} \sum_{e\in P_i^*} \tau(e) b^k \\ \Leftrightarrow \quad & \textstyle\sum_{e\in E} \sum_{i:e\in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} \leq \sum_{e\in E} \sum_{i:e\in P_i^*} \tau(e) b^k \end{aligned}$$

Now we multiply both sides by $b^{\frac{1}{\text{UB}}} - 1$. For the inner sum of the left hand side we get

$$\begin{aligned} (b^{\frac{1}{\text{UB}}} - 1) \cdot \sum_{i:e\in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} &= \sum_{i:e\in P_i} \tau(e) \cdot (b^{\frac{\text{load}_{i-1}(e)+1}{\text{UB}}} - b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}}) \\ &\geq \sum_{i:e\in P_i} \tau(e) \cdot (b^{\frac{\text{load}_{i}(e)}{\text{UB}}} - b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}}) \quad (5.1) \\ &= \tau(e) \cdot (b^{\frac{\text{load}_{k}(e)}{\text{UB}}} - 1). \end{aligned}$$

The telescope sum in (5.1) is obtained by the observation that the load on an edge e increases at most by one in a single step ($\text{load}_i(e) \leq \text{load}_{i-1}(e)+1$).

On the right hand side we get

$$\begin{aligned} (b^{\frac{1}{\text{UB}}} - 1) \cdot \sum_{e\in E} \sum_{i:e\in P_i^*} \tau(e) b^k &\leq (\frac{1}{\text{UB}} b - 1) \cdot \sum_{e\in E} \sum_{i:e\in P_i^*} \tau(e) b^k \quad (5.2) \\ &\leq (\frac{1}{\text{UB}} b - \frac{1}{\text{UB}}) \cdot \sum_{e\in E} \sum_{i:e\in P_i^*} \tau(e) b^k \\ &\leq (b-1)\frac{1}{\text{UB}} \cdot \sum_{e\in E} \sum_{i:e\in P_i^*} \tau(e) b^k \\ &= b^k (b-1) \cdot \sum_{e\in E} \tau(e) \sum_{i:e\in P_i^*} \frac{1}{\text{UB}} \\ &\leq b^k (b-1) \cdot \sum_{e\in E} \tau(e). \quad (5.3) \end{aligned}$$

Here, Equation (5.3) holds since the number of paths that are routed over a certain edge in an optimal solution is bounded by UB. Moreover, the transformation in (5.2) is valid since $b > 1$ and $\text{UB} \geq 1$.

Using the results from both sides we get the claimed performance guar-

antee by simple arithmetic transformations (similar to those in [2]):

$$\begin{array}{lll}
& \sum_{e\in E}\tau(e)(b^{\frac{\text{load}_k(e)}{\text{UB}}}-1) & \leq\ b^k(b-1)\cdot\sum_{e\in E}\tau(e)\\
\Leftrightarrow & \sum_{e\in E}\tau(e)b^{\frac{\text{load}_k(e)}{\text{UB}}} & \leq\ (b^k(b-1)+1)\cdot\sum_{e\in E}\tau(e)\\
\Rightarrow & \sum_{e\in E}\tau(e)b^{\frac{\text{load}_k(e)}{\text{UB}}} & <\ b^{k+1}\cdot\sum_{e\in E}\tau(e)\\
\Rightarrow & \min_{e\in E}\tau(e)\sum_{e\in E}b^{\frac{\text{load}_k(e)}{\text{UB}}} & \leq\ b^{k+1}\cdot|E|\cdot\max_{e\in E}\tau(e)\\
\Leftrightarrow & \sum_{e\in E}b^{\frac{\text{load}_k(e)}{\text{UB}}} & \leq\ b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}\\
\Rightarrow & \max_{e\in E}b^{\frac{\text{load}_k(e)}{\text{UB}}} & \leq\ b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}\\
\Leftrightarrow & \max_{e\in E}\text{load}_k(e) & \leq\ \log_b(b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)})\cdot\text{UB}.
\end{array}$$

□

Aspnes et al. [2] showed that the approach of adapting the upper bound accordingly (BAL-BOUND), instead of assuming that such an upper bound is given (SUB-BAL-BOUND), increases the competitive ratio by at most a factor of 4. We use their approach to prove Theorem 5.5.

Theorem 5.5. BAL-BOUND *is* $4\cdot\log_b(b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)})$*-sfr-competitive. Thus, the stretch factor restricted competitive ratio is in*

$$O\left(\max\left(k,\log_{\sqrt[k]{B}}|E|,\log_{\sqrt[k]{B}}\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}\right)\right).$$

Proof. For readability we firstly introduce some notations. We write BAL-BOUND$_{\text{UB}}$ if we consider Algorithm 7 with given upper bound UB and abbreviate the guaranteed performance ratio of SUB-BAL-BOUND by $c := \log_b(b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)})$. Recall that SUB-BAL-BOUND can be viewed as subroutine of BAL-BOUND.

Let $\sigma^{(\ell)}$ denote the subsequence of requests in phase ℓ (upper bound 2^ℓ) of algorithm BAL-BOUND. Then, consider the phase h where the algorithm terminates. If the algorithm terminates in the first phase ($h = 0$) it holds that

$$\text{BAL-BOUND}(\sigma) = \text{BAL-BOUND}(\sigma^{(0)}) \leq c.$$

Therefore, it remains to show that the claimed competitive ratio holds for any $h \geq 1$. Consider the subsequence $\sigma^{(h-1)}$ and the first request r_1^h in phase h. This is the request that terminates phase $h-1$. Thus, it holds that

$$\text{BAL-BOUND}_{2^{h-1}}(\sigma^{(h-1)}, r_1^h) > c\cdot 2^{h-1}.$$

By Theorem 5.4 it follows that

$$\text{OPT}(\sigma) \geq \text{OPT}(\sigma^{(h-1)}, r_1^h) > 2^{h-1}.$$

Then, summing up over all phases leads to the claimed stretch factor restricted competitive ratio:

$$\begin{aligned} \text{BAL-BOUND}(\sigma) &= \sum_{\ell=1}^{h} \text{BAL-BOUND}_{2^\ell}(\sigma^{(\ell)}) \\ &\leq \sum_{\ell=1}^{h} c \cdot 2^\ell = (2^{h+1} - 1) \cdot c \\ &< 4 \cdot c \cdot 2^{h-1} < 4 \cdot c \cdot \text{OPT}(\sigma). \end{aligned}$$

□

Remark 5.6. *It is also possible to argue that BAL-BOUND never enters a new phase to show the claimed asymptotic sfr competitive ratio.*

Now it remains to show that the stretch factor constraint is respected by each path that is computed by BAL-BOUND.

Theorem 5.7. *Each path selected by* BAL-BOUND *is shorter (with respect to τ) than B times the shortest path length.*

Proof. Let P_i be the path selected by the algorithm for the i-th request and let SP_i be a shortest s_i-t_i-path. Then, it holds that

$$\begin{aligned} & \textstyle\sum_{e \in P_i} c(e) && \leq && \textstyle\sum_{e \in \text{SP}_i} c(e) \\ \Leftrightarrow\ & \textstyle\sum_{e \in P_i} \tau(e) \cdot b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} && \leq && \textstyle\sum_{e \in \text{SP}_i} \tau(e) \cdot b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}}. \end{aligned}$$

Since the static shortest path does not contain a cycle it holds that $\text{load}_{i-1}(e) < i$ for all i. This leads to the following estimate for each i:

$$\begin{aligned} & \textstyle\sum_{e \in P_i} \tau(e) \cdot b^0 && < && \textstyle\sum_{e \in \text{SP}_i} \tau(e) \cdot b^i \\ \Leftrightarrow\ & \text{length}_\tau(P_i) && < && b^i \cdot \text{dist}(s_i, t_i). \end{aligned}$$

Using $b = \sqrt[k]{B}$, we get the claimed bound to the path length, i.e.,

$$\text{length}_\tau(P_i) < B \cdot \text{dist}(s_i, t_i) \quad \forall i.$$

□

The adjustment of the algorithm for the idealized model is straightforward.

Idealized disjointness model. Besides using a set of conflicting edges that only contains the considered edge itself and the corresponding anti-parallel edge, it would be possible to introduce load on nodes (for instance by representing them via dummy edges) in the presented routing approach. The analysis of the presented algorithm BAL-BOUND (Algorithm 7) does not change in this case.

5.2.3 Lower Bound

In order show that no (online) algorithm can achieve a better performance guarantee with respect to the stretch factor restricted competitive ratio than BAL-BOUND (Algorithm 7) we provide the following example.

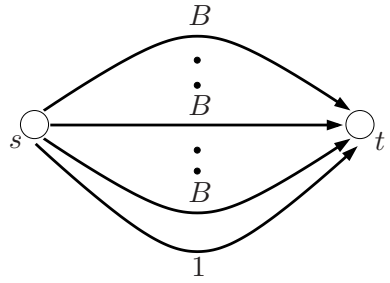


Figure 5.3: Graph described in Example 5.8.

Example 5.8. *We consider an instance of the Load Balancing Problem with Bounded Stretch Factor with stretch factor B and k requests from s to t in the graph illustrated in Figure 5.3. The graph consists of two nodes, s and t, and k parallel edges. On $k-1$ of these edges the transit time is set to B while the remaining edge has transit time 1. For each edge the set of conflicting edges only contains the edge itself.*

In this example, only one edge (transit time 1) can be used since routing over the other edges would violate the stretch factor constraint. Hence, no algorithm for the Online Load Balancing with Bounded Stretch Factor achieves a better load than $|E| = k$. Thus, since in an optimal solution without this constraint, one would assign each request to an exclusive edge, which leads to a maximum load of 1, the sfr competitive ratio is bounded from below by

$$|E| = k = \log_{\sqrt[k]{B}}(B) = \log_{\sqrt[k]{B}} \left(\frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)} \right).$$

This shows that the presented algorithm BAL-BOUND (Algorithm 7) is (asymptotically) optimal with respect to that ratio, cf. Theorem 5.5.

Remark 5.9. *Note that even an offline algorithm that respects the stretch factor constraint would not be able to perform better in the instance considered in Example 5.8.*

5.3 Reservation Schedules and Deadlock Prevention

5.3.1 Introduction

Although the routes computed by Algorithm 7 (BAL-BOUND) are good in the sense that they balance the load on the edges of the given graph we need a mechanism to definitely avoid conflicts. In Section 5.3.2 we introduce a reservation schedule that prevents the vehicles from colliding by requesting edges before occupying them. Such a schedule can be interpreted as an instruction for the construction of a dynamic path at execution time of the static paths. Note that we do not give the time when edges should be reserved in this schedule. This is done by the reservation procedure, cf. Section 5.1.1.

Since this procedure may cause deadlocks, we additionally have to take care for such situations. Due to the deteriorating effects of deadlocks on logistic processes a great deal of attention has been paid to that field of research in recent years. Besides the investigation of petri net approaches [30, 41] there are also graph-theoretic models [10, 21, 42].

The first observation concerning these approaches is that they use so-called zone control; i.e., it is assumed that vehicles move between non-intersecting zones of adequate size. This model is not suitable for our purpose since such a partition of a traffic network into zones is too imprecise. Kim et al. [26] introduce a finer zoning of the network, but, just as the zone control approaches, their algorithm is not able to deal with large-scale vehicle fleets.

Our deadlock prevention algorithm provides both, a fine discretization (vehicles move on the edges of the graph) and a fast routing in large networks with many vehicles. The algorithm is based on the detection of specific cycles, instead of standard cycles as in [10, 26, 42], in a graph that is in a one-to-one correspondence with the considered reservation schedule: the so-called deadlock detection graph. Moreover, in contrast to Guan and Moorthy [21] we already avoid deadlocks at the time of the route computation and not during the execution of the route.

The section is structured as follows: After the description of our model we introduce the deadlock detection graph in Section 5.3.3 and present the deadlock prevention algorithm in Section 5.3.4. Computational results and conclusions are given afterwards.

5.3.2 The Model

Consider a directed graph $G = (V, E)$ and a set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of static paths that have to be scheduled deadlock-free. Then, a *reservation schedule* $S(\mathcal{P}) = \{R_{P_1}, \ldots, R_{P_k}\}$ consists of a set of so-called *requested edges* $R_P : E(P) \to 2^E$ for each path $P \in \mathcal{P}$. These edge sets $R_P(e)$ denote those edges that must be free, i.e., not occupied by another vehicle, before edge e is left. If this is the case the requested edges are reserved and the corresponding vehicle can leave edge e. Note that for each path $P = (e_1, \ldots, e_n)$ the set of requested edges of the last edge, $R_P(e_n)$, is always empty.

Remark 5.10 (Resulting claim length)**.** *For the reservation procedure described in Section 5.1.1 a reservation of a certain edge $f \in R_P(e)$ means that at least the complete region from e to f, i.e., the corresponding polygons, have to be reserved (claimed). If such a claim does not cover the minimum distance (Remark 5.1) it is elongated accordingly.*

To guarantee a smooth execution of a reservation schedule, especially to avoid conflicts, it is necessary that each edge on a certain path has to be requested/reserved before it is entered. Such a reservation schedule is-called *valid*.

Definition 5.11 (Valid reservation schedule)**.** *A reservation schedule $S(\mathcal{P})$ is* valid *if for each path $P = (e_1, \ldots, e_n) \in \mathcal{P}$ it holds that*

$$\forall\ i \in \{2, \ldots, n\}\ \exists\ j \in \{1, \ldots, i-1\} \text{ such that } e_i \in R_P(e_j).$$

We assume that the first edge of the considered path does not have to be reserved since it is already occupied by the vehicle before it starts traveling. Therefore, we can also require that the first edge is not used by another vehicle.

Assumption 5.12 (Reservation of the first edge)**.** *The first edge of each path $P \in \mathcal{P}$ contained in a reservation schedule $S(\mathcal{P})$ is not used, and therefore not requested, by another path $P' \in \mathcal{P}$. Note that this is in line with Assumption 2.6.*

A reserved edge is freed when the vehicle leaves that edge. Thus, we are able to define another edge set, the *occupied edges* $O_P : E(P) \to 2^E$. An edge is called occupied if it has already been reserved and has not been freed. Thus, for an edge e_i on a static path $P = (e_1, \ldots, e_n)$ the set of occupied

edges is defined as

$$O_P(e_i) = e_i \cup \left(\bigcup_{k<i} \{e_j \in R_P(e_k) |\ j > i\} \right).$$

After that general introduction of the reservation schedule we are now going to describe how we apply the concept of conflicting edges, cf. Section 2.2.2 (polygon model), before we focus on the identification of deadlocks in such a schedule.

Conflicting edges. To guarantee disjointness of the executed routes, we can either reserve all edges in $\mathit{confl}(R_P(e)) := \bigcup_{f \in R_P(e)} \mathit{confl}(f)$ after a successful request of the edges in $R_P(e)$ or we can only reserve the edges in $R_P(e)$ after requesting the edges in $\mathit{confl}(R_P(e))$ (cf. Definition 2.11). We decide in favor of the latter variant since this leads to a one-to-many path computation—instead of a many-to-many path computation—in the key procedure of the deadlock prevention algorithm (Algorithm 11). To this end, we distinguish between requested edges in $R_P(e)$ and those in $\mathit{confl}(R_P(e))$ from now on.

Since we aim at providing a reservation schedule that avoids deadlocks, we have to identify *potential deadlock situation*. Here, 'potential' means that the identified situation need not appear but cannot be excluded. The reason for this vagueness is that both are causal for a deadlock situation, the reservation schedule and the order vehicles want to pass the conflict points. Note that the latter cannot be influenced by a reservation schedule. Therefore, we have to take each possible order of the vehicles into account.

Definition 5.13 (Deadlock-free reservation schedule)**.** *Let $S(\mathcal{P})$ be a reservation schedule. Then, a set of paths $\{P_1, \ldots, P_m\} \subseteq \mathcal{P}$ is called* deadlock-ridden *if, for each $i \in \{1, \ldots, m\}$, there exists an edge e_{P_i} such that*

$$\mathit{confl}(R_{P_i}(e_{P_i})) \cap \bigcup_{j \in \{1,\ldots,m\}: j \neq i} O_{P_j}(e_{P_j}) \neq \emptyset.$$

Accordingly, $S(\mathcal{P})$ is called deadlock-free *if the set of paths $\mathcal{P}$ does not contain any deadlock-ridden subset.*

Obviously, deadlocks can only occur if the affected vehicles are stopped which is only the case if a requested edge is not free. Thus, the distance-dependent safety-tube (Remark 5.1) does not have to be taken into account in the deadlock detection process since such a tube is missing if the vehicle is

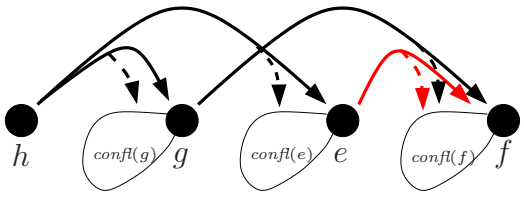
not moving. For the same reason there is no need for an explicit consideration of node reservations in conjunction with potential deadlocks in the idealized model since waiting is only permitted on edges.

Idealized disjointness model. Since waiting on nodes is not permitted in this model, it is sufficient to avoid node conflicts by reserving a node together with the next edge on the path.

Based on the described model we will now concentrate on the detection and prevention of deadlocks. To this end, we introduce a so-called deadlock detection graph that represents a reservation schedule in Section 5.3.3 and provide a deadlock prevention algorithm that makes use of this correlation in Section 5.3.4.

5.3.3 Deadlock Detection Graph

The question we will answer in this section is how we can detect a potential deadlock, i.e., a deadlock-ridden set of paths, in a given reservation schedule. We will show that this can be interpreted as a cycle with a specific property in a special kind of graph.



edge	requested edges
h	e, g
g	f
e	$\emptyset$
f	$\emptyset$

Figure 5.4: Illustration of a deadlock detection graph that corresponds to the reservation schedule shown on the right of a single path $P = (h, g, e, f)$. While the black edges directly result from the set of requested edges, the red edges are inserted since edge e is in $O_P(g)$ and edge f is requested from g (is in $R_P(g)$), respectively.

Such a *deadlock detection graph* $G_D = (V_D, E_D)$ is constructed based on a reservation schedule $S(\mathcal{P})$. The node set V_D consists of all edges of the underlying layout graph $G = (V, E)$, i.e., $V_D = E$. The edge set E_D consists of edges $((e, f), c)$, where $c \in C$ assigns a particular color to that edge. The color set $\mathcal{C}$ contains a different color for each path of the set $\mathcal{P}$. We denote the color of path P by c_P.

Definition 5.14 (Deadlock detection graph)**.** *Consider a graph $G = (V, E)$ and a reservation schedule $S(\mathcal{P})$. Then, the corresponding* deadlock detection graph *$G_D = (V_D, E_D)$ is constructed as follows:*
i) $V_D = E$,
ii) $E_D = \{((e, f), c_P) |\ \exists g \in E$ such that $f \in \mathit{confl}(R_P(g))$ and $e \in O_P(g)\}$.

Figure 5.4 illustrates the construction of a deadlock detection graph from a reservation schedule $S(\mathcal{P})$ (from a collection of sets of requested edges). Obviously, since by definition for each edge $e \in P$ on a certain path P it holds that $e \in O_P(e)$, each requested edge $f \in \mathit{confl}(R_P(e))$ in a reservation schedule leads to an edge $((e, f), c_P)$ in the reservation graph. However, there are additional edges added to the graph.

Since we are able to construct a deadlock detection graph from an arbitrary reservation schedule we are going to use this representation to detect potential deadlocks. To this end, we introduce so-called colorful paths and cycles according to [1] where Alon, Yuster, and Zwick considered the corresponding node version in a completely different context. They focus on the determination of simple paths and cycles of a specified length.

Definition 5.15 (Colorful path, colorful cycle)**.** *Consider a graph G with colored edges. Then, a* colorful path *is a directed static path P in G with the property that all edges on P are colored differently. A* colorful cycle *is a directed cycle with the same property.*

We will now use the fact that a colorful cycle in a deadlock detection graph characterizes a deadlock-ridden set of paths in the corresponding reservation schedule to show that a reservation schedule is deadlock-free if and only if the corresponding deadlock detection graph contains no colorful cycle.

Theorem 5.16. *A reservation schedule $S(\mathcal{P})$ in a directed graph $G = (V, E)$ is deadlock-free if and only if the corresponding deadlock detection graph $G_D = (V_D, E_D)$ contains no colorful cycle.*

Proof. By Definition 5.14 a deadlock detection graph contains a colored edge $((e, f), c_P)$ if and only if there is an edge $g \in E$ with

$$f \in \mathit{confl}(R_P(g)) \quad \text{and} \quad e \in O_P(g). \tag{5.4}$$

To prove the theorem we firstly assume that there is a colorful cycle

$$((e_1, e_2), c_{P_1}), \ldots, ((e_m, e_{m+1} = e_1), c_{P_m})$$

in the deadlock detection graph. Then, from Eq. (5.4) it follows that there

is an edge $e_{P_i} \in E$ for each path $P_i \in \mathcal{P}$ $(i = 1, \ldots, m)$ such that

$$e_i \in \mathit{confl}(R_{P_i}(e_{P_i})) \quad \text{and} \quad e_{i+1} \in O_{P_i}(e_{P_i}).$$

Therefore, the set of paths $\{P_1, \ldots, P_m\} \subseteq \mathcal{P}$ is deadlock-ridden (cf. Definition 5.13) since

$$e_{i+1} \in \mathit{confl}(R_{P_{i+1}}(e_{P_{i+1}})) \cap O_{P_i}(e_{P_i}) \quad \forall i = 1, \ldots, m,$$

which proves the first part of the statement (sufficiency).

To see necessity, assume that there is a deadlock-ridden set of paths $P_1, \ldots, P_m \subset \mathcal{P}$. Hence, there is an edge e_{P_i} on each of these paths such that

$$\mathit{confl}(R_{P_i}(e_{P_i})) \cap \bigcup_{j \in \{1,\ldots,m\}: j \neq i} O_{P_j}(e_{P_j}) \neq \emptyset \quad \forall i \in \{1, \ldots, m\}.$$

Thus, for all $i \in \{1, \ldots, m\}$ there is an edge e_i with

$$e_i \in \mathit{confl}(R_{P_i}(e_{P_i})) \cap O_{P_{j(i)}}(e_{P_{j(i)}})$$

for some $j(i) \in \{1, \ldots, m\}$. Using Equation (5.4) we get that there must be an edge $((e_i, e_{j(i)}), c_{P_i})$ for all $i \in \{1, \ldots, m\}$ and a corresponding $j(i) \in \{1, \ldots, m\}$ in the deadlock detection graph since

$$e_{j(i)} \in \mathit{confl}(R_{P_{j(i)}}(e_{P_{j(i)}})) \quad \text{and} \quad e_i \in O_{P_{j(i)}}(e_{P_{j(i)}}).$$

Thus, there must be a colorful cycle in that graph. Note that such a cycle does not have to contain an edge for each path in $\{P_1, \ldots, P_m\}$ since we do not demand in Definition 5.13 that a deadlock-ridden set of paths is inclusion-wise minimal. □

5.3.4 Deadlock Prevention Algorithm

Theorem 5.16 shows that the recognition of colorful cycles is the key ingredient of any deadlock prevention algorithm in the considered model. More precisely, one has to check whether a new reservation closes a colorful cycle in the deadlock detection graph. To this end, we consider the Colorful Path Problem. We will see later how solving this problem helps us to avoid colorful cycles. Note that, as mentioned before, Alon, Yuster, and Zwick [1] investigated a node variant of that problem.

COLORFUL PATH PROBLEM

Instance:	Directed graph $G = (V, E)$ with colored edges, color set $\mathcal{C}$, target node $t \in V$, set of source nodes $S \subset V$.
Task:	Is there a colorful path from some $s \in S$ to t that does not use a specific color $c \in \mathcal{C}$?

The Colorful Path Problem is $\mathcal{NP}$-complete since the edge version of the Path with Forbidden Pairs Problem, which is known to be $\mathcal{NP}$-complete [16], can be reduced to this problem.

Path with Forbidden Pairs [16]	
Instance:	Directed graph $G = (V, E)$, source node and target node $s, t \in V$, collection $\mathcal{D} = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ of edge pairs.
Task:	Is there a path from s to t in G that contains at most one edge from each pair in $\mathcal{D}$?

Theorem 5.17. *The Colorful Path Problem is $\mathcal{NP}$-complete.*

Proof. Consider an instance I of the Path with Forbidden Pairs Problem with collection $\mathcal{D}$. We construct an instance I' of the Colorful Path Problem by assigning each of the two edges contained in an edge pair of the collection $\mathcal{D}$ the same color and contracting all edges that are not contained in $\mathcal{D}$. Additionally, we choose a dummy color that is not contained in the graph as the forbidden color $c \in \mathcal{C}$ and set $S = \{s\}$.

Then, obviously, a colorful s-t-path in I' corresponds to a path with forbidden pairs in I and vice versa. □

Algorithm 9 solves the Colorful Path Problem. It is related to the algorithm introduced by Alon, Yuster and Zwick [1] for the corresponding node version.

The algorithm iteratively computes all colorful paths from length 1 to length $|\mathcal{C}| - 1$. More precisely, it maintains the information which nodes can be reached via a colorful path. To this end, each label consists of a node and a collection of colors C that contains the colors used on the corresponding path. Note that we do not propagate labels with redundant information, i.e., we do not add the same label again. This can be guaranteed by a lookup in a table that provides, for each possible combination of colors, the information of whether this combination has already been represented by a label on a certain node.

Since we, in contrast to Alon, Yuster and Zwick, consider multiple sources and only a single target, the graph is traversed backwards; i.e., the algorithm starts from the given target node and considers the ingoing edges for each label taken from the set Q_{old}, which, in phase i of the algorithm, contains the collection of possible colorful paths of length $i - 1$.

The algorithm obviously determines all possible colorful paths that do not contain the forbidden color and therefore solves the Colorful Path Problem.

Algorithm 9: COLORFUL-PATH

Data: Directed graph $G = (V, E)$ with colored edges, node $t \in V$, set of nodes $S \subset V$, set of colors $\mathcal{C}$, forbidden color $\hat{c}$.

Result: Is there a colorful s-t-path for some $s \in S$ that does not use the forbidden color $\hat{c}$?

```
begin
    Q_old = {(t, ∅)};
    Q_new = ∅;
    for i = 1; i < |C|; i++ do
        foreach (v, C) ∈ Q_old do
            if v ∈ S then
                return true ;
            foreach in-going edge ((u, v), c) do
                if c ∉ C ∪ {ĉ} and there is no label (u, C*) in Q_new
                 with C* = C ∪ {c} then
                    add (u, C ∪ {c}) to Q_new ;
        Q_old = Q_new;
        Q_new = ∅;
    return false ;
end
```

For the analysis of the run time we refer to [1] since it is similar to the node version. In particular, the additional consideration of a forbidden color and a set of source nodes (instead of a single node) does not change the analysis. The key observation in the proof of Alon, Yuster and Zwick is that the number of labels in each node after i iterations is bounded by $\binom{|\mathcal{C}|}{i}$.

Theorem 5.18. *Algorithm 9 solves the Colorful Path Problem in* $O(|\mathcal{C}| \cdot 2^{|\mathcal{C}|} \cdot |E|)$.

Remark 5.19 (Additional heuristic)**.** *Due to the exponential run time of the algorithm we provide a heuristic modification for Algorithm 9. In fact, we introduce an upper bound to the size of the set* Q_{new}*. The algorithm is modified such that it returns true whenever this bound is reached.*

Now we are able to formulate our deadlock prevention algorithm (Algorithm 10). Given a sequence of (static) paths the algorithm computes a deadlock-free reservation schedule by iteratively inserting these paths, which is done by Algorithm 11 (INSERT-ROUTE).

The basic concept of INSERT-ROUTE is to maintain a deadlock detec-

Algorithm 10: DEADLOCK-PREVENTION

Data: Graph $G = (V, E)$, sequence of static paths $\mathcal{P} = P_1, \ldots, P_k$.
Result: Deadlock-free reservation schedule $S(\mathcal{P})$.
begin
  foreach path P_i **do**
    INSERT-ROUTE (Algorithm 11);
end

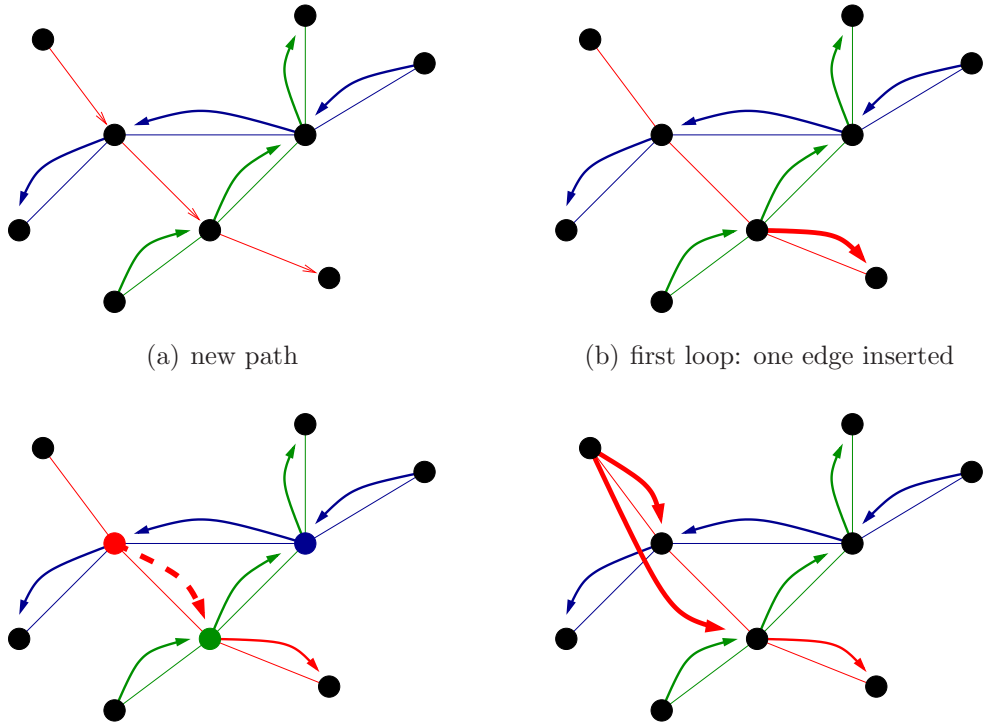
Algorithm 11: INSERT-ROUTE

Data: Graph $G = (V, E)$, deadlock-free reservation schedule $S(\mathcal{P})$ and corresponding deadlock detection graph $G_D = (V_D, E_D)$, static path $P = (e_1, \ldots, e_n)$.
Result: Deadlock-free reservation schedule $S(\mathcal{P} \cup P)$ and the corresponding deadlock detection graph $G_D = (V_D, E_D)$.
begin
  $j = n$;
  $i = n - 1$;
  while $i \geq 1$ **do**
CP **if** *there is no colorful* e_ℓ*-*e_i*-path without color* i *for any* $e_\ell \in \bigcup_{k=i+1}^{j} \mathit{confl}(e_k)$ *in* G_D **then**
RS $R_P(e_i) = \bigcup_{k=i+1}^{j} e_k$;
DG $\forall e_\ell \in \bigcup_{k=i+1}^{j} \mathit{confl}(e_k)$ add $((e_i, e_\ell), c_P)$ to E_D;
      $j = i$;
    $i = i - 1$;
end

tion graph that contains no colorful cycle (see line DG of the algorithm). This is guaranteed by executing Algorithm 9 in line CP. Simultaneously, a corresponding reservation schedule is constructed (line RS). Note that we add reservations to the reservation schedule (edges to the deadlock detection graph) block-wise.

Remark 5.20 (Block-wise insertion of requested edges)**.** *Algorithm 11 inserts requested edges block-wise to the reservation schedule, cf. Figure 5.5 and Figure 5.6. More precisely, we only add a request from a certain edge e if all edges that preceed e on the considered path can also be reserved or are already reserved, cf. line CP of the algorithm.*

By Theorem 5.16 we know that such a reservation schedule is deadlock-



(a) new path (b) first loop: one edge inserted

(c) second loop: cycle detected (d) third loop: two edges inserted

Figure 5.5: Insertion of a new route (red) by INSERT-ROUTE. Figure (a) illustrates the initial situation while the Figures (b), (c) and (d) show the three loops of the algorithm. Note that we do not take conflicting edges into account in this example.

free if it corresponds to a deadlock detection graph without colorful cycles. Therefore, we only have to argue that the deadlock detection graph is constructed correctly (corresponds to the reservation schedule) and that the reservation schedule is valid to obtain Theorem 5.21.

Theorem 5.21. *The reservation schedule constructed by Algorithm 10 is valid and deadlock-free.*

Proof. Firstly, we observe that the constructed reservation schedule is valid. Due to Assumption 5.12, the first edge of the considered path does not lie on a colorful cycle. Therefore, each edge can be requested at least from that edge. Moreover, by construction of the algorithm (the invariant $i < j$ holds in each iteration), each edge is requested from a preceding edge.

To obtain that the constructed reservation schedule is deadlock-free it is sufficient to prove that the deadlock detection graph is constructed correctly, i.e., to show that it corresponds to the computed reservation schedule, since

we can apply Theorem 5.16 in this case. To see this, we observe that edges are requested in blocks (see Remark 5.20), that is, whenever we insert edges of a path $P = (e_1, \ldots, e_n)$ to a set $R_P(e_i)$ it is guaranteed that no edge e_k with $k > i$ is already (or will be) requested from an edge e_ℓ with $\ell < i$ during the algorithm. Therefore, by termination of the algorithm, for each edge e_i of the given path it holds that

$$R_P(e_i) = \emptyset \quad \vee \quad O_P(e_i) = \{e_i\}. \tag{5.5}$$

Thus, whenever there is an edge g with

$$f \in \mathit{confl}(R_P(g)) \quad \text{and} \quad e \in O_{P'}(g)$$

it holds that $f \in \mathit{confl}(R_P(e))$. Hence, by Definition 5.14, it is sufficient to insert those reservations to the deadlock detection graph that represent a requests of edges in the corresponding reservation schedule (line RS of Algorithm 11), cf. Figure 5.6. This is done in line DG of the algorithm.

We conclude that the deadlock detection graph constructed during the algorithm corresponds to the determined reservation schedule. Since the deadlock detection graph has no colorful cycles by construction (line CP of the algorithm), this completes the proof. □

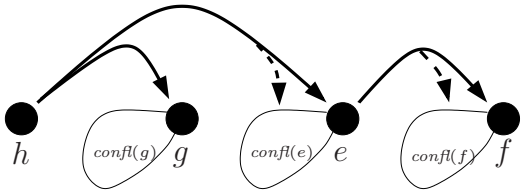
For the analysis of the run time we observe that Algorithm 9 is called at most $|P|$ times in Algorithm 11, where $|P|$ denotes the number of edges on a certain path P.

Corollary 5.22. *Algorithm 11 computes a valid and deadlock-free reservation schedule that contains the new path P in $O(|\mathcal{C}| \cdot 2^{|\mathcal{C}|} \cdot |E| \cdot |P|)$, where $|P|$ denotes the number of edges on P.*

5.4 Computational Results

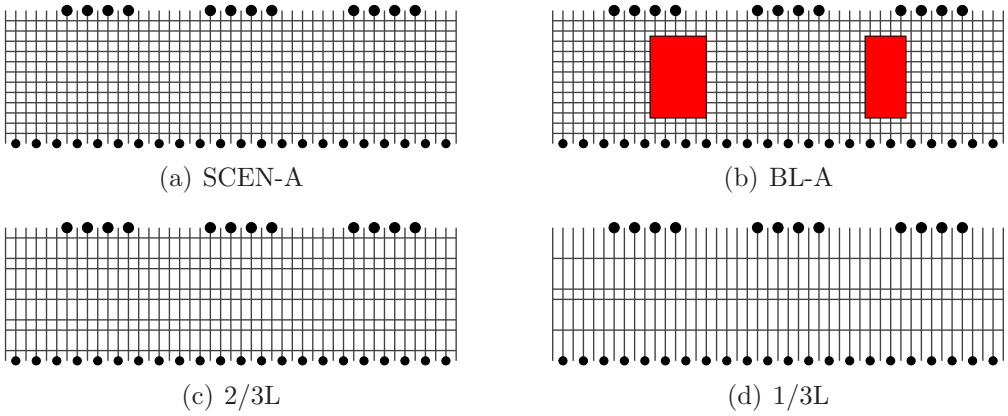
For the evaluation of the presented routing approach, namely algorithm STAT-ROUTE (Algorithm 6), we consider scenario SCEN-A presented in Section 3.3.1. Recall that 72 vehicles serve request generated by a simulation environment (see Section 2.3) between 22 delivery points and 12 pick-up points in a grid-like graph in this scenario. Additionally, we again conduct six-hour simulation runs (about 6000 requests) on an AMD Athlon 64 Dual Core 2.2 GHz with 4 GB RAM.

Based on scenario SCEN-A we create additional scenarios by excluding parts of the underlying graph, cf. Figure 5.7. This is done to measure the performance under different traffic densities. Besides scenario BL-A which



edge	requested edges
h	e, g
g	$\emptyset$
e	f
f	$\emptyset$

Figure 5.6: Reservation schedule (deadlock detection graph) constructed by Algorithm 10 for path $P = (h, g, e, f)$. In contrast to the schedule illustrated in Fig. 5.4 only edges that directly result from the set of requested edges are inserted in the deadlock detection graph. This is due to the block-wise insertion of the edges (see Remark 5.20 and Theorem 5.21).



(a) SCEN-A (b) BL-A

(c) 2/3L (d) 1/3L

Figure 5.7: Illustration of the scenarios investigated for the evaluation of the static routing approach. Besides the plain scenario SCEN-A we consider scenario BL-A with two blocked areas and two scenarios with reduced number of horizontal lanes in the grid-like graph, namely scenario 2/3L and scenario 1/3L.

has already been introduced in Section 3.3.4 we consider two scenarios that result from SCEN-A by reducing the number of horizontal lanes. Note that scaling down the graph is the only possibility for increasing the traffic density significantly in our simulation environment since the number of vehicles in the scenarios is bounded (details, again, are confidential).

BL-A: We consider two blocked areas that cover essential parts of the grid such that there are only one third of the lanes left in these parts (see Figure 5.7(b)).

2/3L: SCEN-A with two thirds of the horizontal lanes (see Figure 5.7(c)).

1/3L: SCEN-A with one third of the horizontal lanes (see Figure 5.7(d)).

As in Section 3.3 we evaluate the *average duration* of the determined paths—recall that we interpret the reservation procedure, cf. Section 5.1.1, as the construction of a dynamic path at the time of the execution of a given static path—and the *computation times*. Additionally, in order to analyze the load balancing approach presented in Section 5.2, we also investigate the *(static) length* of the computed paths and the *load* on the edges of the graph. Moreover, we evaluate the *number and the length of the cycles* detected by the deadlock prevention algorithm (Algorithm 10).

Due to the results of the first evaluations described in Remark 5.23 we restrict the route computation. In fact, we apply directions to the horizontal lanes alternatingly, see Figure 5.8, and compute the static path in that modified graph. Note that we introduced a similar restriction in Section 4.2.3 in order to analyze the dynamic routing approach under these conditions.

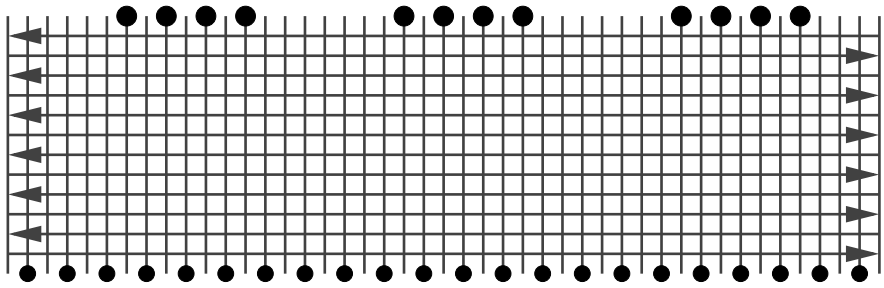


Figure 5.8: SCEN-A with directed horizontal lanes.

Remark 5.23 (Evaluations in the undirected grid-like graph)**.** *It turned out that the static routing algorithm is not competitive in the considered (undirected) grid-like graph. In fact, the systems almost stalls and therefore we omit detailed evaluations.*

The reason for the bad performance is the frequent appearance of so-called head-to-head conflicts if two vehicles want to pass a portion of the graph in opposite directions. Therefore at least one of these vehicles has to reserve the whole area at once. This leads to enormous claims. Moreover, the deadlock detection becomes computationally expensive.

The evaluation is divided into two parts: Firstly, we analyze the performance under variation of the stretch factor B, cf. Section 5.2, in SCEN-A. Afterwards we consider the impact of the traffic intensity on the performance using the three additional scenarios. In particular, we compare the results with those of the dynamic routing algorithm DYN-ROUTE (Algorithm 1).

5.4.1 Variation of the Stretch Factor

In Section 5.2 we introduced Algorithm 7 (BAL-BOUND) for the route computation in our static routing algorithm STAT-ROUTE (Algorithm 6). The cost function of the algorithm depends on the given length constraint to the determined paths, the stretch factor B. Table 5.1 illustrates the evaluation of the performance under variation of that value.

B	average duration (in sec.)	average path length (in m)	max. load	cycle length avg.	cycle length max.	# cycles per request	comp. time avg. (in sec.)	comp. time max. (in sec.)
1.0	209.46	298.43	29	3.79	12	1.27	0.11	1.24
1.1	170.77	299.81	26	2.98	9	0.35	0.09	0.82
1.2	169.01	300.46	25	2.85	9	0.33	0.09	0.70
1.3	172.26	300.99	26	2.84	9	0.40	0.09	0.59
1.4	169.89	304.65	26	2.93	9	0.35	0.10	0.72

Table 5.1: Evaluation of the static routing approach with respect to different stretch factors B.

It turns out that the results of the experiments with stretch factor greater than 1.0 are very similar. They show only minor differences. In contrast, simply computing a static shortest path for each request ($B = 1.0$, no load balancing) leads to significantly different results. While the static path length is, of course, shorter than in the other cases, the maximum load on the edges is higher. This leads to a more complicated deadlock prevention, namely there are much more detected cycles which generates larger claims. Since each detected cycle makes an earlier reservation necessary (cf. Remark 5.10) it is not surprising that the average duration is larger in this case. Moreover, the detected cycles are longer than those in the more balanced cases which results in larger computation times. Thus, we conclude that load balancing in the route computation of our dynamic routing algorithm STAT-ROUTE makes sense, but, at least in the considered grid-like graph, the stretch factor does not play an important role.

Since the least average duration is achieved with stretch factor $B = 1.2$, we choose this setting for the evaluations in Section 5.4.2.

5.4.2 Comparison with the Dynamic Routing Algorithm

Now, we focus on the question of which routing approach—the static one or the dynamic one (Algorithm 1) introduced in Chapter 3—performs better with respect to the investigated objective: the average duration.

As illustrated in Table 5.2 and Figure 5.9, this question cannot be answered generally. It highly depends on the traffic density. While the static approach shows a slightly lower average duration than the dynamic one in the scenarios with a comparatively low traffic volume, namely the plain scenario SCEN-A and scenario 2/3L, it has problems in scenarios with high traffic volume. Especially in the most narrow scenario 1/3L the static approach performs very badly while the average duration measured for the dynamic approach does not increase that much compared with the other scenarios. Moreover, regarding the static approach, the computation times increase with the traffic density which is not the case if the dynamic approach is used.

	static approach			dynamic approach		
	average duration (in sec.)	computation time avg. (in sec.)	computation time max. (in sec.)	average duration (in sec.)	computation time avg. (in sec.)	computation time max. (in sec.)
SCEN-A	169.01	0.09	0.70	182.26	0.08	0.83
2/3L	186.91	0.12	3.86	190.70	0.08	0.98
BL-A	255.79	0.17	1.84	212.31	0.08	1.14
1/3L	693.14	0.31	4.48	259.72	0.08	1.24

Table 5.2: Comparison of the static routing approach with the dynamic routing approach with respect to the average duration and the computation times.

The reason for the bad performance of the static routing approach in scenarios with high traffic volume becomes clear if we regard the evaluation of the deadlock detection shown in Table 5.3.

First of all, note that we have to use the heuristic introduced in Remark 5.19 in all instances except the plain scenario SCEN-A to provide suitable computation times. The upper bound is set to 500. We also tried to evaluate the instances without using the heuristic (or with larger upper bounds), but the performance was even worse since the system almost stalls from time to time due to the enormous computation times (more than 60 seconds). The number of cases in which the upper bound is reached increases

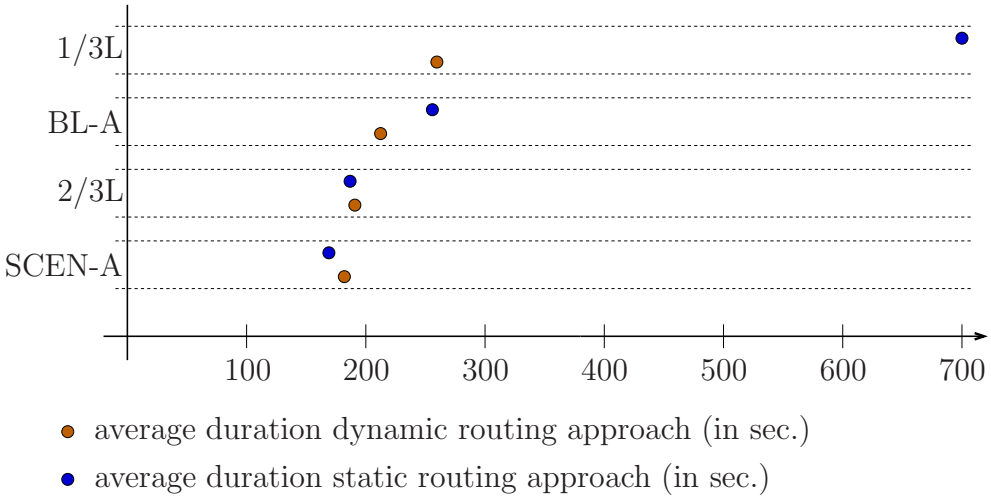


Figure 5.9: Illustration of the performance of the static routing approach in comparison to the dynamic routing approach with respect to the average duration.

with the traffic density and becomes immense in scenario 1/3L. Moreover, we observe the same for the number of detected cycles. The length of the claims (see Remark 5.10) increases with the number of found cycles and the number of canceled searches (heuristic), respectively. It is not surprising that this leads to a loss of performance, cf. Table 5.2 and Figure 5.9.

	cycle length		# cycles	# heuristic used
	average	maximum	per request	per request
SCEN-A	2.85	9	0.33	0.00
2/3L	3.15	10	0.78	0.23
BL-A	4.72	13	1.24	0.33
1/3L	4.25	14	1.35	8.59

Table 5.3: Evaluation of the deadlock detection algorithm with respect to different traffic densities. We consider the length (average and maximum) and the number of detected cycles as well as the number of cases in which the upper bound in the heuristic described in Remark 5.19 is reached (# heuristic used).

But why does the static approach perform better in scenarios with low traffic density? The cause for this (perhaps surprising) result is the greedy reservation procedure used in this case (see Section 5.1.1). The next portion of the route is reserved as soon as possible disregarding that this may interfere other vehicles, see Figure 5.10. In contrast, the dynamic routing algorithm

does not make use of such gaps since the reservations are made before the vehicles start traveling and it is forbidden to use time windows that cannot be left before the next vehicle is scheduled on that edge.

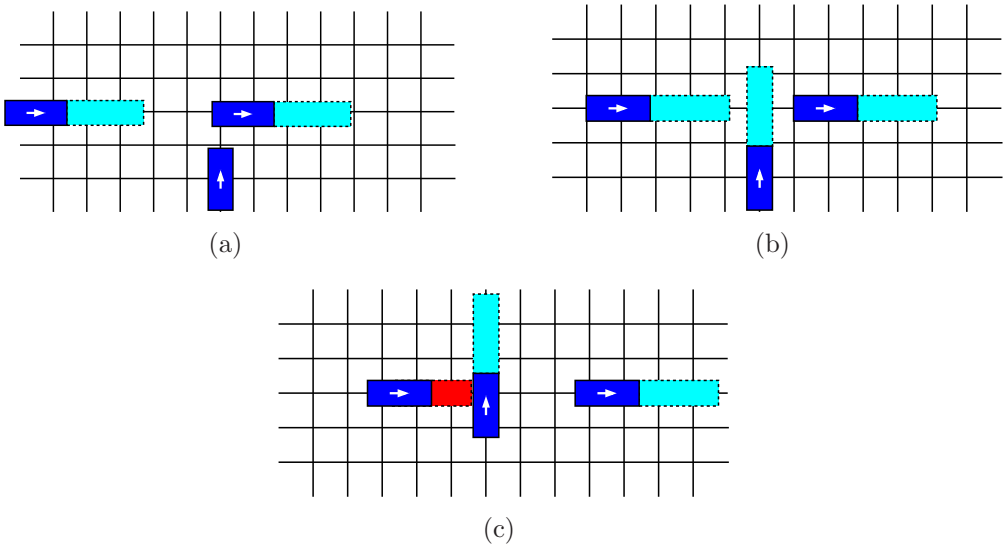


Figure 5.10: Illustration of the greedy reservation procedure used by the static routing approach.

The only way to change the behavior of the dynamic approach in this respect is to permit rerouting of other vehicles also in such cases, cf. Section 3.2. We implemented an approach such as this using ideas similar to those described in Section 3.2.2 where we take priorities into consideration. More precisely, instead of verifying the priority of a vehicle/request that should be ignored we add penalty costs to the label value whenever another vehicle is ignored. We varied the amount of the penalty, but the evaluation of that strategy showed that the benefit of the increased flexibility and the loss that is caused by the required rerouting cancel each other out. In fact, this approach all in all leads to a slight loss of performance with respect to the average duration.

To conclude the evaluation, we remark that the static routing approach is good as long as there are only a few potential deadlocks that have to be avoided since the greedy reservation procedure is of value in this case; but if the claims become larger caused by a more complicated deadlock prevention, it reaches its limits. Therefore, the performance of the static approach is only competitive in scenarios with comparatively small traffic density. However, in such instances it has a slight advantage indeed.

5.5 Conclusions

In this chapter we introduced a so-called static routing algorithm STAT-ROUTE (Algorithm 6). In contrast to the dynamic routing algorithm DYN-ROUTE presented in Chapter 3 (Algorithm 1) time dependences are not taken into account during the route computation. In fact, a standard (static) shortest path with respect to a certain cost function is determined. Therefore, an additional collision avoidance is needed. This is done by reserving areas (claims) in front of the vehicle during the execution of the computed route, cf. Section 5.1.1. Note that the dynamic routing algorithm is already avoiding collisions at the time of the route computation.

Each of these two parts of the vehicle guidance, the route computation and the reservation procedure, is linked to a particular problem of the static routing approach. On the one hand, the computation of a static shortest path may cause congestion or detours that could have been avoided if the time-dependent behavior of the vehicles has been taken into account. On the other hand, the reservation procedure involves the risk of deadlocks.

We cope with both problems via a two-stage routing approach: In the first step we consider a load balancing approach (see Section 5.2). In fact, we compute a static shortest with respect to a particular cost function that depends on the transit time and the load on the edges (the number of vehicles already routed over an edge) for each request. This strategy turns out to by optimal with respect to a measure of quality that takes the length of the routes and the distribution of load on the edges into account; the stretch factor restricted competitive ratio. In a second step we provide a deadlock-free reservation schedule based on the computed route (see Section 5.3). The key in this step is the avoidance of specific cycles in a so-called deadlock detection graph that corresponds to that schedule.

The evaluation of this approach based on the introduced simulation of the HHLA Container Terminal Altenwerder shows that the presented algorithm is basically suitable for real-time use. Actually, as far as we know, this is the first static routing approach that prevents deadlocks before the computed route is executed and is able to cope with large-scale vehicle fleets. Our approach, however, reaches its limits if we increase the traffic density by reducing the number of available lanes/edges of the considered grid-like graph. In contrast, in scenarios with comparatively low traffic volume, our static routing algorithm is even superior to the dynamic routing algorithm, cf. Section 5.4.2.

Thus, the decision, which routing approach should be preferred highly depends on the expected traffic density. Since this might be difficult to anticipate, one can also think of a hybrid approach, that is, one switches between both approaches depending on the current traffic situation. A pre-

condition for such a switch, however, is a complete stop of all vehicles on position where they do not block each other. One possibility for guaranteeing this is to finish all pending requests before switching. Besides the performance loss caused by the switching procedure there remains the problem of evaluating the situation concerning the traffic density at any time. This is difficult since, especially if perturbations occur, the traffic density might change considerably at a moment's notice.

In addition to the traffic density the structure of the underlying graph seems to play an important role. In particular, the appearance of head-to-head conflicts in undirected graphs leads to problems of the static routing algorithm, cf. Remark 5.23, while the dynamic routing algorithm benefits from the flexibility in a bidirectional layout (Remark 4.18).

Beyond these observations concerning the conducted experiments, there are further advantages and disadvantages of the static routing approach that are caused by the absence of time dependence during the route computation. On the one hand, one does not have to take care for deviations in time during the execution of the routes (late or early vehicles), but, on the other hand, there is a lack of information and flexibility for a higher-level management system that assigns the requests to the vehicles (cf. Section 5.1.2).

To conclude the comparison of the dynamic and the static routing approach, we suggest to use the latter in applications where a low traffic density is guaranteed in general or at least for a long time period. In contrast, the dynamic approach is better suited for scenarios with comparatively high traffic density. Moreover, a higher-level management systems benefits from the exactness in time provided by this approach independent of the traffic situation.

Bibliography

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, Journal of the ACM **42**, no. 4 (1995), pp. 844–856. 103, 104, 105, 106

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, *On-line routing of virtual circuits with applications to load balancing and machine scheduling*, Journal of the ACM **44**, no. 3 (1997), pp. 486–504. 90, 91, 96

[3] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani, *On-line admission control and circuit routing for high performance computing and communication*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1995, pp. 412–423. 1

[4] Y. Azar, J. Noar, and R. Rom, *The competitivness of on-line assignment*, in Proceedings of the 3rd ACM-SIAM Symposium on Theory of Computing, 1992, pp. 203–210. 91

[5] M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, *Handbooks in Operations Research and Management Science: Network Routing*, Elsevier, 1995. 1

[6] Y. Bartal, A. Fiat, and S. Leonardi, *Lower bounds for on-line graph problems with application to on-line circuit and optical routing*, in Proceedings of the 28th ACM Symposium on Theory of Computing, 1995. 1

[7] J. E. Beasley and N. Christofides, *An algorithm for the resource constrained shortest path problem*, Networks **19** (1989), pp. 379–394. 20

[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998. 1, 90

[9] R. E. Burkard, K. Feldbacher, B. Klinz, and G. J. Woeginger, *Minimum-cost strong network orientation problems: Classification, complexity, and algorithms*, Networks **33** (1999), pp. 57–70. 5

[10] H. B. Cho, T. K. Kumaran, and R. A. Wysk, *Graph theoretic deadlock detection and resolution for flexible manufacturing systems*, IEEE Transactions on Robotics and Automation **11**, no. 3 (1995), pp. 413–421. 4, 99

[11] M. Desrochers, J. Desrosiers, M. Sauve, and F. Soumis, *Methods for routing with time windows*, European Journal of Operational Research **23** (1986), pp. 236–245. 3, 18, 20

[12] M. DESROCHERS AND F. SOUMIS, *A generalized permanent labelling algorithm for the shortest path problem with time windows*, INFOR **26** (1988), pp. 191–212. 3, 18, 20

[13] M. DESROCHERS AND F. SOUMIS, *A reoptimization algorithm for the shortest path problem with time windows*, European Journal of Operational Research **35** (1988), pp. 242–254. 3, 18, 20

[14] L. R. FORD AND D. R. FULKERSON, *Constructing maximal dynamic flows from static flows*, Operations Research **6** (1958), pp. 419–433. 3, 7, 17

[15] L. R. FORD AND D. R. FULKERSON, *Flows in networks* (1962). 3, 7, 17

[16] H. N. GABOW, S. N. MAHESHWARI, AND L. OSTERWEIL, *On two problems in the generation of program test paths*, IEEE Transactions on Software Engineering **SE-2** (1976), pp. 227–231. 105

[17] J. GAO AND L. ZHANG, *Tradeoffs between stretch factor and load balancing ratio in routing on growth restricted graphs*, in Proceedings of the 23th annual ACM Symposium on Principles of Distributed Computing, 2004, pp. 189–196. 91

[18] E. GAWRILOW, M. KLIMM, R. H. MÖHRING, AND B. STENZEL, *Conflict-free vehicle routing: Load balancing and deadlock prevention*, Matheon Preprint 497, TU Berlin, 2008. 87

[19] E. GAWRILOW, E. KÖHLER, R. H. MÖHRING, AND B. STENZEL, *Conflict-free real-time AGV routing*, in Proceedings of the International Conference on Operations Research 2004, H. Fleuren, D. den Hertog, and P. Kort, eds., Springer, 2005, pp. 18–24. 17, 22

[20] 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, W. Jäger and H.-J. Krebs, eds., Springer, 2008, pp. 165–178. 17, 22

[21] W. H. GUAN AND K. M. R. L. MOORTHY, *Deadlock prediction and avoidance in an AGV system*. SMA Thesis, University of Singapore, 2000. 4, 87, 99

[22] P. HART, N. NILSSON, AND B. RAPHAEL, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on Systems, Science and Cybernetics **4** (1968), pp. 100–107. 23

[23] ILOG SA, FRANCE, *ILOG CPLEX 10.1 Reference Manual*, `http://www.ilog.com/products/cplex`, 10.1 ed., 2007. 72

[24] A. KARLIN, M. MANASSE, L. RUDOLPH, AND D. SLEATOR, *Competitive snoopy paging*, Algorithmica **3** (1988), pp. 70–119. 6

[25] C. W. Kim and J. M. A. Tanchoco, *Conflict-free shortest-time bidirectional AGV routing*, International Journal of Production Research **29**, no. 12 (1991), pp. 2377–2391. 18

[26] K. H. Kim, S. M. Jeon, and K. R. Ryu, *Deadlock prevention for automated guided vehicles in automated container terminals*, OR Spectrum **28**, no. 4 (2006), pp. 659–679. 4, 99

[27] E. Köhler, R. H. Möhring, and M. Skutella, *Traffic networks and flows over time*, in Special Volume Dedicated to the DFG Research Center Mathematics for Key Technologies Berlin, J. Kramer, ed., Berliner Mathematische Gesellschaft, 2002, pp. 49–70. 3, 7

[28] N. Krishnamurthy, R. Batta, and M. Karwan, *Developing conflict-free routes for automated guided vehicles*, Operations Research **41**, no. 6 (1993), pp. 1077–1090. 1, 9

[29] M. Kühne, *Algorithmen für Dynamische Disjunkte Wege*, diploma thesis, TU Berlin, 2008. In German. 55

[30] C. C. Lee and J. T. Lin, *Deadlock prediction and avoidance based on petri nets for zone control*, International Journal of Production Research **33**, no. 12 (1995), pp. 3239–3265. 99

[31] D. Lubell, *A short proof of sperner's theorem*, Journal of Combinatorial Theory **1** (1966), p. 299. 38

[32] N. G. F. Sancho, *Shortest path problems with time windows on nodes and arcs*, Journal of mathematical analysis and applications **186** (1994), pp. 643–648. 3, 18, 20

[33] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer, 2003. 1, 8

[34] R. Sedgewick and J. S. Vitter, *Shortest paths in euclidian graphs*, Algorithmica **1** (1986), pp. 31–48. 23

[35] S. Seiden, J. Sgall, and G. J. Woeginger, *Semi-online scheduling with decreasing job sizes*, Tech. Report KAM-DIMATIA Series 98-410, 1998. 92

[36] D. Sleator and R. Tarjan, *Amortized effeciency of list update and paging rules*, Communications of the ACM **28** (1985), pp. 202–208. 6

[37] I. Spenke, *Complexity and approximation of static k-splittable flows and dynamic grid flows*. PhD Thesis, TU Berlin, 2006. 1, 3, 9, 10, 13, 55, 68, 69, 70, 74, 81, 84, 85

[38] E. Sperner, *A theorem about subsets of a finite set*, Mathematische Zeitschrift **27** (1928), pp. 544–548. In German. 38

[39] P. TOTH AND D. VIGO, *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, 2002. 1

[40] I. F. A. VIS, *Survey of research in the design and control of automated guided vehicle systems*, European Journal of Operational Research **170**, no. 3 (2006), pp. 677–709. 2

[41] N. Q. WU AND M. C. ZHOU, *Resource-oriented petri nets for deadlock avoidance in automated manufacturing*, Proceedings of the 4th IEEE International Conference on Robotics and Automation (2000), pp. 3377–3382. 99

[42] M. S. YEH AND W. C. YEH, *Deadlock prediction and avoidance for zone-control AGVs*, International Journal of Production Research **36**, no. 10 (1998), pp. 2879–2889. 4, 99