EURO J Transp Logist (2012) 1:87–111
DOI 10.1007/s13676-012-0008-7

RESEARCH PAPER

Conflict-free vehicle routing

Load balancing and deadlock prevention

Ewgenij Gawrilow · Max Klimm · Rolf H. Möhring · Björn Stenzel

Received: 27 October 2011 / Accepted: 30 March 2012 / Published online: 8 May 2012

Abstract Collision-free vehicle routing occurs in many applications from robot movements via scheduling multiple cranes in steel logistics to the transport of containers by automated guided vehicles. In cooperation with the HHLA Container Terminal Altenwerder (CTA), we have developed a dynamic online routing algorithm that computes collision-free routes for AGVs by considering implicit time-expanded networks. This approach proved to be very efficient in scenarios with a high traffic density. This is in contrast to the more frequent static approaches that use routes computed in the static graph—i.e., without time expansion—and employ additional methods for collision avoidance during execution of the static routes. These static approaches have the advantage that they are quite robust against disturbances but are rather unpredictable because of the usually heuristic collision avoidance rules that may even run into deadlocks in high traffic scenarios. In this paper, we study if the static approach can be suitably enhanced to meet the performance of the dynamic approach and become predictable and collision and deadlock-free. Our approach is based on online static route computations combined with load balancing techniques and graph algorithms for guaranteed deadlock avoidance. We evaluate our static algorithm on routing scenarios from the HHLA CTA. It turns out that the performance of our static router is slightly superior in low and medium traffic scenarios, but loses against the dynamic router in high traffic scenarios.

E. Gawrilow
TomTom Development GmbH, An den Treptowers 1, 12435 Berlin, Germany
e-mail: gawrilow@math.tu-berlin.de

M. Klimm · R. H. Möhring (✉) · B. Stenzel
Technische Universität Berlin, Institut für Mathematik, Arbeitsgruppe Kombinatorische Optimierung und Graphenalgorithmen, MA 5-1 Straße des 17. Juni 136, 10623 Berlin, Germany
e-mail: rolf.moehring@tu-berlin.de

M. Klimm
e-mail: klimm@math.tu-berlin.de

B. Stenzel
e-mail: stenzel@math.tu-berlin.de

Keywords Vehicle routing · Collision avoidance · Deadlock prevention

Mathematics Subject Classification 90C27 · 90B06 · 05C85

Introduction

Collision-free routing occurs in many applications from robot movements via scheduling multiple cranes in steel logistics to the transport of containers with automated guided vehicles (AGVs). Typically, AGVs are not equipped with an intelligent local-collision-avoidance system and rely solely on cental control. This is usually done by static approaches that first compute routes in the underlying routing graph and then use additional methods during route execution for collision avoidance. These rules are usually heuristic and thus may lead to unpredictable arrival times and even deadlocks.

In cooperation with the HHLA container terminal altenwerder (CTA), Gawrilow et al. (2008) have developed a dynamic routing algorithm that computes collision-free routes for AGVs by considering implicit time-expanded networks. This approach proved to be very efficient in scenarios with a high traffic density. However, due to the time-sensitive nature of dynamic routes, this approach requires recomputation of routes when delays of AGVs and other disturbances occur. In contrast, the static approach has the advantage that the computed routes do not change and are thus robust against disturbances.

In this paper, we study if the static approach can be suitably enhanced to meet the performance of the dynamic approach and become collision and deadlock-free. Our approach is based on static route computations combined with a subsequent collision avoidance that employs a particular reservation procedure of parts of the precomputed routes, called *claiming*. It is realized in two stages. In the first stage we use load balancing algorithms to compute suitable static routes that keep the load on the edges of the routing graph small. To that end, we consider an *online load balancing problem with bounded stretch factor* and develop an optimal algorithm with respect to a specific performance ratio, the stretch factor restricted competitive ratio, see "Online load balancing with bounded stretch factor". Herein, we use methods from Aspnes et al. (1997) and Gao and Zhang (2004).

In a second phase, we then develop in "Reservation schedules and deadlock prevention" a claiming mechanism that fully avoids collisions and deadlocks. Based on the computed static routes, we define a reservation schedule that assigns exclusive claims to parts of an AGV's route, and an associated deadlock avoidance graph for making the reservation schedule deadlock-free. It turns out that detecting deadlocks in this graph is closely related to detecting so-called colorful cycles in graphs, a problem investigated by Alon et al. (1995). Based on their results, we show that deadlock detection is NP-complete and develop an algorithm for deadlock detection that is polynomial in the size of the routing graph, and exponential only in the number of transportation requests.

In "Computational results", we evaluate the combined algorithm on routing scenarios from the HHLA CTA. It turns out that, for a suitable choice of the stretch

factor in the load balancing part, the performance of our static router with guaranteed collision and deadlock avoidance is slightly superior in low and medium traffic scenarios. But it loses against the dynamic router in high traffic scenarios, i.e., for the same number of transportation requests but with increasing number of blocked lanes in the routing graph.

In the remainder of this section, we will give an introduction into our model, our results, and previous work. A closer look at related work and the discussion of our results will be given in the corresponding sections on online load balancing with bounded stretch factor ("Online load balancing with bounded stretch factor"), on reservation schedules and deadlock prevention ("Reservation schedules and deadlock prevention"), and on computational results ("Computational results").

The model

The routing graph is a directed graph $G = (V, E)$. Its edges $e \in E$ describe the lanes or streets in the routing area. Each edge $e \in E$ has a constant transit time $\tau(e)$ that indicates the time needed to traverse this edge. The node set V models the crossings of the lanes.

Transportation requests are arriving over time in an online fashion and are modeled by a sequence $\sigma = r_1, \ldots, r_k$ of requests. Each request r_i consists of a start node s_i, a target node t_i, and a specific AGV to carry out that transport. Note that we do not consider the assignment of vehicles to requests. We assume that this is done by a higher-level management system. The goal is to compute in an online fashion, routes (paths) for the AGVs that are collision-free, i.e., at any time, no two AGVs meet on an edge or in a node of the routing graph. We call this problem an online disjoint vehicle routing problem.

Static approaches for such online disjoint vehicle routing problems work as follows. One computes static paths in the network, ignoring the movement of AGVs over time. More precisely, one computes shortest paths for each request, e.g., using Dijkstra's algorithm, with respect to arc costs consisting of the transit times $\tau(e)$ plus a load dependent penalty cost that is a function of the number of previously computed routes that are already using this edge.

By the static nature of this approach, the computed routes may lead to collisions at execution time. Hence, one needs an additional conflict management system 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 *reserved area*) and block it for all other vehicles (*reservation*) until it has been traversed, see Fig. 1. The other vehicles must then wait along their route until they are allocated the next part of their route.

Static approaches have the advantage that the routes are known in advance and do not depend on the actual travel behavior of the vehicles. They are thus robust against small delays and other disturbances. However, the necessary conflict management system at execution time may have a deteriorating effect on the system performance. It may lead to unpredictable arrival times, which is bad for the next transportation request assigned to that vehicle, to detours, to a higher congestion, or even to deadlocks. A *deadlock* is a situation in which a group of vehicles wish to

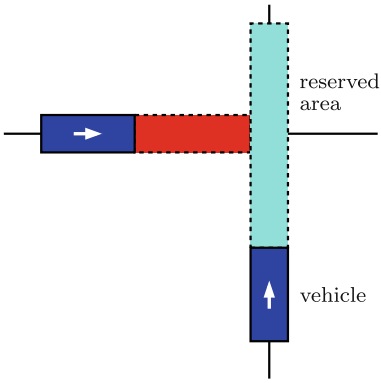


Fig. 1 Reservation procedure. Each vehicle reserves the next part of its route. Mutually exclusive reservations guarantee a collision-free execution of computed (static) paths

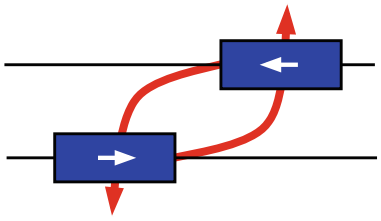


Fig. 2 Simplified deadlock situation. Both vehicles are trying to occupy the same portion/edges of the network, thereby blocking each other

reserve a set of edges which are already occupied by other vehicles in this group such that none of them is able to continue its route and thus the system is blocked. See Fig. 2 for the smallest possible such deadlock.

In contrast, the dynamic routing algorithm of Gawrilow et al. (2008) computes collision-free routes already at computation time by considering an implicit time-expansion of the routing graph. These time-dependent routes permit waiting at the start position and on edges along theirs path. In order to ensure robustness against delays and other disturbances, it applies a rerouting at execution time when disturbances occur, which may lead to additional waiting along an AGV's path, or even a new route from an AGV's current position. It is therefore more flexible at the cost of a higher computational overhead that requires a very fast dynamic router during execution time.

Our approach and results

We develop a two-phase static algorithm. The first phase computes static routes, and the second phase computes a reservation schedule that assigns exclusive claims to parts of an AGV's route, such that the execution of the routes is collision and deadlock-free.

For computing the static routes, we use load balancing techniques that balance the load on the edges of the routing graph such that the resulting paths remain short with respect to the given transit times. This leads in "Online load balancing with bounded stretch factor" to an online load-balancing algorithm that, among all online

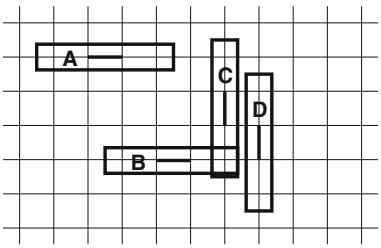


Fig. 3 The figure illustrates the *polygons* that are claimed by a vehicle that moves on the indicated edge. *Polygons B and C* intersect each other while *polygons A and D* do not intersect any other polygon

algorithms, achieves the best possible load subject to a length constraint on the chosen paths (the *stretch factor*).

In a second phase we construct from these paths a reservation schedule that assigns exclusive claims to parts of a vehicle's route and thus avoids collisions, and an associated deadlock avoidance graph for making the reservation schedule also deadlock-free.

For dealing with the physical dimensions of the vehicles we use polygons $\mathcal{P}(e)$ for each edge e that describe the blocked area when a vehicle (more precisely, the center of a vehicle) is located on edge e (Fig. 3). Thus, it is prohibited to use two edges at the same time if the corresponding polygons intersect. We represent this mutual exclusion by sets *confl*(e) of so-called *conflicting edges* for each edge e.

We show in "Deadlock prevention algorithm" that deadlock detection is NP-hard and develop an algorithm for deadlock detection that is polynomial in the size of the routing graph, but exponential in the number of transportation requests. To show these results, we adapt techniques from a paper on colorful cycles in graphs by Alon et al. (1995) to our setting.

Algorithm 1 gives a high-level description of our combined algorithm. We evaluate it on several routing scenarios from the HHLA CTA. We first report on the *average travel time* along the computed paths and the *computation times* to obtain them. In order to analyze our load balancing algorithm, we also report on the *(static) length* of the computed paths and the resulting *load* on the edges of the routing graph. Finally, we evaluate the *number and the length of the cycles* found by the deadlock detection algorithm.

Algorithm 1: STAT-ROUTE

Data: Directed 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 (see "Load balancing algorithm");

    compute a deadlock-free reservation schedule (see "Deadlock prevention Algorithm")

end

Previous work

So far, only Guan and Moorthy (2000) investigated different penalty costs for the online disjoint vehicle routing problem. They considered constant additive penalty costs together with distance-dependent costs. They report on an experimental evaluation, but do not provide theoretical performance guarantees. Their experiments do not show any major effect.

The detection and prevention of deadlocks has been investigated by Lee and Lin (1995) who considered Petri net approaches. Other, graph theoretic models have been investigated by Cho et al. (1995); Guan and Moorthy (2000); Kim et al. (2006); Yeh and Yeh (1998). For details about these approaches we refer to "Reservation schedules and deadlock prevention".

Online load balancing with bounded stretch factor

Introduction

Online load balancing problems have been investigated since the introduction of online algorithms, see e.g., Aspnes et al. (1997); Borodin and El-Yaniv (1998).

Consider a directed graph $G = (V, E)$ and a sequence of routing 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 request i, respectively. Sometimes a request is assigned an additional bandwidth that may depend on the edges used. 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 denotes the total number of requests.

Online load balancing problem	
Instance	Directed graph $G = (V, E)$, sequence of requests $\sigma = (r_1, \ldots, r_k)$
Task	Minimize the maximum load over all edges $e \in E$, i.e., min $\max_{e \in E} \text{load}_k(e)$

Aspnes et al. (1997) presented an $O(\log(|E|))$-competitive algorithm for this problem and, using a lower bound of Azar et al. (1992) for online assignment, showed that their approach is optimal for online load balancing (in the sense of achieving the best possible competitive ratio over all online algorithms).

This standard load balancing problem has been extended by Gao and Zhang (2004) to permit transit times $\tau(e)$ on the edges and constraints on the lengths of the chosen path. They introduced a so-called stretch factor $B > 1$ that bounds the length of a chosen $s_i - t_i$-path, i.e., the length (measured as transit time) of each $s_i - t_i$-path is less than B times the length of a shortest path between s_i and t_i. We call this problem the online load balancing problem 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	Minimize the maximum load over all edges $e \in E$, i.e., min $\max_{e \in E} \text{load}_k(e)$, subject to $\text{length}(P_i) < B \cdot \text{length}(\text{SP}_i)$ for each chosen $s_i - t_i$-path P_i, where here $\text{length}(\text{SP}_i)$ denotes the length of a shortest path between s_i and t_i measured as transit time.

Gao and Zhang (2004) modified the approach of Aspnes et al. (1997) and obtained similar results concerning the competitive ratio for this problem. In particular, they also provide an $O(\log(|E|))$-competitive algorithm. Note that their algorithm has an exponential run time since they compute resource-constrained shortest paths.

We also consider the online load balancing problem with bounded stretch factor, but focus on a different analysis. Instead of comparing the solution of a particular online algorithm with the optimal solution for that problem (in the sense of competitive analysis), we compare it with an optimal solution of the standard load balancing problem, i.e., the offline version without stretch factor constraints. We are interested in this ratio since an optimal load balancing solution in our sense presents the best choice with respect to the congestion generated by our static routing algorithm STATE-ROUTE (Algorithm 1). We refer to it as the *stretch factor restricted (sfr) competitive ratio* and transfer the notation from the standard competitive analysis introduced in Borodin and El-Yaniv (1998) to our criterion.

Definition 1 An online algorithm ALG for the online load balancing problem with bounded stretch factor is c-stretch-factor-restricted-competitive (c-sfr-competitive for short) for a constant c if, for any problem instance $\mathcal{I}$, it computes a solution $\text{ALG}(\mathcal{I})$ with

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

where $\text{OPT}(\mathcal{I})$ denotes the optimal value of the offline version of 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 on k, is given in advance. Seiden et al. (2000) call such approaches semi-online. In our application a good upper bound might be the number of vehicles since there cannot be more 'active' requests than vehicles.

Below we develop a semi-online algorithm and show that it is optimal with respect to its sfr competitive ratio.

Load balancing algorithm

We present an

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

algorithm (Algorithm 2) for online load balancing with bounded stretch factor. The algorithm works in phases. In each phase we consider a certain upper bound UB on the optimal load with respect to the already routed requests in this phase. This upper bound is adjusted depending on the current maximum load produced by the algorithm. Whenever it cannot be guaranteed that UB is still an upper bound, cf. Theorem 1, we double the bound and enter a new phase.

Algorithm 2: 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
    load(e) = 0 ∀e ∈ E ;
    UB = 1;
    b = ∛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) = τ(e) · b^(load(e)/UB) ;
        if ∃e ∈ confl(P_i) : load(e) + 1 > log_b(b^(k+1) · |E| · max_{e∈E} τ(e) / min_{e∈E} τ(e)) · UB then
            /* new phase */
            UB = 2 · UB ;
            load(e) = 0 ∀e ∈ E;
            goto line SP /* repeat shortest path computation */;
        else
            load(e) = load(e) + 1 ∀e ∈ 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. Afterwards, the load on all edges that conflict with an edge of the selected path P is increased by one. These are all edges in $\mathit{confl}(P) := \bigcup_{e\in P} \mathit{confl}(e)$.

In order to analyze the performance ratio of Algorithm 2 (BAL-BOUND) we consider a generic phase and formulate it in greater detail in Algorithm 3 (SUB-BAL-BOUND).

Algorithm 3: 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 on the optimum.

Result: Sequence of static paths $P_1, \ldots, P_k$.

begin

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

$c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$;

load$(e) =$ load$(e) + 1$ $\forall e \in \mathit{confl}(P_i)$;

assign path P_i to request r_i;

end

Theorem 1 gives a performance guarantee for Algorithm 3 that depends on the given upper bound UB

Theorem 1 *Consider a problem instance $\mathcal{I}$ and a fixed upper bound UB > OPT. Then*

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

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

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

We used here that the load on each edge is bounded by k (by definition a static path contains no cycle). Summing up over all requests gives

$$\sum_{i=1}^{k} \sum_{e \in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} \leq \sum_{i=1}^{k} \sum_{e \in P_i^*} \tau(e) b^{\frac{k}{\text{UB}}} \tag{1}$$

$$\Leftrightarrow \sum_{e \in E} \sum_{i : e \in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} \leq \sum_{e \in E} \sum_{i : e \in P_i^*} \tau(e) b^{\frac{k}{\text{UB}}} \tag{2}$$

Multiplying the left hand side of (2) with $b^{\frac{1}{\mathrm{UB}}} - 1$ we obtain

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{i:e \in P_i} \tau(e) b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}} = \sum_{i:e \in P_i} \tau(e) \left(b^{\frac{\mathrm{load}_{i-1}(e)+1}{\mathrm{UB}}} - b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}} \right) \tag{3}$$

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{i:e \in P_i} \tau(e) b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}} \geq \sum_{i:e \in P_i} \tau(e) \cdot \left(b^{\frac{\mathrm{load}_{i}(e)}{\mathrm{UB}}} - b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}} \right) \tag{4}$$

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{i:e \in P_i} \tau(e) b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}} = \tau(e) \cdot (b^{\frac{\mathrm{load}_{k}(e)}{\mathrm{UB}}} - 1). \tag{5}$$

The telescoping sum in (4) follows by observing that the load on an edge e increases by at most one in a single step ($\mathrm{load}_i(e) \leq \mathrm{load}_{i-1}(e) + 1$).

We now multiply the right hand side of (2) also with $(b^{\frac{k}{\mathrm{UB}}} - 1)$ and obtain

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{e \in E} \sum_{i:e \in P_i^*} \tau(e) b^{\frac{k}{\mathrm{UB}}} \leq \frac{b}{\mathrm{UB}} \cdot \sum_{e \in E} \sum_{i:e \in P_i^*} \tau(e) b^{\frac{k}{\mathrm{UB}}} \tag{6}$$

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{e \in E} \sum_{i:e \in P_i^*} \tau(e) b^{\frac{k}{\mathrm{UB}}} = b^{\frac{k}{\mathrm{UB}}} b \cdot \sum_{e \in E} \tau(e) \sum_{i:e \in P_i^*} \frac{1}{\mathrm{UB}} \tag{7}$$

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{e \in E} \sum_{i:e \in P_i^*} \tau(e) b^{\frac{k}{\mathrm{UB}}} \leq b^{\frac{k}{\mathrm{UB}}} b \cdot \sum_{e \in E} \tau(e). \tag{8}$$

$$(b^{\frac{1}{\mathrm{UB}}} - 1) \cdot \sum_{e \in E} \sum_{i:e \in P_i^*} \tau(e) b^{\frac{k}{\mathrm{UB}}} \leq b^{\frac{k}{\mathrm{UB}}+1} \cdot \sum_{e \in E} \tau(e). \tag{9}$$

The inequality in (6) holds since $b^{\frac{1}{\mathrm{UB}}} - 1 \leq \frac{b}{\mathrm{UB}}$. To see this, replace UB by n. Then the inequality in (6) is equivalent to $b \leq (1 + \frac{b}{n})^n$. For fixed $b \geq 1$ and $n \to \infty$, the r.h.s. converges monotonously to e^b. Since $b \leq (1 + \frac{b}{n})^n$ already for $n = 1$, the inequality follows.

Inequality (7) $\leq$ (8) holds since the number of paths that are routed over a certain edge in an optimal solution is bounded by UB. Finally, (9) uses that $b > 1$ and $\mathrm{UB} > 1$.

Combining the inequalities for the l.h.s. and the r.h.s. we show the claimed performance guarantee by simple arithmetic transformations [similar to those in Aspnes et al. (1997)]:

$$\begin{aligned}
&\sum_{e \in E} \tau(e) (b^{\frac{load_k(e)}{\mathrm{UB}}} - 1) \leq b^{\frac{k}{\mathrm{UB}}+1} \cdot \sum_{e \in E} \tau(e) \\
\Leftrightarrow\ &\sum_{e \in E} \tau(e) b^{\frac{load_k(e)}{\mathrm{UB}}} \leq (b^{\frac{k}{\mathrm{UB}}+1} + 1) \cdot \sum_{e \in E} \tau(e) \\
\Rightarrow\ &\sum_{e \in E} \tau(e) b^{\frac{load_k(e)}{\mathrm{UB}}} < b^{k+1} \cdot \sum_{e \in E} \tau(e) \\
\Rightarrow\ &\min_{e \in E} \tau(e) \sum_{e \in E} b^{\frac{load_k(e)}{\mathrm{UB}}} \leq b^{k+1} \cdot |E| \cdot \max_{e \in E} \tau(e)
\end{aligned}$$

$$\Leftrightarrow \sum_{e\in E} b^{\frac{load_k(e)}{\mathrm{UB}}} \le b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}$$

$$\Rightarrow \max_{e\in E} b^{\frac{load_k(e)}{\mathrm{UB}}} \le b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}$$

$$\Leftrightarrow \max_{e\in E}\mathrm{load}_k(e) \le \log_b\left(b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}\right)\cdot\mathrm{UB}.$$

The above proof assumes a fixed upper bound UB. Aspnes et al. (1997) showed that adapting the upper bound appropriately in the different phases increases the competitive ratio by at most a factor of 4. We use their approach to prove Theorem 2.

Theorem 2 *Algorithm* BAL-BOUND *is* $4\cdot\log_b\left(b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}\right)$*-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 introduce some notation. BAL-BOUND$_{UB}$ refers to Algorithm 2 with fixed upper bound *UB*. Let $c := \log_b\left(b^{k+1}\cdot|E|\cdot\frac{\max_{e\in E}\tau(e)}{\min_{e\in E}\tau(e)}\right)$ be the performance ratio of SUB-BAL-BOUND from Theorem 1 and recall that algorithm SUB-BAL-BOUND can be viewed as a subroutine of BAL-BOUND.

Let $\sigma^{(\ell)}$ denote the subsequence of requests in phase ℓ of algorithm BAL-BOUND. Note that there are at most 2^ℓ many requests in such a subsequence. Then consider phase h in which algorithm BAL-BOUND terminates. If this is the first phase ($h = 0$), we have

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

So let $h \ge 1$. Consider the subsequence $\sigma^{(h\text{-}1)}$ and the first request r_1^h in phase h. This is the request that terminated phase $h-1$. Thus

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

Theorem 1 gives

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

Summing up over all phases leads to the claimed stretch factor restricted competitive ratio:

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

It remains to show that every path computed by BAL-BOUND respects the stretch factor.

Theorem 3 *The length of each path selected by algorithm BAL-BOUND is less than B times the length of a shortest path between the same nodes.*

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

$$\sum_{e \in P_i} c(e) \leq \sum_{e \in \mathrm{SP_i}} c(e)$$
$$\Leftrightarrow \sum_{e \in P_i} \tau(e) \cdot b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}} \leq \sum_{e \in \mathrm{SP_i}} \tau(e) \cdot b^{\frac{\mathrm{load}_{i-1}(e)}{\mathrm{UB}}}.$$

Since static shortest paths do not contain cycles, $\mathrm{load}_{i-1}(e) < i$ for all i. This leads to the following inequality for each i:

$$\sum_{e \in P_i} \tau(e) \cdot b^0 < \sum_{e \in \mathrm{SP_i}} \tau(e) \cdot b^i$$
$$\Leftrightarrow \mathrm{length}_\tau(P_i) < b^i \cdot \mathrm{dist}(s_i, t_i).$$

Using $b = \sqrt[k]{B}$, we obtain the claimed bound on the path lengths, i.e.,

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

A lower bound on the performance guarantee

In order to show that no (online) algorithm can achieve a better performance guarantee concerning the stretch factor restricted competitive ratio we consider the following example.

Example 1 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 Fig. 4. 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.

In this example, only the edge with transit time 1 can be used since routing over any other edge would violate the stretch factor constraint. Instead, an optimal

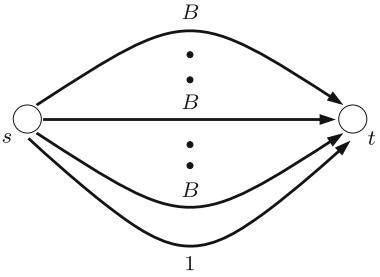


Fig. 4 Graph used in Example 1

solution without this constraint would assign requests to different edges, which leads to a load of 1. Thus the sfr competitive ratio is bounded from below by

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

for any online algorithm for the online load balancing problem with bounded stretch factor.

This shows that Algorithm 2 is (asymptotically) optimal with respect to the ratio presented in Theorem 2.

Remark 1 Note that even an offline algorithm that respects the stretch factor constraint would not be able to perform better for the instance of Example 1.

Reservation schedules and deadlock prevention

Introduction

Although the routes computed by Algorithm 2 (BAL-BOUND) are good in the sense that they balance the load on the edges of the given graph, we still need a mechanism to avoid conflicts. In "The model" we introduce a reservation schedule that prevents the vehicles from colliding. Reservation is done by requesting edges before occupying them. Such a schedule in time-independent and can be interpreted as an instruction for constructing a dynamic path at execution time. Since this procedure may cause deadlocks, we additionally have to take care to avoid such situations.

Since deadlocks are devastating in logistic processes, many approaches for deadlock avoidance have been investigated in the recent years, e.g., based on Petri net approaches by Lee and Lin (1995); Wu and Zhou (2000), or on graph theoretic models as by Cho et al. (1995); Guan and Moorthy (2000); Kim et al. (2006); Yeh and Yeh (1998), see also the overview by Vis (2006).

Most of these approaches use so-called zone controls. They assume that vehicles move between non-intersecting zones of adequate size. This is obviously not a suitable model for our purpose since it is not possible to partition our traffic network into such zones. Kim et al. (2006) introduce a finer zoning of the network, but, just as the other zone control approaches, their algorithm is not able to deal with large vehicle fleets.

Our deadlock prevention algorithm provides both, a fine discretization (vehicles move on 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 Cho et al. (1995); Kim et al. (2006); Yeh and Yeh (1998), in a graph that is in a one-to-one correspondence with the considered reservation schedule. We call this graph the deadlock detection graph. Moreover, in contrast to Guan and Moorthy (2000) we already avoid deadlocks at the time of the route computation and not during the execution of the route.

This section is structured as follows. After the description of our model we introduce the deadlock detection graph in "The deadlock detection graph" and

present the deadlock prevention algorithm in "Deadlock prevention algorithm". Computational results and conclusions are given afterwards.

The model

We consider a set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of static paths that that must be scheduled without deadlocks. A *reservation schedule* $S(\mathcal{P}) = \{R_{P_1}, \ldots, R_{P_k}\}$ for these paths considers the set of *requested edges* $R_P : E(P) \to 2^E$ for each path $P \in \mathcal{P}$. These edge sets $R_P(e)$ contain those edges that must be free, i.e., not occupied by another vehicle, before edge e is left. If this is the case, these edges are reserved. For the polygons used in practice (see Fig. 3), these reservation of edges must be adapted to reserving the polygon associated with the considered AGV. Note that for each path $P = (e_1, \ldots, e_n)$, the set $R_P(e_n)$ of requested edges of the last edge e_n is always empty.

To guarantee a conflict-free execution of a reservation schedule, every edge on a certain path must be requested/reserved before it is entered. Such a reservation schedule is-called *valid*.

Definition 2 (*Valid reservation schedule*) A reservation schedule $S(\mathcal{P})$ is *valid* if every edge on any path $P = (e_1, \ldots, e_n) \in \mathcal{P}$ is reserved by some earlier edge in P, i.e., for each edge $e_i \in P$ there is an edge $e_j \in P \setminus \{e_1, \ldots e_{i-1}\}$ such that $e_i \in R_P(e_j)$.

We assume that the first edge of a path need not be reserved since it is already occupied by the vehicle at its start. Therefore, we may also assume that the first edge is not used by another vehicle.

Assumption 1 (*Reservation of the first edge*) The first edge of a path $P \in \mathcal{P}$ contained in a reservation schedule $S(\mathcal{P})$ is not used and not requested by any other path $P' \in \mathcal{P}$.

A reserved edge is *freed* when the vehicle leaves that edge. To model this, we define the set of *occupied edges* $O_P(e_i)$. An edge is *occupied* by edge e_i if it has already been reserved by earlier edges on the path but not yet been freed, i.e.,

$$O_P(e_i) = \{e_i\} \cup \left(\bigcup_{k<i} \{e_j \in R_P(e_k) | j > i\} \right). \tag{10}$$

Note that the edges in $R_P(e)$ can only be reserved if the corresponding conflicting edges $\mathit{confl}\,(R_P(e)) := \bigcup_{f \in R_P(e)} \mathit{confl}(f)$ are not occupied. Therefore, all these edges must also be requested. But they need not be reserved and are therefore not occupied after the request. To this end, we will always distinguish between requested edges in $R_P(e)$ and conflicting edges in $\mathit{confl}(R_P(e))$.

Since we look for reservation schedules that avoid deadlocks, we need to identify any potential deadlock situation. Here, 'potential' means that the identified situation need not appear, but cannot be excluded. This ambiguity is due to the fact that deadlocks depend also on the order in which vehicles pass potential conflict points.

Since reservation schedules are time-independent, they have no influence on the order. Therefore, a time-independent deadlock definition needs to take all possible orders of the vehicles into account.

Definition 3 Let $S(\mathcal{P})$ be a reservation schedule. 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 $e_{\mathrm{P_i}}$ such that

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

$S(\mathcal{P})$ is called *deadlock-free* if it does not contain a deadlock-ridden subset.

Note that this characterizes potential deadlocks. Actual deadlocks will only occur at run-time if the affected vehicles are stopped, which is the case if requested edges are not free.

Based on the above model we will now develop methods for deadlock detection and prevention.

The deadlock detection graph

We will show that potential deadlocks, i.e., deadlock-ridden sets of paths in a given reservation schedule, correspond to cycles with a specific properties in the deadlock detection graph defined below.

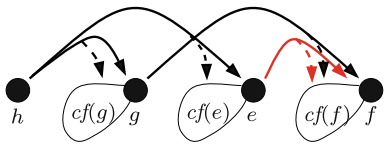
The deadlock detection graph $G_{\mathrm{D}} = (V_{\mathrm{D}}, E_D)$ is derived from a reservation schedule $S(\mathcal{P})$. Its node set V_{D} consists of all edges of the given layout graph $G = (V, E)$, i.e., $V_{\mathrm{D}} = E$. Its edge set E_{D} consists of edges $((e, f), c)$, where $c \in \mathcal{C}$ denotes a particular color that corresponds to requests of edges made in the given reservation schedule $S(\mathcal{P})$. The color set $\mathcal{C}$ contains a separate color for each path of the set $\mathcal{P}$. We denote the color of path P by c_P. The precise definition is as follows.

Definition 4 (*Deadlock detection graph*) Consider a graph $G = (V, E)$ and a reservation schedule $S(\mathcal{P})$. Then, the corresponding deadlock detection graph $G\mathrm{D} = (V_{\mathrm{D}}, E_{\mathrm{D}})$ is constructed as follows:

1. $V_{\mathrm{D}} = E$.
2. $E_{\mathrm{D}} = \{((e,f), c_P) | \exists g \in E \text{ such that } f \in confl(R_P(g)) \text{ and } e \in O_P(g)\}$.

Figure 5 illustrates the construction of the deadlock detection graph from a reservation schedule $S(\mathcal{P})$, i.e., from a collection of sets of requested edges. By definition, $e \in O_P(e)$, so there are loops at every node of G_D corresponding to an edge in a path $P \in \mathcal{P}$ (these are not depicted in Fig. 5). Also every requested edge $f \in confl(R_P(e))$ leads to an edge $((e, f), c_P)$ G_{D}. But there are more edges resulting from the conflict sets.

The deadlock detection graph is defined for arbitrary reservation schedules. We will use use it to detect potential deadlocks. To this end, we define colorful paths and cycles similar to Alon et al. (1995), who considered the corresponding node version in a different context.



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

Fig. 5 Illustration of a deadlock detection graph that corresponds to the reservation schedule (*shown on the right*) for a single path $P = (h, g, e, f)$. The thin loop at a node, say g, with the inscription $cf(g)$ represent the conflict set $confl(g)$ of edge g, and the *dotted arcs* point towards all edges in these conflict sets. The *black edges* result from the set of requested edges, while the *red/light gray edges* are due to the fact that edge e is in $O_P(g)$ and f is requested from g, respectively

Definition 5 (*Colorful path, colorful cycle*) Consider a graph G with colored edges. Then, a colorful path is a static path P in G with the property that any two edges of P are colored differently. A *colorful cycle* is a cycle with the same property.

We will show in Theorem 4 below that a colorful cycle in a deadlock detection graph characterizes a deadlock-ridden set of paths in the corresponding reservation schedule.

Theorem 4 *A reservation schedule $S(\mathcal{P})$ for a 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 4, the deadlock detection graph contains an edge $((e, f), c_P)$ if and only if there is an edge $g \in E$ with

$$f \in confl(R_P(g)) \quad \text{and} \quad e \in O_P(g). \tag{11}$$

Assume now 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. From Eq. 11 we obtain that there is an edge $e_{P_i} \in E$ for each path $P_i \in \mathcal{P} (i = 1, \ldots, m)$ such that

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

Therefore, the set of paths $\{P_1, \ldots, P_m\} \subseteq \mathcal{P}$ is deadlock-ridden (cf. Definition 3) since

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

Conversely, assume that there is a deadlock-ridden set of paths $P_1, \ldots, P_m \subseteq \mathcal{P}$, i.e., each path P_i contains an edge e_{P_i} such that

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

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

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

for some $j(i) \in \{1, \ldots, m\}$. Using Eq. 11 we obtain that, for all $i \in \{1, \ldots, m\}$, there exist an edge $((e_i, e_{j(i)}), c_{P_i})$ 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, the deadlock detection graph contains a colorful cycle.

Note that not every cycle constructed that way must contain an edge for each path in $P_1, \ldots, P_m$ since we do not demand in Definition 3 that a deadlock-ridden set of paths is inclusion minimal.

Deadlock prevention algorithm

Theorem 4 shows that the recognition of colorful cycles is the key ingredient of any deadlock prevention algorithm that uses the given model. So when we make reservations one by one, Theorem 4 tells us that we need to check whether a new reservation closes a colorful cycle in the deadlock detection graph. To this end, we consider the colorful path problem. Again, Alon et al. (1995) investigated a node variant of that problem.

Colorful path problem	
Instance	Directed graph $G = (V, E)$ with edges colored with colors from $\mathcal{C}$, a specific color $c \in \mathcal{C}$, target node $t \in V$, set of start nodes $S \subset V$
Task	Is there a colorful path from some $s \in S$ to t that does not use the 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 is known to be $\mathcal{NP}$-complete, see Garey and Johnson (1979), and can be reduced to this problem.

Path with forbidden pairs	
Instance	Directed graph $G = (V, E)$, start node and target node $s, t \in V$, collection $\mathcal{D} = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ of pairwise disjoint pairs of edges from E
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 *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 of a pair in 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 4 below solves the colorful path problem. The algorithm is related to the one introduced by Alon et al. (1995) for the corresponding node version.

Algorithm 4: 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 c.

Result: Is there a colorful s-t-path for some $s \in S$ that does not use the forbidden color 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), i) do
                if i ∉ C ∪ {c} and there is no label (u, C*) with C* = C then
                    add (u, C ∪ {c}) to Q_new ;
        Q_old = Q_new;
        Q_new = ∅;
    return false ;
end
```

It iteratively computes all colorful paths from length 1 to length $|\mathcal{C}| - 1$, or, more precisely, it maintains the information which nodes can be reached via a colorful path. To that end, we use node labels that contain 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 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. For the analysis of its run time we refer to Alon et al. (1995) since it is similar to the node version. Our additional consideration of a forbidden color and a set of start nodes (instead of a single node) does not change the analysis. The key observation in their proof is that the number of labels in each node after i iterations is bounded by $\binom{|\mathcal{C}|}{i}$.

Theorem 6 *Algorithm 4 solves the colorful path problem in $O(|\mathcal{C}| \cdot 2^{|\mathcal{C}|} \cdot |E|)$ time.*

Remark 2 (Additional heuristic) Because of the exponential run time of the algorithm we also developed a faster, heuristic version of Algorithm 4, which uses an upper bound on the size of the set Q_{new}. The algorithm is modified such that it returns *false* when this bound is reached.

Now we are able to formulate our deadlock prevention algorithm (Algorithm 5). Given a sequence of (static) paths, the algorithm computes a deadlock-free reservation schedule by iteratively adding these paths to the deadlock detection graph. This is done by Algorithm 6 (INSERT-ROUTE).

INSERT-ROUTE constructs a deadlock detection graph that contains no colorful cycle. This is done by calling Algorithm 4 in line CP. Simultaneously, a corresponding reservation schedule is constructed in line RS. By Theorem 4 we know that such a reservation schedule is deadlock-free. Therefore, we only have to argue that the deadlock detection graph is constructed correctly and that the reservation schedule is valid.

Algorithm 5: DEADLOCK-PREVENTION

Data: Directed graph $G = (V, E)$, sequence of 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 6);

end

Algorithm 6: INSERT-ROUTE

Data: Directed graph $G = (V, E)$, deadlock-free reservation schedule $S(\mathcal{P})$ and corresponding deadlock detection graph $G_D = (V_D, E_D)$, 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* $e_\ell \in \bigcup_{k=i+1}^{j}$ *confl*(e_k) *in* G_D **then**

RS $R_P(e_i) = \bigcup_{k=i+1}^{j} \{e_k\}$;

DG $\forall e_\ell \in \bigcup_{k=i+1}^{j}$ *confl*(e_k) add $((e_i, e_\ell), c_P)$ to E_D;

      $j = i$;

    $i = i - 1$;

end

Theorem 7 *The reservation schedule that is constructed by Algorithm* 5 *is valid and deadlock-free.*

Proof First, we observe that the constructed reservation schedule is valid. Due to Assumption 1, the first edge of the considered path is not contained in 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 (and the corresponding set of conflicting edges) 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. Correctness then follows from Theorem 4.

To show correctness of the construction, we observe that edges are requested in blocks (line RS of the algorithm). This means that, whenever we insert edges to a set $R_P(e_i)$, it is guaranteed that no edges e_k with $k > i$ are already (or will be) requested from an edge e_ℓ with $\ell < i$. Therefore, upon termination of the algorithm, each edge e_i of the given path fulfills

$$R_P(e_i) = \emptyset \quad \vee \quad O_P(e_i) = \{e_i\}. \tag{12}$$

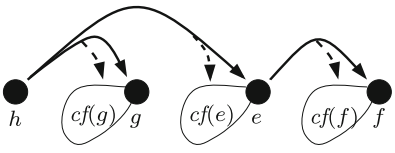
Thus, whenever there is an edge e_j with

$$e_k \in \mathit{confl}(R_P(e_j)) \quad \text{and} \quad e_i \in O_{P'}(e_j),$$

we have $e_k \in \mathit{confl}(R_P(e_i))$. Hence, by Definition 4, it is sufficient to insert those reservations to the deadlock detection graph that represent requests of edges in the corresponding reservation schedule (line RS), cf. Fig. 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), this completes the proof.

For the analysis of the run time we observe that Algorithm 4 is called at most $|P|$ times in Algorithm 6, where $|P|$ denotes the number of edges on that path.



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

Fig. 6 Construction of a reservation schedule (deadlock detection graph) for path $P = (h, g, e, f)$ by Algorithm 5. In contrast to the schedule illustrated in Fig. 5 only edges that directly result from the set of requested edges are inserted into the deadlock detection graph

Computational results

For our experiments we consider the HHLA CTA at Hamburg Harbor which is operated by the Hamburger Hafen und Logistik AG (HHLA). It is the most modern container terminal worldwide regarding the level of automation. In particular, the containers are transported between ship and storage area using AGVs. The AGVs are controlled centrally and do not have any kind of intelligent local-collision-avoidance system built in. The task to route them is thus an instance of our online disjoint vehicle routing problem.

We investigate a particular real-life scenario (SCEN-A) for the evaluation of the presented static routing approach (see Fig. 7a). In this scenario, 72 vehicles serve requests between 22 delivery points and 12 pick-up points in a bidirected grid-like graph.

The additional scenarios BL-A, 2/3L, and 1/3L, are created by excluding parts of the underlying graph, cf. Fig. 7. This is done to measure the performance under different traffic densities.

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 (Fig. 7b).
2/3L: SCEN-A with two thirds of the horizontal lanes (Fig. 7c).
1/3L: SCEN-A with one third of the horizontal lanes (Fig. 7d).

To measure the performance of our approach we investigate the average transit time of the determined paths, i.e., the time needed to serve a request on average. For every of these four scenarios, this average is taken over 10 runs with 6,000 routing requests each. Hence each number is an average of 60,000 container movements. These requests were also provided by HHLA and taken from real life data.

We also considered a comparison with the static routing algorithm by Kim et al. (2006), which is the most advanced based on zone control. However, we could not produce results for our size of instances with 72 AGVs within reasonable time.

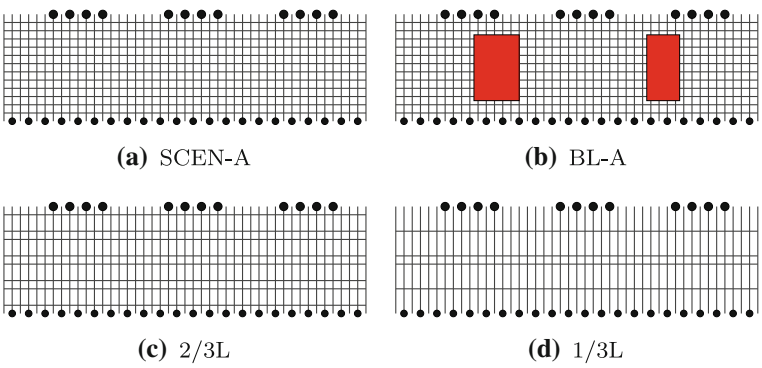


Fig. 7 Illustration of the scenarios investigated for the evaluation of the static routing approach. Besides the plain scenario SCEN-A we consider scenario BL-A with two blocked areas (in *red/dark gray*) and two scenarios 2/3L and 1/3L with reduced numbers of *horizontal lanes* in the *bidirected grid-like graph*

Their approach seems to need rather coarse zones and they report on computational experience with only 16 AGVs. In our approach, a zone is as small as an edge of the routing graph, or, more precisely, as the polygons induced be the AGVs as in Fig. 3.

Our main focus is on computation times.[1] Additionally, in order to analyze our load balancing approach (see "Online load balancing with bounded stretch factor"), we measure the (*static*) *length* of the computed paths and the *load* on the edges of the graph. Finally, we also report on the *number and length of the cycles* found by the deadlock detection algorithm.

Due to our initial evaluations described in Remark 3, we restricted the route computation by orienting the horizontal lanes alternatingly and computed the static paths in that modified directed graph. These orientations of horizontal lanes are also used at the HHLA CTA.

Remark 3 [Evaluations in the bidirected grid-like graph] It turned out that the static routing algorithm is not competitive for the bidirected grid-like graph SCEN-A. 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 when two vehicles want to pass a portion of the graph in opposite directions. Therefore at least one of these vehicles has to reserve to whole area. This leads to large reserved areas that makes deadlock detection computationally expensive.

The evaluation is divided into two parts. We first analyze the performance under variation of the stretch factor B, cf. "Online load balancing with bounded stretch factor", in SCEN-A. Then we report on the impact of the traffic intensity on the performance using the three additional scenarios.

Variation of the stretch factor

In "Online load balancing with bounded stretch factor" we presented Algorithm 2 (BAL-BOUND) for the route computation in our static routing approach. The cost function of the algorithm depends on the given length constraint on the determined paths, i.e., on the stretch factor B. Table 1 illustrates the evaluation of the performance under variation of B.

It turns out that the results are very similar for a stretch factor greater than 1. The differences are only minor. In contrast, simply computing a static shortest path for each request ($B = 1$) 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, as there are many more detected cycles. These, in turn, generate larger reserved areas, since each detected cycle requires an earlier reservation. Thus it is not surprising that the average transit time is longer 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 routing approach definitely improves the performance of the routing algorithm. But, at least

[1] Hardware: Intel Pentium 4 2,8 GHz with 1024 MB RAM.

Table 1 Evaluation of the static routing approach for different stretch factors B

B	Average duration (s)	average path length (m)	Max. load	Cycle length		# cycles per request	Comp. time	
				Avg.	Max.		Avg. (s)	max. (s)
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

in the considered bidirected grid-like graph, the stretch factor does not play an important role.

Since the smallest average transit time is achieved with a stretch factor $B = 1.2$, we choose this setting for the evaluations in "Comparison with the dynamic routing approach".

Comparison with the dynamic routing approach

Now, we focus on the question which routing approach—the static one or the dynamic one introduced by Gawrilow et al. (2008)—performs better with respect to the average transit time. Again, these averages are taken over 10 runs with 6,000 container movements each.

The results of Table 2 show that the average transit time highly depends on the traffic density. While the static router shows a slightly smaller average transit time than the dynamic router in scenarios SCEN-A and 2/3L with low traffic volume, it is clearly inferior in scenarios with high traffic volume. The static router performs particularly bad in the most narrow scenario 1/3L, while the average transit measured for the dynamic router is not much higher than in the other scenarios. Moreover, contrary to the dynamic router, the computation times of the static router increase with the traffic density.

The reason for the bad performance of the static routing approach in scenarios with high traffic volume becomes clear if we consider the behavior of the deadlock detection shown in Table 3 and keep in mind that the deadlock prevention algorithm has an exponential run time. The number and the length of the detected cycles increase with the traffic density, which leads to large computation times. For these computations we used the heuristic mentioned in Remark 2 and set the upper bound to 500 in order to keep computation times small. The number of cases where this bound is reached also increases and becomes very large in scenario 1/3L. We also tried to evaluate that instance without using the heuristic, but the performance was even worse since the system almost stalled from time to time due to large computation times for a single reservation (more than 60 s). The length of the reservations obviously increases with the number of found cycles and the number of canceled searches for a colorful path in the heuristic, respectively. It is not surprising that this leads to a loss of performance.

Table 2 Comparison of the static routing approach with the dynamic routing approach with respect to average duration (transit time) and computation time

	Static approach			Dynamic approach		
	Average duration (s)	Computation time		Average duration (s)	Computation time	
		Avg. (s)	Max. (s)		Avg. (s)	Max. (s)
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 3 Evaluation of the deadlock detection algorithm with respect to different traffic densities

	Cycle length		# cycles per request	# upper bound reached per request
	Average	Maximum		
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

But why does the static approach perform better in scenarios with low traffic density? The reason for this, perhaps surprising, result is the greedy reservation strategy used in the static router. The next portion of the route is reserved as soon as possible disregarding that this may interfere other vehicles, cf. Fig. 8. In contrast,

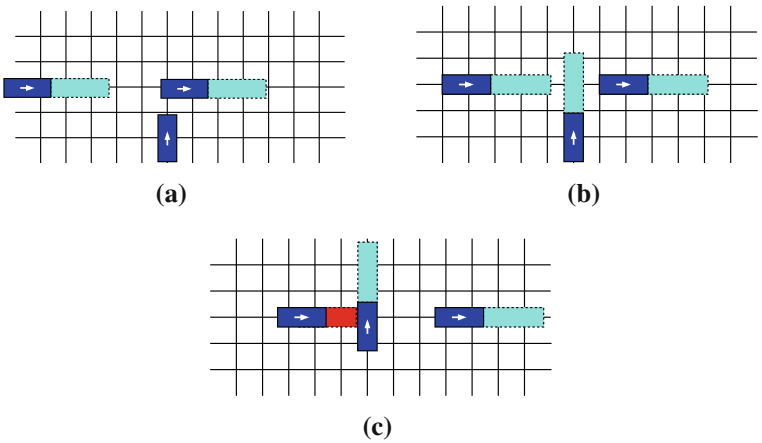


Fig. 8 Illustration of the greedy reservation procedure used in the static routing approach. Due to the greedy reservation procedure each small time window is used

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 area.

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. The greedy reservation procedure pays off in this case. But when the reserved areas become large because of a more complicated deadlock prevention, the static router becomes inferior.

An evaluation of our approach based in a simulation system developed for the HHLA CTA (Gawrilow et al. 2008) has shown that it 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.

Acknowledgments We thank the referees for their valuable comments and for pointing out a flaw in the proof of Theorem 1.

References

Alon N, Yuster R, Zwick U (1995) Color-coding. J ACM 42(4):844–856

Aspnes J, Azar Y, Fiat A, Plotkin S, Waarts O (1997) On-line routing of virtual circuits with applications to load balancing and machine scheduling. J ACM 44:486–504

Azar Y, Noar J, Rom R (1992) The competitiveness of on-line assignment. In: Proceedings of 3rd ACM-SIAM symposium on theory of computing

Borodin A, El-Yaniv R (1998) Online computation and competitive analysis, Cambridge University Press, London

Cho HB, Kumaran TK, Wysk RA (1995) Graph theoretic deadlock detection and resolution for flexible manufacturing systems. IEEE Trans Robot Autom 11(3):413–421

Gao J, Zhang L (2004) Tradeoffs between stretch factor and load balancing ratio in routing on growth restricted graphs. In: Proceedings of the twenty-third annual ACM symposium on principles of distributed computing, pp 189–196

Garey MR, Johnson DS (1979) Computers and intractability, a guide to the theory of NP-completeness, W. H. Freeman and Company, USA

Gawrilow E, Köhler E, Möhring RH, Stenzel B (2008) Dynamic routing of automated guided vehicles in real-time. In: Jäger W, Krebs HJ (eds) Mathematics—key technology for the future. Joint projects between universities and industry 2004−2007, Springer, Berlin, pp 165–178

Guan WH, Moorthy KMRL (2000) Deadlock prediction and avoidance in an AGV system. SMA Thesis

Kim KH, Jeon SM, Ryu KR (2006) Deadlock prevention for automated guided vehicles in automated container terminals. OR Spectr 28(4):659–679

Lee CC, Lin JT (1995) Deadlock prediction and avoidance based on Petri nets for zone control. Int J Prod Res 33(12):3239–3265

Seiden S, Sgall J, Woeginger GJ (2000) Semi-online scheduling with decreasing job sizes. Oper Res Lett 27(5):215–221

Vis IFA (2006) Survey of research in the design and control of automated guided vehicle systems. Eur J Oper Res 170:677–709

Wu NQ, Zhou MC (2000) Resource-oriented Petri nets for deadlock avoidance in automated manufacturing. In: Proceedings of 2000 IEEE international conference on robotics and automation, pp 3377–3382

Yeh MS, Yeh WC (1998) Deadlock prediction and avoidance for zone-control agvs. Int J Prod Res 36(10):2879–2889