Conflict-free Vehicle Routing: Load Balancing and Deadlock Prevention

Max Klimm, Ewgenij Gawrilow, Rolf H. Möhring, and Björn Stenzel*

Technische Universität Berlin, Institut für Mathematik, MA 5-1,
Straße des 17. Juni 136, 10623 Berlin, Germany
{klimm,gawrilow,moehring,stenzel}@math.tu-berlin.de

Abstract. We study a so-called static approach for the problem of routing vehicles conflict-free through a given street network. In fact, we assume that routes are computed without taking time-dependences into account and collisions are avoided via a particular reservation procedure. In this context, the task is to cope with two arising problems: the appearance of congestion and detours on the one hand and the risk of deadlocks on the other.
We provide a two-stage routing approach for that problem. In the first phase we focus on balancing the load on the edges of the given graph that models the underlying street network. Therefore, we consider the Online Load Balancing Problem with Bounded Stretch Factor and give an optimal algorithm with respect to a specific performance ratio, the stretch factor restricted competitive ratio. Furthermore, in a second phase, we investigate the detection and avoidance of deadlock situations.
For the evaluation of the entire algorithm we consider the routing of Automated Guided Vehicles (AGVs) at HHLA Container Terminal Altenwerder (CTA).

1 Introduction

We consider an undirected graph $G = (V, E)$. The edge set E denotes the lanes of the underlying traffic network. Each edge $e \in E$ has a certain transit time $\tau(e)$ which indicates the time needed to traverse this edge. The node set V models the crossings of the lanes.

Transportation tasks are consecutively arriving over time and are modeled by a sequence $\sigma = r_1, \ldots, r_k$ of requests. Each request $r_i = (s_i, t_i)$ consists of a start node s_i and an end node t_i. Note that we do consider the assignment of vehicles to requests. In contrast, we assume that this is done by a higher-level management system in advance.

*supported by the DFG Research Center MATHEON "Mathematics for key technologies"

In a static approach for online disjoint vehicle routing problems one computes static paths 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 $\tau(e)$ plus a load dependent penalty cost which is a function of the number of routes that are already using this edge.

In such a routing approach the computed paths are, of course, not conflict-free. Hence, one needs an additional conflict management that, at execution time of the routes, guarantees that no collisions occur. This can be done by iteratively allocating to a vehicle the next part of its route (the *reserved area*) and block it for all other vehicles (*reservation*), see Figure 1.

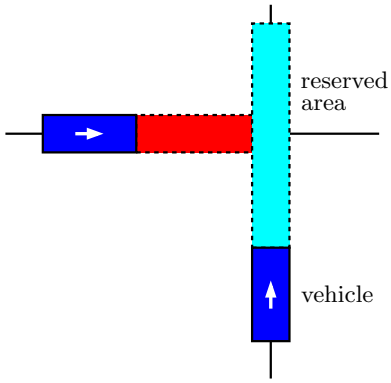


Fig. 1. Reservation procedure. Each vehicle reserves the next part of the route. Mutual exclusive reservations guarantee a collision-free execution of computed (static) paths.

In comparison to so-called dynamic routing approaches (see [8]) such an approach a priori shows various drawbacks. The most alarming one is caused by the collision avoidance at execution time. The reservation rules can cause *deadlocks*, as illustrated in Figure 2, which have a deteriorating effect on the system performance. Deadlocks appear if a group of vehicles wish to reserve a set of edges which are already occupied by another vehicle in this group such that none of them is able to continue its route and thus the system is blocked. Note that the dynamic routing approach provides deadlock-free routes already at the time of the route computation.

In addition to deadlocks, another inbuilt drawback in the static setting is the appearance of *detours* and *high congestion*. This results in traveling times that can be far away from the shortest possible traveling time.

Previous Work. So far only Guan and Moorthy [9] investigated different kinds of penalty costs for this kind of vehicle routing problem. In fact, they considered constant additive penalty costs on the one hand and distance-dependent costs on the other. But, besides they do not give any theoretical performance guarantee for these approaches, the experimental evaluation does not show any major effect.

The detection and prevention of deadlocks has been investigated by Lee and Lin [11] who considered petri net approaches. Additionally, there are graph theoretic models used [5, 9, 10, 14]. For details concerning these approaches we refer to Section 3.

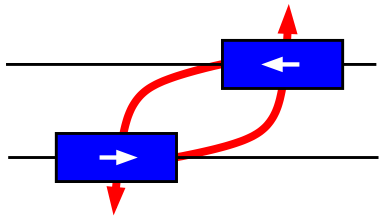


Fig. 2. Simplified deadlock situation. Both vehicles are trying to occupy the same portion/edges of the network, thereby blocking each other.

Our Approach. In order to cope with the problem of congestion and detours we use a sophisticated routing approach that balances the load on the edges of the graph such that the resulting paths remain short with respect to the transit times. In fact, in Section 2, we present an online load-balancing algorithm that guarantees a minimal load under a given length constraint to the chosen paths.

In a second step we construct so-called reservation schedules to avoid potential deadlock situations. The corresponding a deadlock detection and prevention algorithm will be introduced in Section 3.

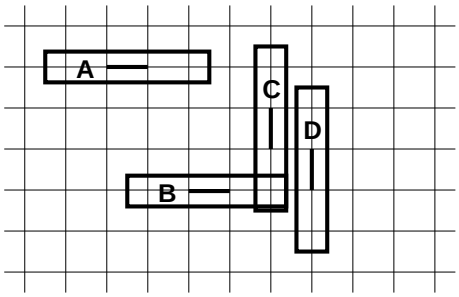


Fig. 3. The figure illustrates the polygons that are claimed by a vehicle that moves on the indicated edge. Polygon B and C intersect each other while polygons A and D do not intersect another polygon.

For dealing with the physical dimensions of the vehicles we use polygons $\mathcal{P}(e)$ for each edge e, which describe the blocked area when a vehicle (the center of a vehicle) is located on edge e (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.

Algorithm 1: STAT-ROUTE

Data: Directed graph $G = (V, E)$, sequence of requests $\sigma = r_1, \ldots, r_k$.
Result: Sequence if 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 Section 2.2);
    compute a deadlock-free reservation schedule (see Section 3.4);
end

2 Online Load Balancing with Bounded Stretch Factor

2.1 Introduction

In recent years many people did research into the Online Load Balancing Problem [2, 4]: Consider a directed graph $G = (V, E)$ and a sequence of requests $\sigma = (r_1 = (s_1, t_1), \ldots, r_k = (s_k, t_k))$, where s_i and t_i denote the source and target node of the request i, respectively. Sometimes a certain bandwidth b_i, that possibly also depend on the edges used, is assigned to each request. We will focus on the case where this bandwidth is equal to one. In this case, the load on an edge e after the i-th request ($\text{load}_i(e)$) is defined as the number of requests already routed over e. The task is to minimize the maximum load over all edges, i.e., $\min \max_{e \in E} \text{load}_k(e)$, whereas k again denotes the 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. [2] gave an $O(\log(|E|))$-competitive algorithm for that problem and showed, using the lower bound of Azar, Naor and Rom [3] for online assignment, that their approach is optimal for online load balancing.

This standard load balancing problem has been extended to the case with transit times $\tau(e)$ on edges and a certain constraint to the length of a chosen path by Gao and Zhang [6]. In fact, they introduced a so-called stretch factor $B > 1$ that bounds the length of a chosen s_i-t_i-path, i.e., each s_i-t_i-path has to be shorter than B times the length of a shortest path between s_i and t_i. We call this problem the Online Load Balancing with Bounded Stretch Factor.

Online Load Balancing Problem with Bounded Stretch Factor	
Instance:	Directed graph $G = (V, E)$, transit times $\tau : E \to \mathbb{R}$, stretch factor B, sequence of requests $\sigma = (r_1, \ldots, r_k)$.
Task:	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, whereas $\text{length}(\text{SP}_i)$ denotes the length of a shortest path between s_i and t_i.

Gao and Zhang [6] modified the approach of Aspnes et al. [2] and obtained similar results concerning the competitive ratio for this problem. In particular, they also provide a $O(\log(|E|))$-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 we focus on a different kind of analysis. Instead of comparing the solution of a particular online algorithm with the optimal solution for that problem (competitive analysis), we consider the optimal solution of the standard load balancing problem, i.e., without the given stretch factor constraint. We are interested in this ratio since an optimal load balancing solution can be seen as the best choice with respect to generated congestion in our static routing algorithm STAT-ROUTE (Algorithm 1). We refer to it as the *stretch factor restricted (sfr) competitive ratio* and transfer the notation from the standard competitive analysis introduced in [4].

Definition 1. *An online algorithm* ALG *for the Online Load Balancing Problem with Bounded Stretch Factor is called* c-stretch-factor-restricted-competitive *or* c-sfr-competitive *if, for any problem instance* $\mathcal{I}$*, it achieves a solution*

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

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

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

In addition, we assume that the number of requests k, or at least a good upper bound, is given in advance. Seiden, Sgall and Woeginger [12] call such approaches semi-online. In our application, the disjoint vehicle routing, a good upper bound might be the number of vehicles since there can not be more 'active' requests than vehicles.

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

2.2 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 to the optimal load with respect to the already routed requests in this phase. This upper bound is adjusted dependent on the current maximum load produced by the algorithm, i.e., whenever it can not be guaranteed that UB is still an upper bound, cf. Theorem 1, we enter a new phase (and double the bound).

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$(e) = 0\ \forall e \in E$;
  UB $= 1$;
  $b = \sqrt[k]{B}$;
  foreach $r_i = (s_i, t_i)$ **do**
SP compute a shortest s_i-t_i path P_i w.r.t. the cost function
    $c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$;
    if $\exists e \in \mathit{confl}(P_i)$: load$(e) > \log_b(b^2 k \cdot |E| \cdot \frac{\max_{e\in E} \tau(e)}{\min_{e\in E} \tau(e)}) \cdot$ UB **then**
      UB $= 2 \cdot$ UB;
      load$(e) = 0\ \forall e \in E$;
      goto line SP ;
    else
      load$(e) =$ load$(e) + 1\ \forall e \in \mathit{confl}(P_i)$;
      assign path P_i to request r_i;
end

For each request the algorithm computes a shortest path with respect to the cost function $c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$, where b is the k-th root of the given stretch factor B. Afterwards the load on all edges that conflict with an edge of the selected path P, namely the edges in $\mathit{confl}(P) := \bigcup_{e\in P} \mathit{confl}(e)$, is increased by one.

In order to show the claimed stretch factor restricted competitive ratio of Algorithm 2 (BAL-BOUND) we introduce Algorithm 3 (SUB-BAL-BOUND) which can be viewed as subroutine (phase) of Algorithm 2 since there we assume that an upper bound UB is given in advance.

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.
Result: Sequence of static paths $P_1, \ldots, P_k$.
begin
  $\text{load}(e) = 0 \ \forall e \in E$;
  $b = \sqrt[k]{B}$;
  foreach $r_i = (s_i, t_i)$ **do**
    compute a shortest s_i-t_i path P_i w.r.t. the cost function
    $c(e) = \tau(e) \cdot b^{\frac{\text{load}(e)}{\text{UB}}}$;
    $\text{load}(e) = \text{load}(e) + 1 \ \forall e \in \mathit{confl}(P_i)$;
    assign path P_i to request r_i;
end

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

Theorem 1. *For a fixed upper bound $UB \geq OPT$ and for any problem instance $\mathcal{I}$ it holds that*

$$\text{SUB-BAL-BOUND}(\mathcal{I}) \leq \log_b \left(b^2 k \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)} \right) \cdot \text{UB}.$$

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

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

for all i. Summing up over all requests leads to the following inequality:

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

Now we multiply both sides by $b-1$. Then, for the inner sum of the left hand side, we get

$$(b-1) \cdot \sum_{i : e \in P_i} \tau(e) b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}} = \sum_{i : e \in P_i} \tau(e) \cdot (b^{\frac{\text{load}_{i-1}(e)}{\text{UB}} + 1} - b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}}) \quad (1)$$

$$\geq \sum_{i : e \in P_i} \tau(e) \cdot (b^{\frac{\text{load}_{i}(e)}{\text{UB}}} - b^{\frac{\text{load}_{i-1}(e)}{\text{UB}}}) \quad (2)$$

$$= \tau(e) \cdot (b^{\frac{\text{load}_{k(e)}}{\text{UB}}} - 1). \tag{3}$$

Here, the telescope sum in (2) is obtained by the observation that the load on an edge e increases at most by one in a single step ($\text{load}_i(e) \leq \text{load}_{i-1}(e)+1$). Additionally, note that $\text{UB} \geq 1$ holds in each phase of the algorithm.

On the right hand side we get

$$(b-1) \cdot \sum_{e \in E} \sum_{i: e \in P_i^*} \tau(e) b^{\frac{\text{load}_{i-1(e)}}{\text{UB}}} \leq b(b-1) \cdot \sum_{e \in E} \sum_{i: e \in P_i^*} \tau(e) \tag{4}$$

$$\leq b(b-1)k \cdot \sum_{e \in E} \tau(e). \tag{5}$$

Equation (4) holds due to fact that, in an optimal solution, the load on a certain edge never exceeds the given upper bound ($\frac{\text{load}_{i-1}(e)}{\text{UB}} \leq 1 \quad \forall i$). Additionally, since there must be an optimal solution that only contains simple paths (positive cost function), the number of paths that are routed over a certain edge is bounded by k which leads to (5).

Using the results from both sides we get the claimed performance guarantee by simple arithmetic transformations:

$$\begin{aligned}
&\textstyle\sum_{e \in E} \tau(e)(b^{\frac{\text{load}_{k(e)}}{\text{UB}}} - 1) && \leq b(b-1)k \cdot \textstyle\sum_{e \in E} \tau(e) \\
\Leftrightarrow\ &\textstyle\sum_{e \in E} \tau(e) b^{\frac{\text{load}_{k(e)}}{\text{UB}}} && \leq (b(b-1)k+1) \cdot \textstyle\sum_{e \in E} \tau(e) \\
\Rightarrow\ &\textstyle\sum_{e \in E} \tau(e) b^{\frac{\text{load}_{k(e)}}{\text{UB}}} && \leq b^2 k \cdot \textstyle\sum_{e \in E} \tau(e) \\
\Rightarrow\ &\min_{e \in E} \tau(e) \textstyle\sum_{e \in E} b^{\frac{\text{load}_{k(e)}}{\text{UB}}} && \leq b^2 k \cdot |E| \cdot \max_{e \in E} \tau(e) \\
\Leftrightarrow\ &\textstyle\sum_{e \in E} b^{\frac{\text{load}_{k(e)}}{\text{UB}}} && \leq b^2 k \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)} \\
\Rightarrow\ &\max_{e \in E} b^{\frac{\text{load}_{k(e)}}{\text{UB}}} && \leq b^2 k \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)} \\
\Leftrightarrow\ &\max_{e \in E} \text{load}_k(e) && \leq \log_b(b^2 k \cdot |E| \cdot \tfrac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)}) \cdot \text{UB}.
\end{aligned}$$

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

Theorem 2. BAL-BOUND *is* $4 \cdot \log_b(b^2 k \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)})$*-sfr-competitive. Thus, the stretch factor restricted competitive ratio is in*

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

Proof. Due to readability we write $\text{SUB-BAL-BOUND}_{\text{UB}}$ if we consider Algorithm 3 with given upper bound UB and abbreviate the ratio guaranteed of

SUB-BAL-BOUND by $c := \log_b(b^2 k \cdot |E| \cdot \frac{\max_{e \in E} \tau(e)}{\min_{e \in E} \tau(e)})$. Moreover, recall that SUB-BAL-BOUND can be viewed as subroutine of BAL-BOUND.

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

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

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

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

By Theorem 1 it follows that

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

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

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

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

Theorem 3. *Each path selected by BAL-BOUND is shorter (with respect to τ) than B times the shortest path length.*

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

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

Since the cost function is positive, the static shortest path will never contain a cycle. Thus, it holds that $\text{load}_{i-1}(e) < i$ for all i. This leads to the following estimate for each i:

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

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

$$\text{length}(P_i) < B \cdot \text{length}(\text{SP}_i) \quad \forall i.$$

2.3 Lower Bound

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

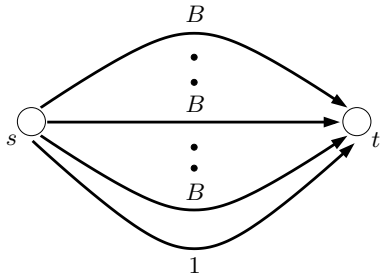


Fig. 4. Graph described in Example 1.

Example 1. We consider an instance of the Load Balancing Problem with Bounded Stretch Factor with stretch factor B and k requests from s to t in the graph illustrated in Figure 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 one edge (transit time 1) can be used since routing over the other edges would violate the stretch factor constraint. Thus, since in an optimal solution without this constraint, one would assign each request to an exclusive edge, which leads to a load of 1, the sfr competitive ratio is bounded from below by

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

for any online algorithm for the Online Load Balancing Problem with Bounded Stretch Factor.

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

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

3 Reservation Schedules and Deadlock Prevention

3.1 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 need a

mechanism to definitely avoid conflicts. In Section 3.2 we introduce a reservation schedule that prevents the vehicles from colliding by requesting edges before occupying them. Such a schedule can be interpreted as an instruction for the construction of a dynamic path at execution time.

Since this procedure may cause deadlocks, we additionally have to take care for such situations. Due to the deteriorating effects of deadlocks to logistic processes a lot of attention is payed to that research field the recent years. Besides the investigation of petri net approaches [11, 13] there are also graph theoretic models used [5, 9, 10, 14].

The first observation concerning most of these approaches is that they use so-called zone control, i.e., it is assumed that vehicles move between non-intersecting zones of adequate size. This, obviously, is not a suitable model for our purpose since it is not possible to partition our traffic network into such zones. Kim et al. [10] introduce a finer zoning of the network, but, just as the zone control approaches, their algorithm is not able to deal with large-scale vehicle fleets.

Our deadlock prevention algorithm provides both, a fine discretization (vehicles move on 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 [5, 10, 14], in a graph that is in a one-to-one correspondence with the considered reservation schedule, the so-called deadlock detection graph. Moreover, in contrast to Guan and Moorthy [9] we avoid deadlocks already at the time of the route computation and not during the execution of the route.

The section is structured as follows. After the description of our model we introduce the deadlock detection graph in Section 3.3 and present the deadlock prevention algorithm in Section 3.4. Computational results and conclusions are given afterwards.

3.2 The Model

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

To guarantee a smooth execution of a reservation schedule, especially no occurrence of conflicts, it is necessary that each edge on a certain path have to 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 for each path $P = (e_1, \ldots, e_n) \in \mathcal{P}$ holds:*

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

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

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

A reserved edge is freed when the vehicle leaves that edge. To this end, we define another edge set, the *occupied edges* $O_P(e_i)$. An edge is called occupied if it has already been reserved and has not been freed, i.e.,

$$O_P(e_i) = e_i \cup \left(\bigcup_{k<i} \{e_j \in R_P(e_k) |\ j > i\} \right). \tag{6}$$

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 also have to be requested. But note that those edges do not have to be reserved and are therefore not occupied after the corresponding reservation. To this end, we will always distinguish between requested edges in $R_P(e)$ and those in $\mathit{confl}(R_P(e))$.

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

Definition 3. *Let $S(\mathcal{P})$ be a reservation schedule. Then, a set of paths $\{P_1, \ldots, P_m\} \subseteq \mathcal{P}$ is called* deadlock-ridden *if, for each $i \in \{1, \ldots, m\}$, there exists an edge e_{P_i} such that*

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

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

Based on the described model we will now concentrate on the detection and prevention of deadlocks. To this end, we introduce a so-called deadlock detection graph that represent a reservation schedule in Section 3.3 and give a deadlock prevention algorithm that makes use of this correlation in Section 3.4.

3.3 Deadlock Detection Graph

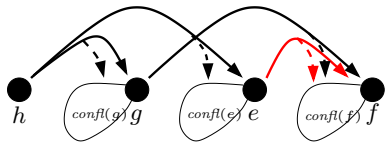
The question we will answer in this section is how we can detect a potential deadlock, i.e., a deadlock-ridden set of paths, in a given reservation schedule. We will show that this can be interpreted as a cycle with a specific property in a special kind of graph.

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

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

1. $V_D = E$.
2. $E_D = \{((e, f), c_P) |\ \exists g \in E$ such that $f \in \mathit{confl}(R_P(g))$ and $e \in O_P(g)\}$.

Figure 5 illustrates the construction of a deadlock detection graph from a reservation schedule $S(\mathcal{P})$, i.e., from a collection of sets of requested edges. Obviously, since by definition for each edge $e \in P$ on a certain path P holds $e \in O_P(e)$, each requested edge $f \in R_P(e)$ in a reservation schedule leads to an edge $((e, f), c_P)$ in the reservation graph. However, note that there are additional edges added to the graph.



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) of a single path $P = (h, g, e, f)$ While the black edges directly result from the set of requested edges, the red edges are due to the fact that edge e is in $O_P(g)$ and f is requested from g, respectively.

Since we are able to construct a deadlock detection graph from an arbitrary reservation schedule we are going to use this representation to detect potential deadlocks. To this end, we introduce so-called colorful paths and cycles according to [1] where Alon, Yuster, and Zwick considered the corresponding node version.

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 all edges on P are colored different. A* colorful cycle *is a cycle with the same property.*

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

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

$$f \in \mathit{confl}(R_P(g)) \quad \text{and} \quad e \in O_P(g). \tag{7}$$

Firstly, to see necessity, we assume that there is a colorful cycle $((e_1, e_2), c_{P_1}), \ldots, ((e_m, e_{m+1} = e_1), c_{P_m})$ in the deadlock detection graph. From Eq. 7 we get that there is an edge $e_{P_i} \in E$ for each path $P_i \in \mathcal{P}$ $(i = 1, \ldots, m)$ such that

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

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

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

To see sufficiency, assume that there is a deadlock-ridden set of paths $P_1, \ldots, P_m \subset \mathcal{P}$, i.e., there is an edge e_{P_i} such that

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

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

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

for some $j(i) \in \{1, \ldots, m\}$. Using Eq. 7 we get that there must be an edge $((e_i, e_{j(i)}), c_{P_i})$ for all $i \in \{1, \ldots, m\}$ and a corresponding $j(i) \in \{1, \ldots, m\}$ in the deadlock detection graph since

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

Thus, there must be a colorful cycle in that graph. Note that not any cycle that is 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.

3.4 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. More precisely, one has to check whether a new reservation closes a colorful cycle in the deadlock detection graph. To this end, we consider the Colorful Path Problem. Note that, as mentioned before, Alon, Yuster, and Zwick [1] investigated a node variant of that problem.

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

The Colorful Path Problem is $\mathcal{NP}$-complete since the edge version of the Path with Forbidden Pairs Problem, which is known to be $\mathcal{NP}$-complete [7], can be reduced to this problem.

Path with Forbidden Pairs [7]	
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 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 solves the Colorful Path Problem. The algorithm is related to the one introduced by Alon, Yuster and Zwick [1] for the corresponding node version.

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. Therefore, each label consists of node and a collection of colors C that contains the colors used on the corresponding path. Note that we do not propagate labels with redundant information, i.e., we do not add the same label again. This can be guaranteed by a look up in a table that provides for

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
```

each possible combination of colors the information whether this combination is already 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 start 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 does not contain the forbidden color and therefore solves the Colorful Path Problem. For the analysis of the run time we refer to [1] since it is similar to the node version. In particular, the additional consideration of a forbidden color and a set of start nodes (instead of a single node) does note 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* $O(|\mathcal{C}| \cdot 2^{|\mathcal{C}|} \cdot |E|)$.

Remark 2 (Additional heuristic). Due to the exponential run time of the algorithm we provide a heuristic for Algorithm 4. In fact, we introduce an upper bound to the size of the set Q_{new}. The algorithm is modified such that it returns *true* whenever the upper 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 inserting these paths, which is done by Algorithm 6 (INSERT-ROUTE).

The basic concept of INSERT-ROUTE is to provide 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. 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 to obtain Theorem 7.

Algorithm 5: DEADLOCK-PREVENTION

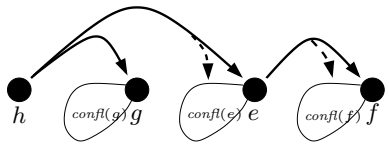
Data: Directed graph $G = (V, E)$, sequence of paths $\mathcal{P} = P_1, \ldots, P_k$.
Result: Deadlock-free reservation schedule $S(\mathcal{P})$.
begin
  foreach path P_i **do**
    INSERT-ROUTE (Algorithm 6);
end

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. Firstly, we observe that the constructed reservation schedule is valid. Due to Assumption 1, the first edge of the considered path does not lie on a colorful



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 in the deadlock detection graph.

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 an preceeding edge.

To obtain that the constructed reservation schedule is deadlock-free it is sufficient to proof that the deadlock detection graph is constructed correctly, i.e., to show that it corresponds to the computed reservation schedule, since we can apply Theorem 4 in this case. To see this, we observe that edges are requested in blocks (line RS of the algorithm), that is, 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 $\ell < i$ during the algorithm. Therefore, by termination of the algorithm, for each edge e_i of the given path holds

$$R_P(e_i) = \emptyset \quad \vee \quad O_P(e_i) = \{e_i\}. \tag{8}$$

Thus, whenever there is an edge e_j with

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

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

4 Computational Results

For our experiments we consider the HHLA Container Terminal Altenwerder (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 so-called Automated Guided Vehicles (AGVs).

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

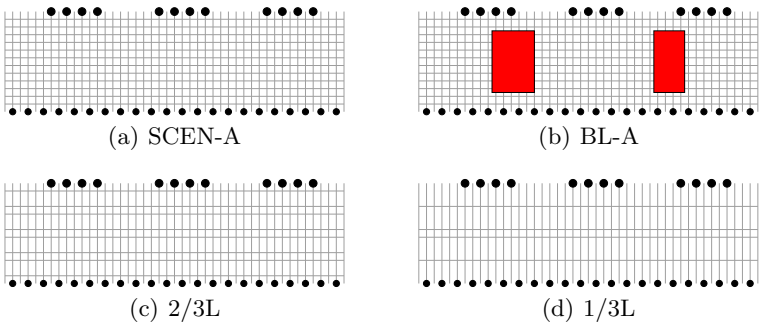


Fig. 7. Illustration of the scenarios investigated for the evaluation of the static routing approach. Besides the plain scenario SCEN-A we consider scenario BL-A with two blocked areas and two scenarios with reduced number of horizontal lanes in the grid-like graph, namely scenario 2/3L and scenario 1/3L.

Additional scenarios, namely BL-A, 2/3L, and 1/3L, are created by excluding parts of the underlying graph, cf. Figure 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 (Figure 7(b)).
2/3L: SCEN-A with two thirds of the horizontal lanes (Figure 7(c)).
1/3L: SCEN-A with one third of the horizontal lanes (Figure 7(d)).

To measure the performance of our approach we investigate the *average duration* of the determined paths, i.e., the time needed to serve a request on average. Moreover, we consder the *computation times*.* Additionally, in order to analyze the presented load balancing approach (see Section 2), we focus on the *(static) length* of the computed paths and the *load* on the edges of the graph. Moreover, we evaluate the *number and the length of the cycles* found by the deadlock detection algorithm.

*Hardware: Intel Pentium 4 2,8 GHz with 1024 MB RAM.

Due to the results of the first evaluations described in Remark 3 we restrict the route computation. In fact, we apply directions to the horizontal lanes alternatingly and compute the static path in that modified graph.

Remark 3 (Evaluations in the undirected grid-like graph). It turned out that the static routing algorithm is not competitive in the considered (undirected) grid-like graph. In fact, the systems almost stalls and therefore we omit detailed evaluations.

The reason for the bad performance is the frequent appearance of so-called head-to-head conflicts if two vehicles want to pass a portion of the graph in opposite directions. Therefore at least one of these vehicles has to reserve to whole area together. Besides this might lead to enormous reserved areas the deadlock detection becomes computationally expensive.

The evaluation is divided into two parts. Firstly, we are going to analyze the performance under variation of the stretch factor B, cf. Section 2, in SCEN-A. Afterwards we take a look at the impact of the traffic intensity to the performance using the three additional scenarios.

4.1 Variation of the Stretch Factor

In Section 2 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 to the determined paths, the stretch factor B. Tabular 1 illustrates the evaluation of the performance under variation of that value.

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

Table 1. Evaluation of the static routing approach with respect to different stretch factors B.

It turns out that the results of the experiments with stretch factor greater than 1 are very similar. In fact, they show only minor differences. 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, namely there are much more detected

cycles which generates larger reserved areas since each detected cycle makes an earlier reservation necessary. Thus, it is not surprising that the average duration 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 dynamic routing approach makes sense, but, at least in the considered grid-like graph, the stretch factor does not play an important role.

Since the least average duration is achieved with stretch factor $B = 1.2$, we choose this setting for the evaluations in Section 4.2.

4.2 Comparison with the Dynamic Routing Approach

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

Table 2. Comparison of the static routing approach with the dynamic routing approach with respect to the average duration and the computation times.

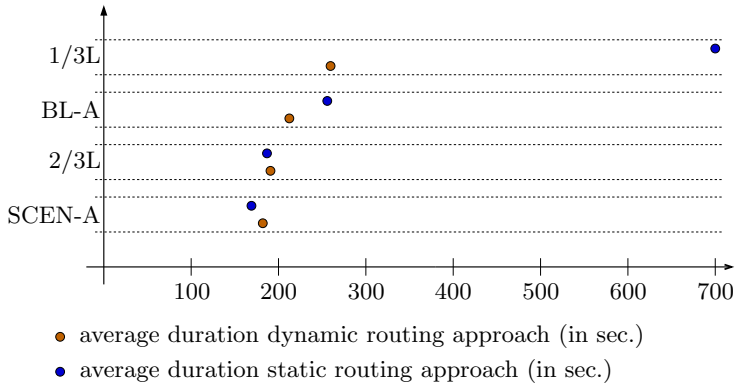


Fig. 8. Illustration of the performance of the static routing approach in comparison to the dynamic routing approach with respect to the measured average duration.

Now, we focus on the question which routing approach, the static one or the dynamic one introduced in [8] performs better with respect to the chosen objective, the average duration.

As illustrated in Tabular 2 and Figure 8, this can not be answered generally. It highly depends on the traffic density. While the static approach shows a slightly lower average duration than the dynamic one in the scenarios with a comparatively low traffic volume, namely the plain scenario SCEN-A and scenario 2/3L, it has problems in scenarios with high traffic volume. Especially in the most narrow scenario 1/3L the static approach performs very bad while the average duration measured for the dynamic approach is not that high compared with the other scenarios. Moreover, in the static approach, the computation times increase with the traffic density which is not the case in the dynamic approach.

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

Table 3. Evaluation of the deadlock detection algorithm with respect to different traffic densities.

The reason for the bad performance of the static routing approach in scenarios with high traffic volume becomes clear if we regard the evaluation of the deadlock detection shown in Tabular 3 and keep in mind that the deadlock prevention algorithm has an exponential run time. The number and length of the detected cycles increase with the traffic density which leads to large computation times. To this end, we used the heuristic mentioned in Remark 2 and set the upper bound to 500 in order to provide suitable computation times. The number of cases where this bound is reached also increases and becomes immense in scenario 1/3L. Note that we also tried to evaluate that instance without using the heuristic, but the performance was even worse since the system almost stalls. The length of the reservations, obviously, increases with the number of found cycles and the number of canceled searches for a colorful path (heuristic), respectively. It is not surprising that this leads to a loss of performance.

But why does the static approach perform better in scenarios with low traffic density? The cause for this, maybe surprising, result is the greedy reservation strategy used in this case. The next portion of the route is reserved as soon as possible disregarding that this may interfere other vehicles, cf. Figure 9. In contrast, the dynamic routing algorithm does not make use of such gaps since the reservations are made before the vehicles start traveling and it is forbidden

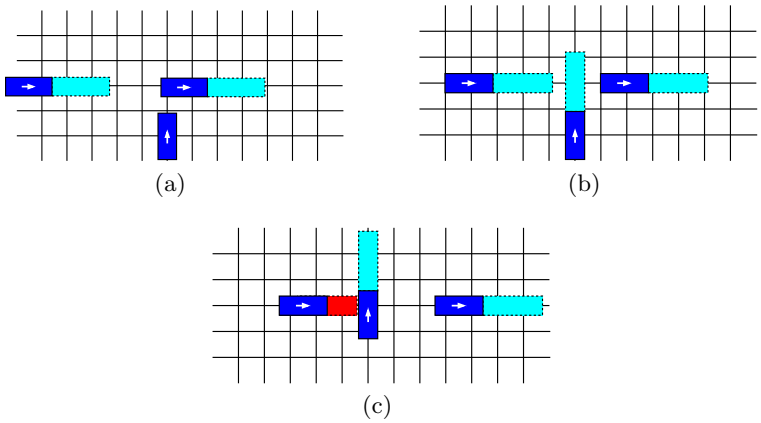


Fig. 9. Illustration of the greedy reservation procedure used in the static routing approach. Due to the greedy reservation procedure each small time window is used.

to use time windows that can not 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 since the greedy reservation procedure is of value in this case, but if the reserved areas become larger caused by a more complicated deadlock prevention, it reaches its limits.

References

1. N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
2. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44:486–504, 1997.
3. Y. Azar, J. Noar, and R. Rom. The competitivness of on-line assignment. In *Proc. 3rd ACM-SIAM Symposium on Theory of Computing*, 1992.
4. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
5. H. B. Cho, T. K. Kumaran, and R. A. Wysk. Graph theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(3):413–421, 1995.
6. J. Gao and L. Zhang. Tradeoffs between stretch factor and load balancing ratio in routing on growth restricted graphs. *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 189–196, 2004.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

8. E. Gawrilow, E. Köhler, R. H. Möhring, and B. Stenzel. Dynamic routing of automated guided vehicles in real-time. Preprint 2007/39, TU Berlin, Inst. Mathematik, 2007.
9. W. H. Guan and K. M. R. L. Moorthy. Deadlock prediction and avoidance in an AGV system. SMA Thesis, 2000.
10. K. H. Kim, S. M. Jeon, and K. R. Ryu. Deadlock prevention for automated guided vehicles in automated container terminals. *OR Spectrum*, 28(4):659–679, 2006.
11. C. C. Lee and J. T. Lin. Deadlock prediction and avoidance based on petri nets for zone control. *International Journal of Production Research*, 33(12):3239–3265, 1995.
12. S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. Technical Report KAM-DIMATIA Series 98-410, 1998.
13. N. Q. Wu and M. C. Zhou. Resource-oriented petri nets for deadlock avoidance in automated manufacturing. *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, pages 3377–3382, 2000.
14. M. S. Yeh and W. C. Yeh. Deadlock prediction and avoidance for zone-control agvs. *International Journal of Production Research*, 36(10):2879–2889, 1998.