Dynamic Routing of Automated Guided Vehicles in Real-time

by

Ewgenij Gawrilow Ekkehard Köhler
Rolf H. Möhring Björn Stenzel

No. 039/2007

Dynamic Routing of Automated Guided Vehicles in Real-time

Ewgenij Gawrilow Ekkehard Köhler Rolf H. Möhring
Björn Stenzel

October 10, 2007

Abstract. Automated Guided Vehicles (AGVs) are state-of-the-art technology for optimizing large scale production systems and are used in a wide range of application areas. A standard task in this context is to find efficient routing schemes, i.e., algorithms that route these vehicles through the particular environment. The productivity of the AGVs is highly dependent on the used routing scheme.

In this work we study a particular routing algorithm for AGVs in an automated logistic system. For the evaluation of our algorithm we focus on Container Terminal Altenwerder (CTA) at Hamburg Harbor. However, our model is appropriate for an arbitrary graph. The key feature of this algorithm is that it avoids collisions, deadlocks and livelocks already at the time of route computation (conflict-free routing), whereas standard approaches deal with these problems only at the execution time of the routes. In addition, the algorithm considers physical properties of the AGVs and certain safety aspects implied by the particular application.

1 Introduction

Automation of large scale logistic systems is an important method for improving productivity. Often, in such automated logistic systems Automated Guided Vehicles (AGVs) are used for transportation tasks. Especially, so called free-ranging AGVs are more and more used since they add a high flexibility to the system. The control of these AGVs is the key to an efficient transportation system that aims at maximizing its throughput.

In this work we focus on the problem of routing AGVs. This means we study how to compute good routes on the one hand and how to avoid collisions on the other hand. Note that dispatching of AGVs, i.e., the assignment of transportation tasks to AGVs, is not part of the routing and therefore not considered in this paper.

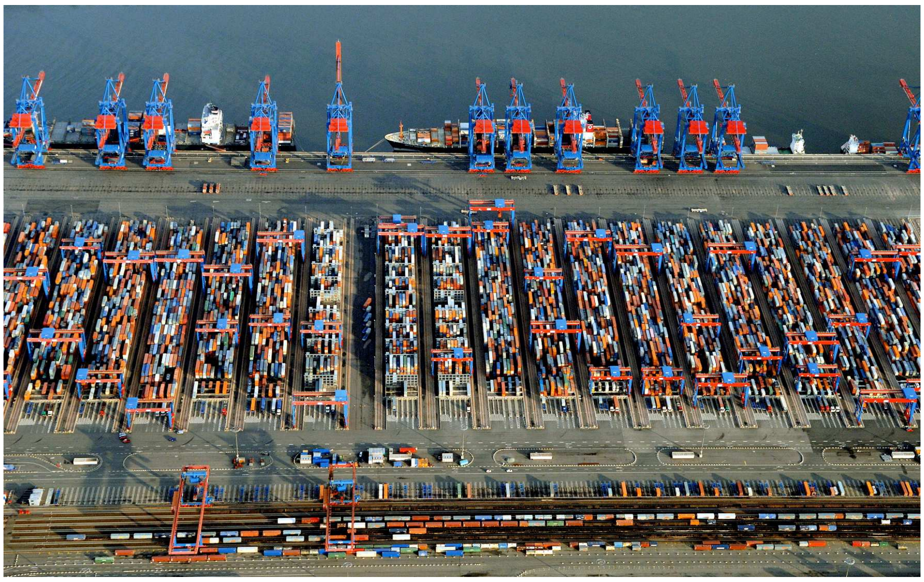

Figure 1: The HHLA Container Terminal Altenwerder (CTA). ©HHLA

Our application is the Container Terminal Altenwerder (see Fig. 1), which is operated by our industrial partner, the Hamburger Hafen und Logistik AG (HHLA).

We represent the AGV network by a particular grid-like graph that consists of roughly 10,000 arcs. and models the underlying street network of a traffic system consisting of a fleet of AGVs. The task of the AGVs is to transport containers between large container bridges for loading and unloading ships and a number of container storage areas. 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.

Previous Work. First ideas for free-ranging AGV systems were introduced by Broadbent et al. [2]. Since then, several papers concerning this topic have been published [18]. In this paper we focus on routing approaches in the case where dispatching of AGVs is already made.

In so-called *offline* approaches all requests (transportation tasks) are known right from the beginning. Krishnamatury, Batta and Karwan [11] as well as Qui and Hsu [13] discuss the AGV routing problem in this case. While Krishnamatury, Batta and Karwan present a heuristic solution for general graphs (where this routing problem is $\mathcal{NP}$-hard [16]), Qui and Hsu consider a very simple graph and present a polynomial time algorithm.

In contrast, online approaches assume that requests appear sequentially. The *local* approach by Taghaboni-Dutta and Tanchoco [17] is such an online method. Here, a decentralized algorithm decides about the routes, based only on local information. In particular, it does not determine the whole path for an AGV, but iteratively computes sub-paths from checkpoint to checkpoint.

The *static* approach uses the full information about the already routed AGVs. Algorithms that are based on these approaches usually compute geographically shortest paths with optional additional penalty costs on congested arcs from the source to the destination of the current request [12]. This static formulation needs an additional collision avoidance system to make the routes collision-free, since time dependences are not represented in that model.

Our Contribution. In this work we study a dynamic online AGV routing model for an arbitrary graph. This approach is motivated by dynamic flow theory (see [6, 7, 10]) and several papers on the Shortest Path Problem with Time-Windows [3, 4, 5, 14]. The main advantage of our model and algorithm over the known online methods is that the time-dependent behavior of AGVs is fully modeled, such that both conflicts and deadlock situations can be prevented already at the time of route computation. The newly designed model is not only very accurate in the mapping of properties of the actual application but, as we show in our computational experiments, it is also well suited for being used in a real-world production system.

The paper is organized as follows. In Section 2 we describe how we model the AGV network. In Section 3 we introduce our algorithm and show that it runs in polynomial time. Section 4 explains how subtle technical characteristics of the particular application can be represented in our model to get it ready for being used in practice. The computational results are presented in Section 5.

2 The Model

We model the automated transportation system by a directed graph G representing the feasible lanes of the system. These lanes are given by certain transponder positions. In the application, this graph has about 10,000 arcs. Initially, we assume that every arc a has a fixed, constant transit time $\tau(a)$.

Transportation tasks are consecutively arriving over time and are modeled by a sequence $\sigma = r_1, \ldots, r_n$ of requests. Each request $r_j = (s_j, t_j, \theta_j)$ consists of a start node s_j, an end node t_j, and a desired starting time θ_j. The aim is to minimize the overall transit times, that is the sum of transit times over all requests. We approach this goal by iteratively computing a shortest path for each request, which is a natural method in this online setting.

The physical properties of the AGVs demand for a variety of special features of the model. Although each route of an AGV can be represented in the given graph, not every route in this graph can in fact be conducted by an AGV. The reason for this difficulty is the complicated turning behavior, which makes it necessary to start turning the wheel already long before the particular intersection is reached. As a consequence, an AGV needs a sufficiently long straight route segment between two consecutive curves. To cope with this rather complicated turning behavior we introduce in a preprocessing step a set of *artificial arcs* to the network, each representing a possible turn (see Figure 2). In addition, at each node of the graph we introduce turning rules defining which out-going arcs of a node can be used from a particular in-going arc. As a result we get a much larger network (about 45,000 arcs) that captures all possible movements of an AGV — each feasible route in this network can be executed by an AGV.

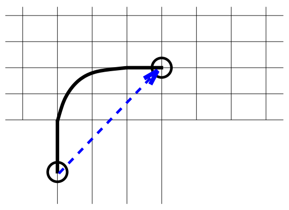

Figure 2: The figure illustrates an artifical arc (blue dotted arrow) that models a curve. This is done for all permitted curves.

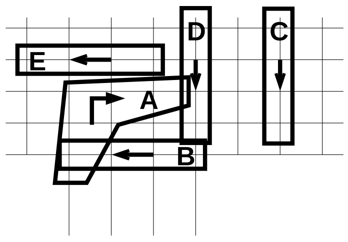

Figure 3: The figure illustrates the polygons that are claimed by an AGV that moves in the indicated direction. Polygon A, B and D pairwise intersect each other while polygons C and E do not intersect any of the others, respectively.

Another complicating property is the size of the AGVs compared to the rather closely meshed network of lanes. If an AGV traverses or stands on an arc a, it affects a much larger portion of the network than only arc a, which is then blocked for other AGVs (see Figure 3). In Section 2.2, we describe how we take this into account in our conflict-free (dynamic) approach.

2.1 Static routing and its drawbacks

A standard approach for routing of AGVs in an online setting is the so-called static routing. In this case one only computes static routes in the network, ignoring their time dependent nature. More precisely, one computes a standard shortest path, e.g., using Dijkstra's algorithm, with respect to arc costs consisting of the transit times τ_a plus a load dependent penalty cost which is a function of the number of routes that are already using this arc (see [12]). The computed routes are, of course, not collision-free. Hence, one needs an additional conflict avoidance system that, at execution time of the routes, guarantees that there are no collisions. One way to do this is to iteratively allocate to an AGV the next part of its route (the "claim") and block it for all other AGVs ("claiming").

The advantage of the static approach is clear. It is easy to implement and allows very fast route computation. However, various drawbacks are caused by the collision avoidance at execution time. In particular, the claiming rules can cause deadlocks and have a deteriorating effect on the system performance.

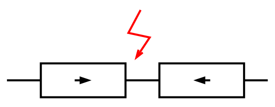
Figure 4: Simplified deadlock situation. Both AGVs are trying to occupy the same arc of the network, thereby blocking each other.

Deadlocks (see Fig. 4) appear if a group of AGVs wish to claim an area which is already occupied by another AGV in this group. None of them is able to continue its route and thus the system is blocked. Algorithmic solutions for that problem are only suitable for a very small number of AGVs [9, 12].

In addition to deadlocks, a variety of other drawbacks like detours and high congestion occur in the static setting; again, since time-dependent behavior is not considered. This results in traveling times that can be far away from the shortest possible traveling time. Moreover, actual arrival times of the AGVs at their destinations are completely random and cannot be predicted at the time of route computation. This is a major drawback for other planning steps in the logistic chain that depend on the knowledge of these arrival times.

2.2 Dynamic routing of AGVs

In order to avoid the problems of the simple model given in Section 2.1, we follow a completely different approach that computes shortest (w.r.t. traveling time) and conflict-free routes simultaneously.

There are two key ingredients which must be considered in our approach. On the one hand, one has to deal with the physical dimensions of the AGVs because they usually have to claim several arcs in the directed graph at the same time. On the other hand, the approach has to be time-dependent (dynamic).

Every arc can be seen as a set of time intervals, each representing a different AGV that is routed over this arc or, at least, blocks this arc during some time interval. Note that these intervals have to be mutually disjoint since an overlap would mean that the corresponding AGVs collide on this arc at the time of the overlap. In fact, in our algorithm, we will not maintain the set of intervals in which an arc is blocked, but the complementary set of free time-intervals (*time-windows*).

Maintaining these sets of intervals may be seen as a compact representation of the standard time-expanded graph, in which there is a copy of each vertex/arc for each point in time (with respect to some time discretization). In contrast, the set of time-windows of an arc a only models those times, in which there actually is no AGV on a. Similar compact representations of a time-expanded graph by time intervals have been studied before, see e.g. [3, 4, 5, 14] and Section 3.2.

For dealing with the physical dimensions of the AGVs we use polygons $P(a)$ for each arc a, which describe the blocked area when an AGV (the center of an AGV) is located on arc a (Fig. 3). Thus, it is prohibited to use two arcs at the same time if the corresponding polygons intersect. For each arc a, this leads to a set *confl*(a) of so called *geographically dependent arcs* which must not be used at the same time. If an AGV travels along an arc a during the interval $[\theta_1, \theta_2]$, all geographically dependent arcs are blocked from θ_1 to θ_2. Note that in this approach there is no need to model traveling on nodes since each edges contains its end nodes.

After routing a request one has to readjust the time-windows according to the arc usage of the newly found route and their geographically dependent arcs. Note that this implies that one does not have to take care of the physical dimensions of the AGVs during route computation, since it is already fully represented by readjusting the time-windows on all affected arcs.

As mentioned before, the advantage of this approach is the fact that the problems of Section 2.1 are avoided because in a conflict-free approach there is no need for an additional collision avoidance since the routes are planned conflict-free in advance.

Additionally, as a welcome side effect, the completion time of a request is known immediately after route computation since the time-dependent behavior is fully modeled. This is a great advantage for a higher-level management system which plans the requests.

3 The Algorithm

The algorithm consists of two parts. The first part is a preprocessing step; during the second part all requests are routed iteratively in a real-time route computation and for each computed route the time-windows of the affected arcs are adjusted.

The structure of the real-time computation (route computation and readjustment of time-windows) is illustrated in Fig. 5.

3.1 Preprocessing

The preprocessing step determines the conflict sets and the turning rules for each arc a. First, all polygons $P(a)$ (see Section 2.2) are compared pairwise. If the polygons $P(a)$ and $P(b)$ of arcs $a, b \in A(G)$ intersect, then a is added to *confl*(b) and b is added to *confl*(a). Second, one computes for each arc a a list $OUT(a)$ of arcs containing those arcs b that are permitted to be used after arc a on a feasible route respecting the physical properties of an AGV. This is done only once for a given layout (harbor).

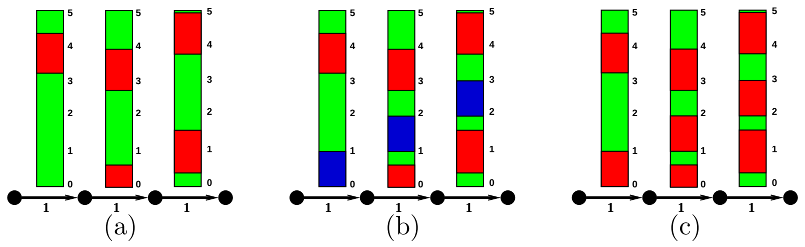
Figure 5: Illustration of the real-time computation on three consecutive arcs with transit time 1. (a) shows the situation before the new request arrives. There is a graph with some blockings (red) and some time-windows (green) on the time axis (y axis). The task is to compute a quickest path that respects the time-windows. This is illustrated in (b). The chosen path is blocked afterwards (see (c)).

3.2 Route computation: quickest paths with time-windows

As pointed out in Section 2.2, the route computation can be done in an idealized model where the dimension of the AGV need no longer to be considered, since the conflict sets take care of this. Instead, one just has to compute a route for an infinitesimal mass point representing the center of the AGV.

This simplified problem is related to the *Shortest Path Problem with Time-Windows (SPPTW)* [3, 4, 5, 14] and can be formulated as follows: Given a graph G, a source node s, a destination node t, a start time θ, transit times $\tau(a)$, costs $c(a)$ and a set of time-windows $\mathcal{F}(a)$ on each arc a; compute a shortest path (w.r.t. arc costs $c(a)$) that respects the given time-windows.

Since AGVs are allowed to stop during their route, waiting is allowed on such a path. 'Respecting' the time-windows means that AGVs wait on an arc or traverse an arc a only during one of its "free" time-windows given by $\mathcal{F}(a)$.

The SPPTW and also our variant is $\mathcal{NP}$-hard. The hardness can be shown by reduction of the Constrained Shortest Path Problem (CSPP [1]).[1]

Our algorithm for this problem is a generalized arc-based label setting algorithm resembling Dijkstra's algorithm. A *label* $L = (a_L, c_L, I_L, pred_L)$ on an arc a_L consists of a cost value c_L, a predecessor $pred_L$ and a time interval I_L. Each label L represents a path from start node s to the tail of a_L, whereas c_L contains the cost value of the path up to the tail of a_L; the label interval $I_L = (A_L, B_L)$ represents an interval of possible arrival times at arc a_L (at the tail of a_L); $pred_L$ is the predecessor of a_L on that path. We define an ordering for these labels. We say that a label L *dominates* a label L' if and only if

$$c_L \leq c_{L'} \quad \text{and} \quad I_{L'} \subseteq I_L.$$

The labels are stored in a priority queue H, e.g., a binary heap. The gener-

[1]The instance of the SPPTW is constructed by placing time-windows $[0, R]$ at each arc while R denotes the resource constraint in the CSPP instance.

alized arc-based Dijkstra algorithm works as follows.

- **Initialization.**
  Create a label $L = (a, 0, (\theta, \infty), nil)$ for all out-going arcs a of s and add them to the priority queue H.

- **Loop.**
  Take the label L with lowest cost value c_L from H. If there is no label left in the queue, output the information that there is no feasible path from s to t. If t is the tail of a_L, output the corresponding path.

  – **For** each time-window on arc a_L.

    * **Label Expansion.**
      Try to expand the label interval along a_L through the time-window of the arc a_L (new label interval should be as large as possible, see Fig. 6), add the costs $c(a_L)$ to the cost value c_L and determine the new predecessor. If there is no possible expansion, consider the next time-window of arc a_L.
    * **Dominance Test.**
      For each out-going arc a in $OUT(a_L)$, add the new label to the heap if it is not dominated by any other label on a. Delete the labels in the heap that are dominated by the new label.

Since the SPPTW is $\mathcal{NP}$-hard the algorithm cannot be executed in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. However, the AGV routing problem differs from the SPPTW in a subtle point. In AGV routing, the cost of a path is the sum of the transit times of the arcs on the path plus waiting times on arcs which is the crucial property that makes the problem polynomial. Thus, the costs on an arc a (the transit time of a plus possible waiting time on a) depend no longer only on the arcs itself, but also on the routing history given by the label interval and the current time-window. We call the resulting problem the *Quickest Path Problem with Time-Windows (QPPTW)*.

For the QPPTW we can obtain a polynomial time algorithm, since here the costs correlate to the lower bounds of the label intervals.

Theorem 1. *The described generalized arc-based Dijkstra algorithm solves the QPPTW in polynomial time (in the number of time-windows).*

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

Consider the correlation between costs and traveling time (including waiting times): they differ only by an additive constant, namely the starting time. Thus, for any two labels that are expanded w.r.t. the same time-window the cost value controls the dominance relation. Hence, one label dominates another if and only if it has a lower transit time.

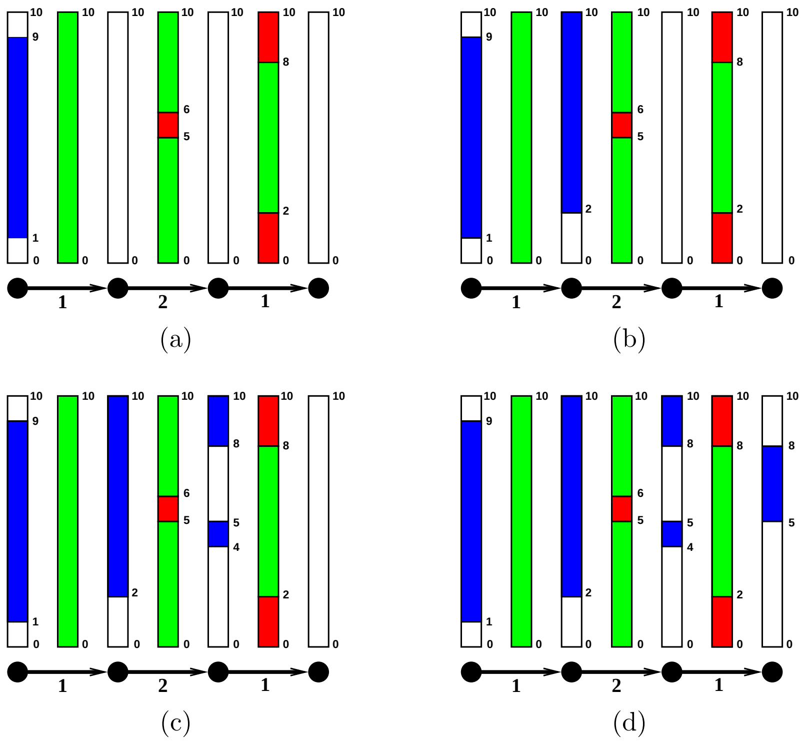

Figure 6: Label Expansion on three consecutive arcs. The label intervals are represented by blue bars and are placed above the nodes. The blockings are colored red (arcs). The green intervals between these blockings are the time-windows. The figures (a) to (d) show the successive expansion of the label intervals.

Therefore, the number of possible labels on an arc a is bounded by the number of time-windows on all in-going arcs (in-going time-windows $\mathcal{F}^-(a)$). As a consequence, the number of iterations in each loop (the number of labels taken from the priority queue) is bounded from above by the product of the number of arcs and the maximum number of in-going time-windows over all arcs ($\sum_{a\in A} |\mathcal{F}^-(a)|$). In each iteration a label is expanded along at most the number of time-windows $|\mathcal{F}(a)|$ at a and each of the resulting labels is compared with at most $\sum_{b\in OUT(a)} |\mathcal{F}^-(b)|$ existing labels. If the priority queue is implemented as a heap, updating can be done in $O(\log(\sum_{a\in A} |\mathcal{F}^-(a)|))$. This leads to the following run time:

$$O\left(\left(\sum_{a\in A} |\mathcal{F}^-(a)|\right)\cdot\left(\max_{a\in A}|\mathcal{F}_a|\right)\cdot\left(\max_{a\in A}\{\sum_{b\in OUT(a)}|\mathcal{F}^-(b)|\}\right)\cdot\log\left(\sum_{a\in A}|\mathcal{F}^-(a)|\right)\right).$$

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

For an additional acceleration of the algorithm we use *goal-oriented search* [8, 15].

After each route computation we verify for each arc a of the computed path, whether the time-windows on the arcs in *confl(a)* have to be readjusted.

4 Additional Practical Requirements

To make the algorithm practical, additional ingredients have to be taken into account.

4.1 Container orientation

Since containers are not completely symmetric, it may be necessary to give an AGV an explicit target orientation. We model this orientation constraint by a flag, indicating whether the AGV is in the right driving direction to reach the target in the correct orientation without a turn.

To this end, we maintain labels at an arc for each of the two possible directions and define the domination rule accordingly. Using the observations of Theorem 1 we get the following result.

Theorem 2. *The described generalized arc-based Dijkstra algorithm solves the QPPTW in polynomial time (in the number of time-windows) even if the orientation of AGVs (containers) is taken into consideration.*

4.2 Safety tubes and re-routing

In spite of the fact that the computed routes are conflict-free, additional safety is required in practice because the AGVs possibly deviate from the computed routes in time. Also technical problems may occur while traveling through the network. We have implemented two different safety tubes, a distance-dependent and a time-dependent one, and re-routing techniques to cope with this difficulty.

The distance-dependent tube blocks an area in front of the AGV. The length depends on the speed of the AGV and is at least the distance needed to come to a complete stop (braking distance). This allows the AGV to stop if something unexpected happens (for example an unexpected stop of another AGV) without causing a collision.

The time-dependent 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 differences between computed times in the model and the times when the AGVs reaches a point in reality.

In order to cope with more challenging perturbations as large deviations from the expected starting time, lower driving speeds than expected or vehicle breakdowns we also implemented re-routing strategies based on the described algorithm.

4.3 Non constant transit times

Instead of constant transit times as described in Section 2, we take the variable speed of the AGVs into account, i.e., we model the acceleration behavior and the different possible maximum speeds.

The maximum speed depends on the kind of movement (curve or straight section), the weather conditions, and the status of the AGV. The acceleration value depends on the current speed.

5 Computational Results

We now address two important questions with our approach.

- Is the approach better than the static one?
- Is the algorithm suitable for real-time computation?

Both questions can be answered in the affirmative. The comparison of both approaches shows that the conflict-free approach is superior to the static one (exact numbers at CTA have to be kept confidential). Additionally, the presented algorithm is able to provide fast answers. On average, the computation in all scenarios does not require more than a few hundredth of a second. And also the maximum values of less than half a second are small enough to ensure fast real-time computation in practice.

Acknowledgments. We are grateful to Andreas Parra and Kai-Uwe Riedemann from the Hamburger Hafen und Logistik AG (HHLA) and Boris Wulff from Container Terminal Altenwerder (CTA) for their valuable remarks and suggestions and for providing us with real world data.

References

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

[2] Broadbent, A. J. et al. (1987) Free-ranging AGV and Scheduling System. In Automated Guided Vehicle Systems, 301–309

[3] Desrosiers et al. (1986) Methods for routing with time windows. European Journal of Operations Research 23, 236–245

[4] Desrosiers, M., Soumis, F. (1988) A generalized permanent labelling algorithm for the Shortest Path Problem with Time Windows, INFOR 26(3), 191–212

[5] Desrosiers, J., Solomon, M. (1988) Time window constrained routing and scheduling problems, Transportation Science 22, 1-13

[6] Ford, L. R., Fulkerson, D. R. (1959) Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433

[7] Ford, L. R., Fulkerson, D. R. (1962) Flows in Networks. Princeton University Press, Princeton, NJ

[8] Hart, P., Nilsson, N., Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. In IEEE Transactions on Systems, Science and Cybernetics SCC-4, 100–107

[9] Kim, K. H., Jeon, S. M., Ryu, K. R. (2006) Deadlock prevention for automated guided vehicles in automated container terminals. OR Spectrum 28 (4), 659–679

[10] Köhler, E., Möhring, R. H., Skutella, M. (2002) Traffic networks and flows over time. In Jürg Kramer, Special Volume Dedicated to the DFG Research Center "Mathematics for Key Technologies Berlin", published by Berliner Mathematische Gesellschaft, 49–70, http://www.math.tu-berlin.de/coga/publications/techreports/2002/Report-752-2002.html

[11] Krishnamurthy, N., Batta, R., Karwan, M. (1993) Developing conflict-free routes for automated guided vehicles. Operations Research 41, 1077–1090

[12] Moorthy, K. M. R. L., Guan, W. H. (2000) Deadlock Prediction and Avoidance in an AGV System. SMA Thesis

[13] Qui, J., Hsu, W.-J. (2000) Conflict-free AGV routing in a bi-directional path layout. In Proceedings of the 5th International Conference on Computer Integrated Manufacturing (ICCIM 2000), volume 1, 392–403

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

[15] Sedgewick, R., Vitter, J.S. (1986) Shortest paths in Euclidian graphs. Algorithmica 1, 31–48

[16] Spenke, I. (2006) Complexity and Approximation of Static k-Splittable Flows and Dynamic Grid Flows. PhD Thesis Technische Universit Berlin

[17] Taghaboni-Dutta, F., Tanchoco, J. M. A. (1995) Comparison of dynamic routeing techniques for automated guided vehicle system. International Journal of Production Research 33(10), 2653–2669

[18] Vis, I. F. A. (2006) Survey of research in the design and control of automated guided vehicle systems. European Journal of Operational Research 170, 677–709

Reports from the group

"Combinatorial Optimization and Graph Algorithms"

of the Department of Mathematics, TU Berlin

2005/03 *Nicole Megow, Marc Uetz, and Tjark Vredeveld:* Models and Algorithms for Stochastic Online Scheduling

2004/37 *Laura Heinrich-Litan and Marco E. Lübbecke:* Rectangle Covers Revisited Computationally

2004/35 *Alex Hall and Heiko Schilling:* Flows over Time: Towards a more Realistic and Computationally Tractable Model

2004/31 *Christian Liebchen and Romeo Rizzi:* A Greedy Approach to Compute a Minimum Cycle Bases of a Directed Graph

2004/27 *Ekkehard Köhler and Rolf H. Möhring and Gregor Wünsch:* Minimizing Total Delay in Fixed-Time Controlled Traffic Networks

2004/26 *Rolf H. Möhring and Ekkehard Köhler and Ewgenij Gawrilow and Björn Stenzel:* Conflict-free Real-time AGV Routing

2004/21 *Christian Liebchen and Mark Proksch and Frank H. Wagner:* Performance of Algorithms for Periodic Timetable Optimization

2004/20 *Christian Liebchen and Rolf H. Möhring:* The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond

2004/19 *Ronald Koch and Ines Spenke:* Complexity and Approximability of k-splittable flow problems

2004/18 *Nicole Megow, Marc Uetz, and Tjark Vredeveld:* Stochastic Online Scheduling on Parallel Machines

2004/09 *Marco E. Lübbecke and Uwe T. Zimmermann:* Shunting Minimal Rail Car Allocation

2004/08 *Marco E. Lübbecke and Jacques Desrosiers:* Selected Topics in Column Generation

2003/050 *Berit Johannes:* On the Complexity of Scheduling Unit-Time Jobs with OR-Precedence Constraints

2003/49 *Christian Liebchen and Rolf H. Möhring:* Information on MIPLIB's timetab-instances

2003/48 *Jacques Desrosiers and Marco E. Lübbecke:* A Primer in Column Generation

2003/47 *Thomas Erlebach, Vanessa Kääb, and Rolf H. Möhring:* Scheduling AND/OR-Networks on Identical Parallel Machines

2003/43 *Michael R. Bussieck, Thomas Lindner, and Marco E. Lübbecke:* A Fast Algorithm for Near Cost Optimal Line Plans

2003/42 *Marco E. Lübbecke:* Dual Variable Based Fathoming in Dynamic Programs for Column Generation

2003/37 *Sándor P. Fekete, Marco E. Lübbecke, and Henk Meijer:* Minimizing the Stabbing Number of Matchings, Trees, and Triangulations

2003/25 *Daniel Villeneuve, Jacques Desrosiers, Marco E. Lübbecke, and François Soumis:* On Compact Formulations for Integer Programs Solved by Column Generation

2003/24 *Alex Hall, Katharina Langkau, and Martin Skutella:* An FPTAS for Quickest Multicommodity Flows with Inflow-Dependent Transit Times