

Transforming Automatic Scheduling in a Working Application for a Railway Infrastructure Manager

Florian H.W. Dahms ^a, Anna-Lena Frank ^b,
Sebastian Kühn ^b, Daniel Pöhle ^{b,1}

^a Vulpes AI GmbH

Textorstrasse 97, 60596 Frankfurt am Main, Germany

^b neXt Lab, Timetable and Capacity Management, DB Netz AG

Rotfeder-Ring 3, 60327 Frankfurt am Main, Germany

¹ E-mail: daniel.poehle@deutschebahn.com, Phone: +49 (0) 69 265 48267

Abstract

In this article, we present a practical approach for the optimized creation of railway timetables. The algorithms are intended to be used by Deutsche Bahn, Germanys largest railway infrastructure provider. We show how our methods can be used, both for creating a timetable in advance and for answering ad-hoc requests coming in via a digital app. Numerical experiments are provided to show that our solution exceeds manual creation of timetables in terms of capacity usage, travel times and the time taken for creating the timetable.

Keywords

railway timetable computation, traffic networks, network optimization

1 Introduction

The usual process of creating a timetable, especially for freight trains, is a manual *make-to-order* process. But as Feil and Pöhle (2014) already pointed out: It is necessary to overcome the manual process to be able to find global optimal solutions in an industrialized creation of timetables as the amount of data to be processed is steadily increasing. Additionally, in order to improve the efficiency of the process of creating a timetable as well as making better use of the available infrastructure it is convenient to transform the procedure to an *assemble-to-order* process.

This transformation as well as different approaches for an automatic creation of rail freight timetables have been widely discussed in the literature. Planning of slots was first introduced for cyclic timetables using a periodic event scheduling problem (PESP) by Nachtigall (1998) and extended by Opitz (2014). Großmann et al. (2012) showed that the PESP can also be encoded as a SAT problem which leads to significantly lower computation times. For non cyclic timetables, Großmann et al. (2013) proved that a mixed integer formulation works for practical problem sizes. The train path assignment problem as the second step after planning of slots was introduced by Nachtigall and Opitz (2014) and extended for optimization with different traffic days by Nachtigall (2015). Most recently within the DFG project ATRANS considered all aspects of automatic creation of timetables (Streitzig et al. (2016); Li et al. (2017)).

In this paper, we show how the academic models mentioned above can be transformed

into an application in real world optimization with an innovation that is threefold: First, we are able to generate more capacity without actually building additional infrastructure but by making better use of it. Second, we are able to reduce the travel time for most of the scheduled trains by simultaneously considering all requests and using global optimization instead of manual, local optimization. Third, we reduce response times by implementing the first fully automated process in short-term capacity planning from ordering to the actual departure of the train. All resulting in better customer value.

The organization of the paper is as follows. First, we describe the modelling approach in Section 2. After that, we discuss the use cases and the data used in our numerical experiments alongside the results in Section 3. We close with an outlook (Section 4) to possible extensions of our approach in the future.

2 Train Path Planning and Train Path Assignment

In this section we briefly describe our modelling approach to automatically create timetables. It is divided into two main parts: First, we precalculate slots, which are located on the most frequently used parts of the infrastructure (Section 2.1). Therefore, we extend the idea of Opitz (2014). The second part is the train path assignment (Section 2.2), based on the idea of Nachtigall (1998, 2015) and Nachtigall and Opitz (2014), where some single train paths are assigned to a complete timetable.

2.1 Train Path Calculation

This section will give a brief overview of the process used for creating a train path. The entire process consists of three main steps. First, we search one or more different routes through the infrastructure. In the second step we create a network of discrete building blocks (called snippets) along these routes for which we calculate travel and blocking times. Finally, we put these snippets together to form a non conflicting train path. The same process is used to calculate train paths for individual requests as well as the precalculated slots for the annual timetable. For the slots we consider frequently travelled relations, each starting and ending in a *Betriebsstelle*. A *Betriebsstelle* is an organizational unit into which the German railway infrastructure is divided. For each relation we consider up to three different train characteristics, chosen to be representative for most of the traffic expected on the relation.

Routing

To reduce the problem size, our first step consists of finding routes that could be relevant for the train. We use an A* algorithm for finding a shortest route on our infrastructure, which is a digital representation of the German railway network. The algorithm utilizes geo coordinates for calculating the beeline distance between *Betriebsstellen* as the lower bound. For each *Betriebsstelle* we keep a list of all possible ways to traverse it. Each possibility is termed a *Fahrweg*. The *Fahrwege* form a graph where each *Fahrweg* is a vertex, with directed edges indicating which *Fahrweg* is a direct successor of another. The edge costs are based on the *Fahrwege* lengths. As certain *Fahrwege* are preferred to others, we multiply the costs with factor greater or equal to one, with larger factors for less desirable *Fahrwege*. In this way the use of intersections and the use of tracks designated for the opposite direction can be discouraged, but we do not consider track or congestion charges due to regulatory

reasons. It is this graph we use for finding routes for our train paths.

While exploring the graph the algorithm filters out all paths which are incompatible with the characteristics of the train. For example the train might exceed maximum mass or width of the Fahrweg or might require an electrified track. As it is not always the best option to use the shortest path, we create multiple alternative routes. For searching the subsequent routes, we increase the costs of edges that were used in already found routes. A route is only accepted as a real alternative if it differs from all already calculated ones in at least one Betriebsstelle. We set an upper limit to the length of alternative routes which is a multiple (1.3 in the experiments) of the shortest route found.

Snippet Creation

For each route we calculate the travel and blocking times that would be required for a train using the route without intermediate stops. To enable stopping, we search for all tracks that could be used for a stop of the train along the route. A track is only considered for stopping, if it branches off the main route and joins it again within a single Betriebsstelle. For each such track we create one snippet leading from the main route to the stop and one from the stop to the main route. These snippets have a length of 7km, a length that ensures that acceleration and deceleration to and from the main travel speed is always possible within the snippet. For each snippet we again calculate travel and blocking times. In order to connect these snippets with the main route, we cut it into snippets at the points where our stopping snippets branch off or join it. In addition to these stopping snippets, we create snippets for alternative non stopping traversals of a Betriebsstelle for all tracks that traverse the Betriebsstelle similar to the original main route. Sometimes it can be beneficial for a train to travel with less than its maximum speed to match the speed of a preceding train without stopping unnecessarily. To enable this we create alternatives of each snippet with reduced maximum speeds.

In this way we get a directed, acyclic graph of snippets representing the possible ways the train can travel along each route including intermediate stops. Each snippet has travel and blocking times calculated. A path from a source snippet (those without predecessors) to a sink snippet (without successors) will always represent a valid train path (without a specific starting time of the train).

Train Path Calculation

The last step to calculate the train path is to determine a starting time and a path through the snippet graph such that no blocking time is in conflict with another train path and the number of stops is minimized. For each snippet we can calculate the possible departure times of the snippet that are not conflicting with other trains by projecting the blocking times of other trains back to the snippets start. Thus, we get a list of intervals for each snippet where each interval represents the allowed start times. Next we reduce the intervals further to ensure that only such intervals remain that can be reached by one of the snippets predecessors. For each snippet we calculate the possible arrival times given the known departure intervals. Note, that for a snippet not ending in a stop, the resulting intervals will have the same length but are shifted by the travel time of the snippet. For stopping snippets the intervals can increase in size as long as the stop is not used by a different train. For each snippet we take the union of all its predecessors arrival intervals and intersect them with the possible departures of the snippet. This yields the new departure times of the snippet. As the snippet graph is directed and acyclic, we can find a topological ordering of the snippets such that

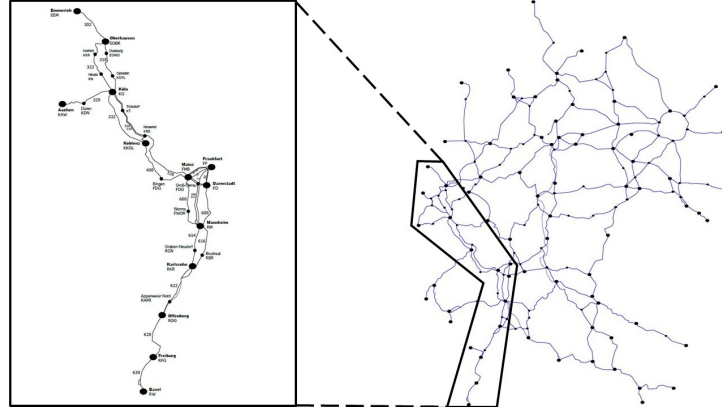


Figure 1: Right: a macroscopic map of the infrastructure administrated by DB Netze, on which slots are created. The time dependency is neglected for a better overview. Left: Zoom into the German part of the Rhine-Apline Corridor.

this reduction of departure times needs to be done only once per snippet.

The possible arrival times of the sink snippets will now be such that there must exist a non conflicting valid train path arriving at the given time. We calculate train paths by choosing the earliest arrival time. For a given arrival time we can select the predecessor snippets that have a compatible departure time. From them we select those that lead to the least number of stops (we prefer non stopping predecessors) first and then to the latest departure. The resulting train path is guaranteed to be conflict free, while keeping the number of stops and travel time low. The algorithm runs sufficiently fast, requiring $\mathcal{O}(m + n)$ calculations where m is the number of snippets and n is the number of edges in the snippet graph. Note that the maximum number of intervals per snippet is bounded by a constant, the number of seconds per day.

An example for calculated train paths is shown in Figure 2 within a path-time curve.

2.2 Train Path Assignment

As a result of the slot calculation presented in the previous section (Section 2.1), we are provided with a set of slots starting and ending at a specific Fahrweg at a specific time. The slots form a graph, we call \mathcal{G} for later reference, where the slots are the edges and the tuple of Fahrweg and point in time are the vertices. In a process consisting of four steps, this graph is used to assign train paths to the requests made by customers. First, we map the requests of the customer to the graph \mathcal{G} , resulting in so-called *break-in* and *break-out points*. Then, we search for the shortest path within \mathcal{G} , where a path represents a consecutive chain of slots. In the third part, we calculate train paths from the start of a request to the break-in point and from the break-out point to the target of the request. Finally, we optimize the result for all requests using a column generation approach.

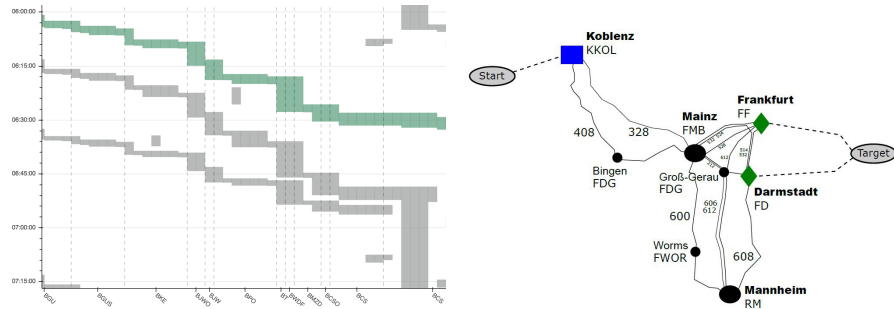


Figure 2: Left: parts of multiple train paths shown as blocking times within the time interval between 6.00am and 7.15am and Betriebsstelle BGU and BCS. Right: a small section of Figure 1. The customer request starts at *start* and ends in *target*. The break-in and break-out points are marked as blue box (□) and green diamond (◇), respectively. The time dependency is neglected for a better overview.

Calculation of Break-in and Break-out Points

As most customer requests do not start or end at vertices of the graph \mathcal{G} (compare Figure 2), we need to map the start and end of a request to vertices of \mathcal{G} , which we call break-in and break-out points, respectively. Therefore, we calculate a number of different routes (10 in the experiment), using the A* algorithm previously described. Then, beginning at the start of the route, we consecutively check for each Fahrweg of each route whether it is associated to a vertex of the graph \mathcal{G} . If so, we found a break-in point and stop; otherwise no sequence of predefined slots can be used and the request has to be fulfilled with an individually calculated train path. In order to find the break-out points, we repeat the above process starting from the end of each route.

Routing on Slots

The mapping of a request previously described may not be unique (as indicated in Figure 2), so that multiple sources and sinks have to be considered. Furthermore, due to the technical properties of the requests such as e.g. length, acceleration or width, the use of slots is restricted depending on the request. Those as well as the slots also have time restrictions. Thus we end up with a restricted, time-dependent multi-source-multi-sink shortest path problem to be solved for each request.

We use standard techniques to reduce the complexity of the problem like time expansion to tackle the time dependency. We model the multi-source-multi-sink problem with dummy edges from a super source and to a super sink. We cope with the restriction due to the technical properties by using dynamic filters. In that way, we are able to reduce the problem to a standard shortest path problem, which we solve using a Dijkstra algorithm.

Individual Train Path Calculation

In the previous steps for each request, we either create a path of slots from a break-in point to a break-out point or we know that there is none. So in order to provide a train path from start to end for each request, we individually create train paths for the missing parts of each request, i.e. either both a train path from start to break-in point and a train path from break-

out point to end or a train path from start to end. For those train paths we use the approach presented in Section 2.1. Consequently, after that step we provide exactly one train path per request.

Optimization

The process described above is sufficient if only one request has to be fulfilled at a time like in the use case of the app. But if there is more than one request – like in the use case of creating an annual timetable – the simple assignment of the shortest path for each request leads to conflicts between the train paths. We solve these conflicts using a column generation approach, where in each iteration we generate new train paths which resolve more conflicts than in the iteration before. Our experiments show, that the process terminates within up to 10 hours for sufficiently large problems (compare Section 3.2).

3 Use Cases and Numerical Experiments

In this section, we describe two use cases (Section 3.1 and 3.2) to which we are able to apply the methods mentioned before. For each use case we provide numerical experiments showing the threefold benefit of faster response times, increased capacity usage and reduced travel times.

3.1 Click&Ride App

For a short-term train path request, e.g. a train run for the next day, we can improve the response time to the railway operator by using our approach in a fully automated process. We will introduce the new way of booking a train path with a mobile application called *Click&Ride-App*. We commit to provide the railway operator with a train path offering in at most three minutes. In comparison, today’s process for manual planning takes several hours in most cases and may require up to three days. To ensure a maximum duration of three minutes we need to automate every single step in the planning process. A simplified process sequence is shown in Figure 3.

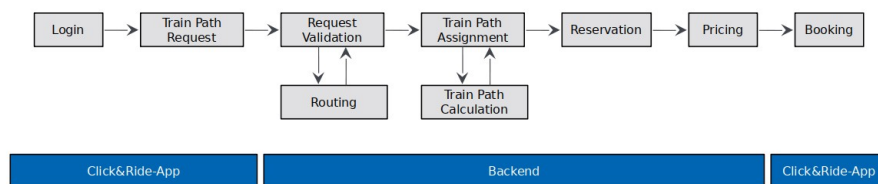


Figure 3: Simplified process of Click&Ride. The process is fully automated in the backend.

Click&Ride is a new B2B channel. After logging in, the train path request will be submitted to the back-end processes. Figure 4 gives an impression of the user interface of Click&Ride. At first there is a validation service that ensures formal and technical fit to the tracks that will be used. For example an electric vehicle cannot use a track with no overhead line. If there are any problems with the train path request, the railway operator will get instant feedback on the app’s screen and the possibility to change the request. If

DB Click&Ride

1. Angabe Zeit / Relation

Zeitpunkt der Fahrt: ☒ Abfahrt ☐ Ankunft

Relation:

☒ Heute ☐ Morgen ☐ Übermorgen

Früheste Abfahrt: 23:00

Via-Punkt einfügen:

Uhrzeit:

2. Angabe Zug

Triebfahrzeug:

Wagenzug:

Fahrt ohne Wagenzug: ☐

Triebfahrzeug ergänzen:

☐ ETCS L2 ☒ LZB ☒ EBusA

☐ ETCS L3 ☐ CIR

Gesamtzug Richtungswechsel erlauben: ☒

Wagengruppenanzahl:

3. Weitere Angaben

Vorgang:

Zugnr.kontingenz/normal, Zugnr.:

Technische Unterstützung: ☒

Impression: ☐ Nutzungsbedingungen ☐ Datenschutzerklärung

Gefördert durch: ☐ Bundesministerium für Verkehr und digitale Infrastruktur

© 2018 Deutsche Bahn AG

entworfenes Logo des Deutschen Bundesverkehrsministeriums

Figure 4: GUI Click&Ride to put a request. First the time requirements and waypoints need to be entered (upper left). Then the characteristics of the requested train need to be provided (lower left and upper right). Thirdly, in the lower right, there is space for additional information.

the request is validated, it is handled by the optimization in train path assignment (see Section 2.2). If the technical properties of the requested train do not match the existing slots on the lines, a train path will be generated automatically by train path calculation as described in Section 2.1. Railway operators have a limited time to check the offered train path before booking. To make sure the capacity on the track cannot be given to other railway operators in the meanwhile, the train path is reserved for a maximum of ten minutes. To complete the response the total price for the train path is calculated and displayed in the Click&Ride-App. An example response is shown in Figure 5.

Numerical Experiments

For our experiments we use 1301 real customer requests from November, 14th, 2013 in Germany. A customer request consists of the waypoints, time requirements and the characteristics of the train. The waypoints are at least the start of the request and its target, but may also include some stops which should be served in between. The time requirements for our data consists only of an interval at the start of the request. But it is also possible to provide further time restrictions on the other waypoints. The characteristics of the request include all data necessary to calculate its dynamic properties such as acceleration and its static parameters like length, width and mass of the requested train. We consider the actual German infrastructure available in 2013. Furthermore, we also regard the blockages of passenger trains, which were scheduled on November, 14th, 2013 in order to have a realistic setup. We measure the quality of a train path in a metric called BFQ (*Beförderungszeitquotient*) which is the travel time actually required by the train divided by the travel time that would have been necessary without additional stops. Note that for the Click & Ride BFQ we use the shortest travel time on the shortest route in the denominator while the BFQ for the manual planners was calculated using the shortest travel time on the route chosen by the planner.

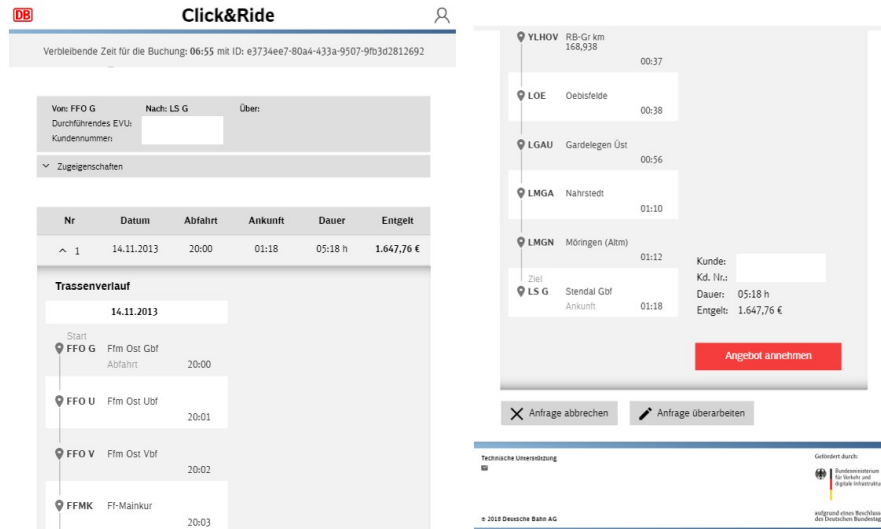


Figure 5: Part of the response, i.e. a train path, to the request shown in Figure 4.

This slight discrepancy in the numbers is to the advantage of the manual planners, as the route in the denominator could be longer.

Table 1: Percentiles for response times, automatic BFQ and manual BFQ for 1301 customer requests.

Metric	50%	90%	95%	99%	Max
C&R response time	3.68 sec.	50.28 sec.	82.98 sec.	147.89 sec.	222.34 sec.
C&R BFQ	1.03	1.46	1.73	2.53	6.07
Manual BFQ	1.22	1.94	2.52	4.35	10.54

The experimental results, as provided in Table 1, show that 95% of the request are served within 82.98 seconds. 95% of requests have a BFQ of at most 1.73 while the same percentile had a BFQ of 2.52 when planned manually. The maximal response time is 222.34 seconds. Our target of serving a response in less than three minutes can be fulfilled in all but a few edge cases (4 out of the 1301). This is a vast improvement compared to the up to three days required in the current manual process. The BFQ shows that the automatic process on average leads to faster train paths compared to the manual process.

3.2 Annual Timetable

The introduced methods will allow us to change the process of the creation of an annual timetable. In this use case all customer requests, for passenger and freight trains, are put at the same time and have to be provided with a timetable fulfilling all requests within 50 days. This currently means a huge effort to DB Netz and can be eased by planning the freight trains automatically. In an iterative process we manually create timetables for the

passenger trains first. In the second step, the freight trains are calculated automatically with the methods described in Sections 2.1 and 2.2. Afterwards the timetables are adapted and improved iteratively (compare Figure 6 for a sketch of the process).

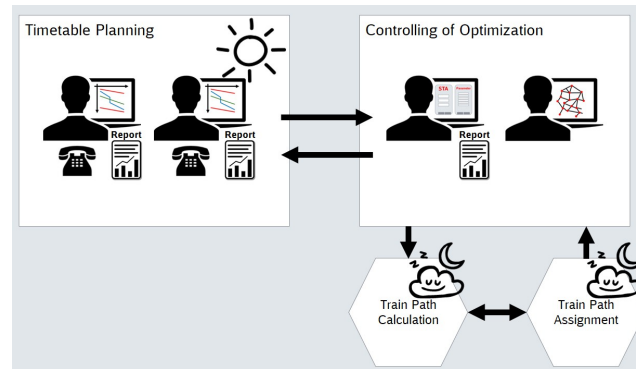


Figure 6: Iterative Process for Annual Timetable: Planning by hand during the day and automated optimization during the night

Numerical Experiments

For this experiment we again consider the data from November, 14th, 2013 regarding infrastructure, blockages and customer requests. For clarity of presentation we restrict the shown results to the German part of Corridor One (EEIG Corridor Rhine-Alpine EWIV (2019)) rather than the entire German network. Corridor One stretches from sea ports of Rotterdam, Zeebrugge, Antwerp, Amsterdam and Vlissingen to the port of Genoa covering Netherlands, Belgium, Germany, Switzerland and Italy. Here only the part in Germany from Emmerich and Aachen to Basel will be considered. We have a test set consisting of 210 requests and create slots on 38 sections (compare Figure 1, left).

The experiment shows, that our approach is very efficient compared to manual planing in 2013, as we are able to create an average of about 3% more slots. Furthermore the creation of 935,814 slots only takes 2.5 hours compared to several days back in 2013.

Table 2: Key performance indicators (KPIs) for the train path assignment

KPIs	Result
Number of Customer Requests	210
computing time	02:57h
percentage train path assignments on slots	80%
average BFQ	1.27

From Table 2 we see that precalculation of slots is beneficial for our approach as 80% of all assigned train paths use slots. 20% of the requests are assigned to individually created train paths. The average BFQ is 1.27 which is more than 5% better compared to the results of the manual process in 2013. As the train path assignment takes about 3 hours, a whole timetable for Corridor One is created in 5.5 hours fulfilling 210 customer requests on 935,814 slots.

4 Outlook

The presented approach of freight train timetable creation prepares DB Netz AG for the future by making better use of the infrastructure and reducing manual workload. Furthermore, this approach is a way to offer timetables faster and in better quality than nowadays. For the future, we plan to provide a choice set of up to three different timetables within the app, for the customer to decide which fits best. Up to now, we only consider one day for the planning, so a possible limitation might result from the extension to a whole year, e.g. the amount of data generated or the requirement to create homogenous timetables. Nevertheless, we are currently working on this extension to cover multiple days in a simultaneous slot assignment. Additionally we want to use the presented techniques for additional use cases, e.g. the creation of short term timetables and the handling of construction sites. This requires that we extend the algorithms from freight to passenger train scheduling.

Acknowledgements

This work was financially supported by BMVI, Project DigiKap. We also thank Sascha Diekmann for his support with the figures.

References

- Feil, M., Pöhle, D., 2014. “Why Does a Railway Infrastructure Company Need an Optimized Train Path Assignment for Industrialized Timetabling”, International Conference on Operations Research, Aachen.
- Großmann, P., Hölldobler, S., Mantey, N., Nachtigall, K., Opitz, J., Steinke, P., 2012. “Solving Periodic Event Scheduling Problems with SAT”, In: *Advanced Research in Applied Artificial Intelligence*, Springer Berlin Heidelberg.
- Großmann, P., Labinsky, A., Opitz, J., Weiß, R., 2013. “Capacity-utilized Integration and Optimization of Rail Freight Train Paths into 24 Hours Timetables”, In: *Proceedings of the 3rd International Conference on Models and Technologies for Intelligent Transportation Systems 2013*, TUDpress, Dresden.
- Li, X., Nachtigall, K., Martin, U., Oetting, A., 2017. “Methodik zur effizienten markt geeigneten Trassenbelegung im spurgeführten Verkehr”, *Eisenbahntechnische Rundschau*, vol. 6.
- Nachtigall, K., 1998 “Periodic Network Optimization and Fixed Interval Timetables”, Habilitation thesis, Hildesheim.
- Nachtigall, K., Opitz, J., 2014. “Modelling and Solving a Train Path Assignment Model”, International Conference on Operations Research, Aachen.
- Nachtigall, K., 2014. “Modelling and Solving a Train Path Assignment Model with Traffic Day Restriction”, In: *Operations Research Proceedings 2015*, Springer, Heidelberg.
- Opitz, J., 2009. “Automatische Erzeugung und Optimierung von Taktfahrplänen in Schienenverkehrsnetzen”, Dissertation, Dresden.
- Streitzig, C., Nachtigall, K., Martin, U., Oetting, A., 2016. “Anforderungsgerechte Algorithmen zur effizienten marktgeeigneten Trassenbelegung”, *Eisenbahntechnische Rundschau*, vol. 8.
- EEIG Corridor Rhine-Alpine EWIV. “<https://www.corridor-rhine-alpine.eu/about-us.html>” Website by 2019-01-31.