# An Approach to Optimize Certain Properties of a Train Connection in Timetable Information

**Ein Ansatz zur Optimierung bestimmter Eigenschaften einer Zugverbindung bei der Fahrplanauskunft**

Bachelor-Thesis von Daniel Roth

Tag der Einreichung:

1. Gutachten: Prof. Dr. Karsten Weihe
2. Gutachten: Dr. Mathias Schnee
3. Gutachten: M.Sc. Felix Gündling

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Algorithmik

An Approach to Optimize Certain Properties of a Train Connection in Timetable Information
Ein Ansatz zur Optimierung bestimmter Eigenschaften einer Zugverbindung bei der Fahrplanauskunft

Vorgelegte Bachelor-Thesis von Daniel Roth

1. Gutachten: Prof. Dr. Karsten Weihe
2. Gutachten: Dr. Mathias Schnee
3. Gutachten: M.Sc. Felix Gündling

Tag der Einreichung:

**Erklärung zur Bachelor-Thesis**

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den November 3, 2015

(Daniel Roth)

**Abstract**

We introduce an approach that adds attributes, like night train or on-board restaurant, as search criteria to a time table information system. It is possible to specify temporal requirements like when or for how long the attribute may occur. The challenge is to find optimal journeys that maximize the duration of an occurring attribute while fulfilling all temporal requirements and still minimizing the other search criteria travel time, number of transfers and price. Our approach overcomes those problems by changing mainly the domination criteria of the used Pareto Dijkstra multi criteria search. Early elimination of potential journeys that do not fulfil the requirements lead to the result that in most of the cases the search time increase is merely measurable. Even a challenge like night train search where big distances can lead to many possible journeys requires in average only 8% more search time. For bike transportation the search time can even be reduced.

**Zusammenfassung**

Wir stellen einen Ansatz vor, der es ermöglicht Attribute, wie Nachtzug oder Boardrestaurant, als Suchkriterien einem Fahrplanauskunftssystem hinzuzufügen. Weiterhin ist es möglich zeitliche Abhängigkeiten zu spezifizieren, wie wann oder wie lange das Attribut verfügbar sein soll. Hierbei ist die Herausforderung optimale Verbindungen zu finden, die die Dauer eines Attributes maximieren und gleichzeitig alle zeitlichen Abhängigkeiten erfüllen. Zusätzlich müssen die drei anderen Suchkriterien Reisezeit, Anzahl an Umstiegen und der Preis minimiert werden. Unser Ansatz löst diese Probleme, indem vor allem die Dominanzkriterien der verwendeten Pareto Dijkstra Suche angepasst werden. Die Suchzeit erhöht sich hierbei nur geringfügig, da Verbindungen, die die gestellten Anforderungen nicht erfüllen, frühzeitig Ausgeschlossen werden können. So ist bei der Nachtzugsuche eine Druchschnittlicher Anstieg der Suchzeit von 8% zu messen. Bei der Fahrradmitnahme wird die Suchzeit sogar reduziert.

## Contents

# 1 Introduction

Many trains provide special services for customers, if it is something simple like providing a wagon for bike transportation like it is seen in most of the trains nowadays or something more extravagant like serving a meal in an on-board restaurant. For bigger distances where journeys start at night and finish the next day, night trains are provided that create an environment that makes it possible to sleep for the duration of the journey. The advantage of integrating attributes in the search is that the other search criteria travel time, number of transfers, and price can be optimized at the same time. Manually searching those journeys always yields the risk of finding journeys that are too expensive, take too long or require too many transfers. A customer would shy away from these more easily in fear of making a bad deal. Finding guaranteed optimal journeys makes him maybe reconsider. The positive effect is that on the one hand the customer pays less and therefore probably uses the service more often and on the other hand the corporation makes more money and gets a better reputation. A better integration of the services could also make it more attractive for the corporation to expand it over more trains.

In this thesis we present an approach that extends the search in a time table information system with certain train attributes. This arises two challenges. First we need to identify in what way a customer wants to search for attributes and what attributes qualify for this. Upon that, a satisfying model has to be created. Then we need to find a way to implement the model in the existing time table information system without violating the property of finding optimal results. To see how the approach would perform in practice, we need to examine if the resulting journeys deliver realistic alternatives for the customer. Additionally, the question, under which circumstances the best results can be expected, must be answered. With circumstances the distance between the source and target station and the start time of the query are meant. Furthermore, the search has to be fast enough for a smooth user experience especially for big graphs like the one from *Deutsche Bahn AG* it must scale.

At first in Chapter 2 we present the model, the approach builds up on. This includes the graph model as well as the multi criteria search that is used to find optimal results. In addition, we give a definition of optimality. The whole algorithmic approach is explained in Chapter 3. It divides in two parts, the integration of indispensable and temporal bounded attributes. Both parts can be implemented fully independently. In Section 3.3 an approach is presented to solve problem that now skipping a train at the same station can yield better journeys. In Chapter 4 we analyse how the approach performs in practice for the attributes bike transportation, on-board restaurant, and night train. This includes a graph analysis, the creation of realistic customer oriented queries and the effects on the search criteria travel time, number of transfers, and price.

## 1.1 Related Work

There are not many approaches for specifically integrating a night train search in a time table information system. Two approaches are presented in [GSMH11]. The first one uses the fact

that in general only few nigh train connections are available. Thus, it is possible to efficiently iterate over all qualifying. To find a good journey, so called feeder connections are looked for. These are two connections, one that brings the customer from the source station to the station where the night train departures and one that brings him from the night train to the target station. The second approach is similar to the one from this thesis. The sleeping time itself is added as a new criterion to the multi-criteria search. Without optimization, this approach leads to unacceptable high calculation time. Their approach differs in the way how journeys that will not yield optimal solution are pruned. They make use of different heuristics that result in finding in some cases non optimal solutions.

# 2 Existing Model

The approach in this thesis builds up on the time table information system from [Dis07] and the changes made to it from [Gü15]. We will first give a brief summary of the graph model where we describe the aspects on which our approach builds up on.

## 2.1 Graph Model

The graph contains stations that are connected by routes. Every station has a station node. A route is a sequence of route nodes that are connected by route edges. Every route node is assigned to a station node. A route is characterized by the fact that all trains of a route hold at the same stations in the same order. At every station multiple trains of different routes can pass and overtake each other while trains of one route always drive in the same order while not overtaking them self. A route edge between two route nodes A and B stores the information when each train of the route departures at A and arrives at B. For every pair of departure and arrival time a **LightConnection** object is denoted. It also stores a pointer that points to an object, called **FullConnection**, that saves more related information like the price or the train class.

## 2.2 Multi Criteria Search

The goal of the timetable information system is to find journeys from a source station to a target station where each journey is optimal. To allow each journey to be optimal, no resulting journeys may be better in all search criteria than any other resulting journey. The search criteria travel time, number of transfers, and price need to be minimized.

Such a multi criteria search can lead to situations where although only, lets say, the two criteria travel time and cost are used but the result still consists of multiple journeys. An example for this is given in table 2.1. Every journey, if optimal, can not be dominated by any other. While the first one is faster than every other it also costs the most. The second one is slower but costs less. The same goes for the third and fourth. This could be continued even further. This example demonstrates that a simple algorithm for finding shortest paths, like the one from Dijkstra, won't be enough to find a satisfying solution.

| duration | cost |
|---|---|
| 10 | 100 |
| 20 | 90 |
| 30 | 80 |
| 40 | 70 |

**Table 2.1:** A simple example that considers only the two search criteria duration and cost.

## 2.3 Label Domination

Because it is required to optimize multiple criteria, all values that need to be optimized are saved. Since situation like the one from table 2.1 can occur, it can be necessary to save multiple paths for each node that are all optimal. This is done by using **Label** objects. A label saves a value for each criterion and a pointer to the label at the previous route node.

The following rule is used to decide if one label dominates another:

For the set of search criteria $SC_{old} = \{travelTime, transfers, price\}$ a label $A$ dominates a label $B$ iff

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &\exists x \in SC_{old} : A.x < B.x \end{aligned} \tag{2.1}$$

In other words if B would be removed there is still a label that is equally good in all criteria and at least in one better. This gives a guarantee that no optimal label is removed and thus no optimal journey is removed from the result set.

| travel time in minutes | transfers | cost |
|---|---|---|
| 197 | 1 | 8526 |
| 222 | 2 | 8442 |
| 223 | 2 | 8227 |
| 248 | 3 | 8143 |
| 278 | 3 | 8101 |
| 338 | 3 | 8071 |
| 398 | 3 | 7384 |

**Table 2.2:** The tabular shows travel time, number of transfers, and cost for each resulting journey when looking for a connection from Gera Hbf to Grabow(Meckl).

## 2.4 Pareto Dijkstra

An extended version of Dijkstra, called Pareto Dijkstra, that is capable to find an optimal solution to the algorithmic problem from above is introduced. We will not go too much into detail here because the extensions from this elaboration are not affecting the generic structure of the algorithm.

Dijkstra finds a path that minimizes one given criterion. Let us say that the costs are needed to be minimized. The original form of the Dijkstra algorithm goes through every node of a graph and saves the guaranteed minimum cost value for each node until the target node is found. That is the minimum cost of a journey from the source to the target node. To obtain the sequence of stations that lead to the node, a pointer that points to the previous node is also saved. To optimize for multiple criteria, multiple labels instead of only one value can be stored at every route node. Furthermore, only labels at a node may be stored that do not dominate each other.

In Algorithm 1 the generic version of the algorithm is given. It is a slightly simplified version of the one presented in [Gü15].

### 2.4.1 Initialization

The input consists of a set of rout nodes $V$ and route edges $E$. Furthermore, a set of start labels and the source and target node are given. As usual, a priority queue is used for an efficient access to the next optimal label.

As long as there are elements in the queue, the next 'best' label gets extracted, written in $l$, and removed from the queue. To make the labels comparable for the priority queue, one label is better if it has a smaller travel time to the target. If the travel times are even, the number of transfers are used. If even them are even, in a last instance the prices are getting compared.

### 2.4.2 Iteration Step

First in the loop from 5 to 16 a new label $l_{new}$ is created. If the target station reached, $l_{new}$ is inserted in the set of target labels if it is not dominated by any label there. If the target station is not reached yet, two different checks are made to check if $l_{new}$ is even good enough. First it gets compared to the labels at the target station with the help of a lower bound graph. Then, $l_{new}$ gets compared to other labels at the same node. If it is not dominated by any label, it will finally be inserted in the queue.

#### Create new Label

At first, in the call in line 6 a filter is appended that checks if the new label fulfils different criteria. In Section 3.1 filters play an important role. It is also checked if the target node is reached, in which case also no label is created. If the criteria are fulfilled, a label $l_{new}$ will be created that includes the next station at which the edges points.

#### Lower Bound

In line 11 it is checked if $l_{new}$ is dominated by a label at the target station. Algorithm 3 displays the function in pseudo code. The check is done by simply iterating over all result labels and examining if they dominate $l_{new}$. Function $dominatesLowerBound$ checks if $l_{new}$ is even capable of minimizing the target travel time, transfer count, or price. This is done by using a lower bound graph. The lower bound graph saves for each criterion the minimum value that is required to travel from each node in the graph to the target node. Because $dominatesLowerBound$ is rather simple to implement, we do not provide pseudo code. Now it is possible to predict for every label what the smallest possible values for each search criterion would be to reach the target.

#### Label Domination

In a second step in line 12 it is checked if there are labels at the next node that dominate $l_{new}$. If that is the case, no label needs to be inserted and the algorithm goes on with extracting the next label from the queue. Further, is is checked if $l_{new}$ dominates an existing label $l_{exist}$. In that case $l_{exist}$ is removed from the queue and from the set of labels. Because the set of labels at a

route node contains always labels that do not dominate each other and $l_{new}$ dominates one of these, $l_{new}$ itself can not be dominated by any other existing label at the node. What can be the case is that $l_{new}$ dominates one of the remaining labels at the node. If that is the case, those are removed as well. After iterating over all labels, $l_{new}$ is inserted in the set of labels at the node iff it was not dominated by any other label. When comparing the prices, some changes have to be made to still find optimal journeys. We will not examine this further here because it does not affect our approach.

### 2.4.3 Filter Results

At the end, in line 18 $filterResult$ is called. The pseudo code for this function can be found in Algorithm 4. It is checked if there is any resulting label that dominates another resulting label. As we can see, we use here the method $dominatesHard$ for the domination check. We again do provide, because of simplicity, no pseudo code for this function. $dominatesHard$ does not need further adaptations for the price domination treatment like it was the case for $dominates$. Up to this point, both methods differ only in the price calculation and in the way which labels are compare with them. For our approach, we need to adapt both methods in different ways (see Section 3.2).

```
1  Function void paretoDijkstra(V, E, startLabels, source, target)
       // priorityQueue
2      q ← startLabels;
3      while not p.empty() do
4          l ← p.pop();
5          for e ∈ {(l.node, x) ∈ E|x ∈ V} do
6              l_new ← l.createNewLabel(x);
7              if x = target then
8                  addResult(l_new);
9                  continue;
10             end
11             if not dominatedByResult(l_new) and
12              not dominatedByOtherLabels(l_new, x) then
13                 q.insert(l_new);
14                 labels.insert(l_new);
15             end
16         end
17     end
18     filterResults();
19     return labels(target);
```

**Algorithm 1:** function paretoDijkstra: finds best path from source to target with a multi criteria search.

```
1  Function bool dominatedByOtherLabels(l_new, node_dest)
2      for l_exists ∈ labels(node_dest) do
3          if l_exists.dominates(l_new) then
4              return true;
5          end
6          if l_new.domnates(l_exists) then
7              q.remove(l_exists);
8              labels.remove(l_exists);
9          end
10     end
11     return false;
```

**Algorithm 2:** Algorithm dominatedByOtherLabels: checks whether a given label is dominated by any label at the given node.

```
1  Function bool dominatedByResult(l_new)
2      for l_result ∈ results do
3          if l_result.dominatesLowerBound(l_new) then
4              return true;
5          end
6      end
7      return false;
```

**Algorithm 3:** Algorithm dominatedByResult: checks with a lower bound comparison if the given label is dominated by a label at the target node.

```
1  Function bool filterResults()
2      for l_result ∈ results do
3          for l_other ∈ results do
4              if l_results = l_other then
5                  continue;
6              end
7              if l_result.dominatesHard(l_other) then
8                  results.remove(l_other);
9              end
10         end
11     end
```

**Algorithm 4:** Algorithm filterResults: removes all labels that are dominated by other labels at the target node.

# 3 Search Attributes

In the following chapter we concentrate on how to extend the existing multi criteria search with the search criterion *attribute*. Every LightConnection can have multiple attributes. It is important to differ between attributes as they are saved in the current model and those who are only designated for the search, the search attributes. We assume that the most important search attributes are the following: bike and disabled person transportation, on-board restaurant, and night train. In the current model are many different attribute types where each has its own ID. Table 3.1 shows an assignment of the attribute IDs to the search attributes.

### Search Attributes in the Graph Model

As mentioned in Section 2.1, a LightConnection stores a departure and arrival time and a pointer to a FullConnection object. This objects has an object, called **Attributes**, that stores the information which of the attributes, that can be searched for, is available. For every attribute only a check if it is available or not must be possible so it is enough to assign for each attribute a bit that is either set or not. Because there are only six attributes that can be looked for, only six bits are required. It is easier to save and access whole bytes thus the remaining two bits are set to 0 so in total 8 bits are required.

### Availability Check

This kind of storage method is not only memory efficient but also allows to check with only two logical operations if the wanted attributes are available in the LightConnection or not. This works by defining first a byte $sa \in B^8$ with $B \in \{0, 1\}$ that has set a 1 for every attribute that is searched for. To check if a LightConnection has the wanted attributes, the assigned attributes byte $x \in B^8$ is used. It has the wanted attributes iff

$$sa \wedge_b x = sa$$

where $\wedge_b$ is the column wise logical AND operator.

### Example

Let us assume that the bit for bike transportation is the first one in the vector. The second one is restaurant and the third one night train. Further assume a given LightConnection has the

| Search Attribute | Attribute id |
|---|---|
| bike transportation | FB, FR, FK, FT, G , FU, h;, KF, KH |
| disabled person transportation | EH, HS, 97, OA, OB, OC |
| restaurant | BR, BT, BM, ZR |
| snack | BO, MB, SN |
| air conditioner | KL |
| night train | NZ, CN, LW, SW, LI |

**Table 3.1:** Assignment of attribute IDs from the current model to look for attributes.

attributes bike and night train. The attributes vector is than defined as $x$. If we look for bike transportation we have to set the search attribute vector as $sa_{bike}$ and if we want to look for a restaurant it should be look like $sa_{restaurant}$.

$$x = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, sa_{bike} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, sa_{restaurant} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

Because the bike transportation is available but not the restaurant, the result should be true for $sa_{bike}$ while it should be false for $sa_{restaurant}$. And indeed it is for bike transportation

$$sa_{bike} \wedge_b x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \wedge_b \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = sa_{bike}$$

but for restaurant it is

$$sa_{restaurant} \wedge_b x = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \wedge_b \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \neq sa_{restaurant}$$

### Classification of Search Attributes

There are two kinds of search attributes. Those who have to be available in each train of a journey, called *Indispensable attributes* (see Section 3.1) and those who just have to meet temporal constraints, called *temporal bounded attributes* (see Section 3.2). Both types differ in the requirements they set for the found journeys. A classification with some chosen attributes can be found in table 3.2. The next chapters explain the characteristics and give examples for each class.

| Type | Search Attribute |
|---|---|
| Indispensable attribute | bike transportation, disabled person transportation |
| Temporal bounded attribute | restaurant, snack, air conditioner, night train |

**Table 3.2:** Classification of search attributes

## 3.1 Indispensable Attributes

Indispensable attributes are all those who have to be strictly available in every train in the journey. The typical example for an indispensable attribute is the bike transportation. A customer that travels with a bike will not be able to take a train that does not allow bike transportation. So it would not make sense to show journeys where in some trains bike transportation is not possible. Another example is the transportation of disabled persons. Extensive studies on how indispensable attribute affect the connection search can be found in Section 4.1.

### 3.1.1 Implementation

In comparison to temporal bounded attributes, indispensable attributes are rather simple to implement. All labels that do not fulfil the domination criterion are filtered out. This is the case when the availability check returns false. Labels are filtered out immediately after creation, before they are compared for domination with other labels (see Section 2.4). The domination methods are therefore unaffected by this attribute.

### 3.1.2 Validation

This method is simply removing edges for the Pareto Dijkstra search, thus only effecting the graph by reducing it. Assuming that the search finds optimal journeys, this property must therefore also be true on the reduced graph.

## 3.2 Temporal Bounded attributes

In this chapter we first give a brief explanation what temporal bounded attributes are and how they differ to indispensable attributes. Then in Section 3.2.1 we explain step by step each property of temporal bounded attributes and how to implement them.

Temporal bounded attributes differ fundamentally from indispensable attributes. As the name says, the search needs to be specified with temporal bound. That means a start and an end time for when the attribute should be available should be available. While indispensable attributes require, that the attribute is available at any time, a temporal bounded attribute has to be available only within the given start and end time and even then it is not strictly required.

#### Non-indispensable

A customer can give a requirement on what attribute he wishes to be available in the train. Respectively he will get results, if available, that fulfil the requirement, but also the results that would be returned for a search without attributes are shown. We define therefore the theorem

**Theorem 1.** *The set of journeys, resulting from a search with a query, is always a subset of the set of journeys resulting from a search with the same query that additionally includes a Temporal Bounded Attribute requirement.*

The customer can therefore give his personal preferences about what attribute he wishes to be available without having to fear that journeys, that would have been shown before, are not there anymore. It is also possible to further specify time dependent criteria like when and how long the attribute should be available in the train.

In theory every kind of attribute could be a temporal bounded or an indispensable attribute because every attribute in the graph is represented by a boolean that is either set or not. In practice the set of indispensable attributes and temporal bounded ones is disjoint. That is because attributes like bike and disabled person transportation do only make sense if they are always available. Then there are attributes like restaurant, night train, or air conditioning that are dispensable. A customer would never want that a journey is not shown because no air conditioning is available. But what is shown, are journeys that provide this attribute for a longer duration, even when they are worse in all other search criteria. In Section 3.2.1 all those possible adjustments for temporal bounded attributes are presented.

### 3.2.1 Extended Search Criteria for Attributes

Each of the following search criteria gives the customer more possibilities to specify what journeys with certain attributes he will get. Of all the different possible indispensable attributes, we will focus on *restaurant* and *night train*. We can assume that these are the typical temporal bounded attributes a customer would look for. In the next sections we will refer to them in examples. In Section 4.2 we then analyse how they behave in a practical use on a real timetable from *Deutsche Bahn AG*.

### Extended data structure

The query gets extended by the attribute search criteria $startTime$ and $endTime$, $duration$, $minDuration$ and $toleranceFactor$. Furthermore, each label obtains the two variables attribute duration $ad$ and attribute max duration $amd$.

The changes that need to be made take all effect in the following four methods:

- $createLabel(lable)$

- $dominates(lable)$

- $dominatesLowerBound(lable)$

- $dominatesHard(lable)$

In Section 2.4 is explained where in the generic algorithmic approach those methods are called. The following sections explain in detail the reason for implementing them, how they influence the search and how they can be implemented in an efficient way.

#### Start and End Time

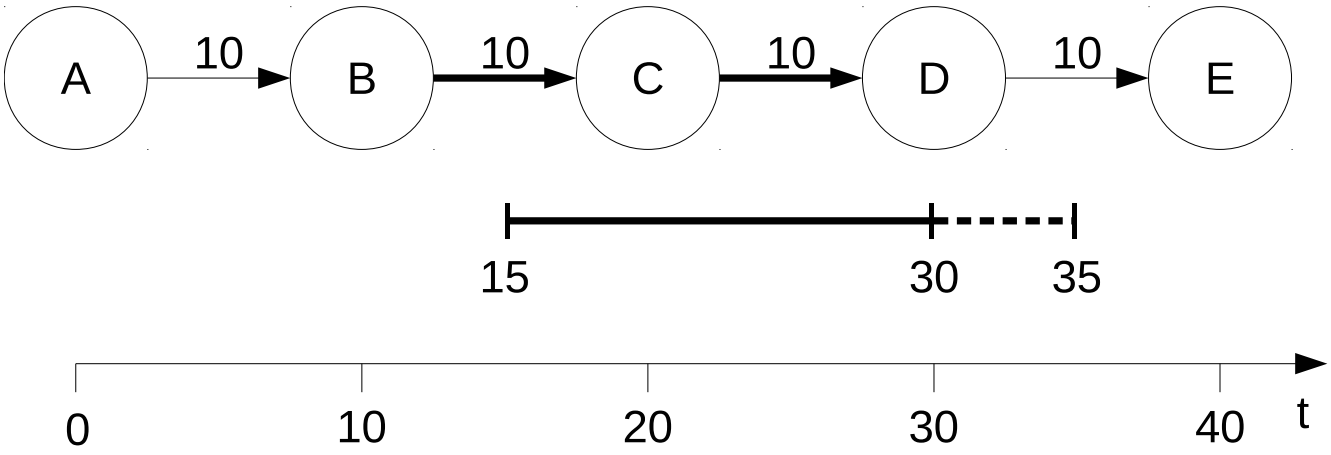


**Figure 3.1:** Example of a journey that goes from station A to E. The bold arrows from B to D indicate that here a train with a requested attribute is driving.

Every temporal bounded attribute needs to be specified with a start and an end time. The result contains journeys that maximize the time an attribute occurs in a train within the given time span.

### Intersecting time bounds

If the train intersects the start or end time, only the time within the bounds counts. That means for the start time $s$ if a train with the wanted attribute is departing at time $s - x$ and arriving at a time $s + y$, only the time the attribute is available within the temporal bounds is counted, that is $y$. Respectively for the end time $e$, if a train with the wanted attribute is departing at time $e - x$ and arriving at $e + y$, only $x$ will be considered.

In summary the attribute duration $d$ for a train that departures at $dTime$ and arrives at $aTime$ for an attribute with the specified interval bounds $startTime$ and $endTime$ is specified as follows:

$$d = max\{0, min\{aTime, endTime\} - max\{dTime, startTime\}\} \tag{3.1}$$

**Example**

Lets look at an example to get a sense on how this works. Figure 3.1 shows a representation of a journey. The circles are the station nodes and the arrows show the path the journey follows. The query has a source station A and a target station E. The chosen attribute start time is 15 and the end time 35. The journey starts at the route node A at time 0. It goes from A to B within 10 minutes. At B a transfer occurs and the new train that drives from B to C has the wanted attribute (signalized by the bold arrow). The train continues to drive to D where a transfer occurs to a train that drives from D to E but has not the wanted attribute (the arrow is not bold anymore). For simplicity it is assumed that there is no waiting time when changing the train and every next train departures within the same minute.
In the first 5 minutes, when driving from B to C with the wanted attribute, the start time is not reached yet. Thus, these 5 minutes will not count for the search but only the remaining 5 minutes. Because at C no transfer occurs, the next 10 minutes do also count for the attribute duration. At D a transfer occurs to a train that has not the wanted attribute, so this journey can not count anymore to the attribute duration. All in all the journey has a train that has the wanted attribute within the given interval for 15 minutes.

**Implementation**

The implementation requires changes in the methods $createNewLabel$ and $dominates$. In $createLable$ $ad$ and $amd$ have to be updated. The variable $ad$ saves the duration of the required attribute that the current train provides up to this point. Variable $amd$ saves the biggest duration of any train on the journey up to this point that has the required attribute. First the attribute duration gain $d$ from this edge is calculated with Equation 3.1. Then $ad$ and $amd$ are updated according to the following equation:

$$ad_{new} = \begin{cases} d & \textbf{if transfer occures} \\ ad_{old} + d & \textbf{else} \end{cases} \tag{3.2}$$

$$amd_{new} = max\{amd_{old}, ad_{new}\}$$

**Dominates without lower bound**

Now, to find journeys that optimize the two values $ad$ and $amd$, both simply have to be added as a domination criterion. Compared to the other domination criteria travel time, number of transfers, and price (see Section 2.2) $ad$ and $amd$ have to be maximized. Therefore, Equation 2.1 gets replaced by Equation 3.3

For the set of search criteria $SC_{attribute} = \{ad, amd\}$ it is

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &\forall x \in SC_{attribute} : A.x \geq B.x \wedge \\ &(\exists x \in SC_{old} : A.x < B.x \vee \\ &\exists x \in SC_{attribute} : A.x > B.x) \end{aligned} \tag{3.3}$$

In Figure 3.2 is an example that shows why not only $amd$ but also $ad$ should be added as domination criteria. For simplicity only $travelTime$ is additionally chosen as a domination criterion. A journey from S to T is looked for. There are the two possible paths:

1. S → C → B → T

2. S → A → B → T

The bold arrows again signalize a train with the wanted attribute. The path A → B → T is taken by a single train, so no transfer is required.
The first journey shows a fast train that reaches station 2 in 15 minutes. Therefore, at C it has $travelTime$, $ad$, and $amd$ of 15. The train stops here so a transfer to a train departing at the same station at route node B is required. The transfer sets the $ad$ back to 0. The journey goes along and ends after another 10 minutes at the target node. Because $ad$ was set to 0, $amd$ has not been increased anymore and stays therefore at 15.
The second journey takes 20 minutes to reach route node B. At this point, the label from the first journey is in every point, except $ad$, dominant (label b1 dominates label a1). Not considering $ad$ would result in the removal of the label. The **same** train drives along to T. At the target, the label from the second journey (label a2) can now not even according to the old rule from Equation 2.1 be dominated by the label from the first journey (label b2).
The second journey had at route node B the advantage that no train transfer was required. Staying in a train can increase $amd$ in the future because $ad$ has not to be set to 0. That is why for a domination check between two labels at a route node, the $ad$ has to be considered as well, while this is not necessary for labels that are at the target node.

#### DominatesHard

We recall from Section 2.4.3 that $dominatesHard$ is only called for labels that already reached the target node. As we have seen, here only the maximum attribute duration, that is provided by the journey, counts. Thus, only $amd$ needs to be considered and not $ad$. A label A then dominates a label B iff

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &B.amd \geq A.amd \wedge \\ &(\exists x \in SC_{old} : A.x < B.x \vee \\ &B.amd > A.amd) \end{aligned} \tag{3.4}$$

#### Dominates with lower bound

It is possible to calculate the maximum attribute duration value that a label can possibly reach until it reaches the target node. This information can be used to exclude labels from the search earlier. Similar to the lower bounds for travel time, number of transfers, and price we define an upper bound $L.ub$ for a Label $L$. It is

$$L.ub = max\{A.amd, A.ad + endTime - now\} \tag{3.5}$$

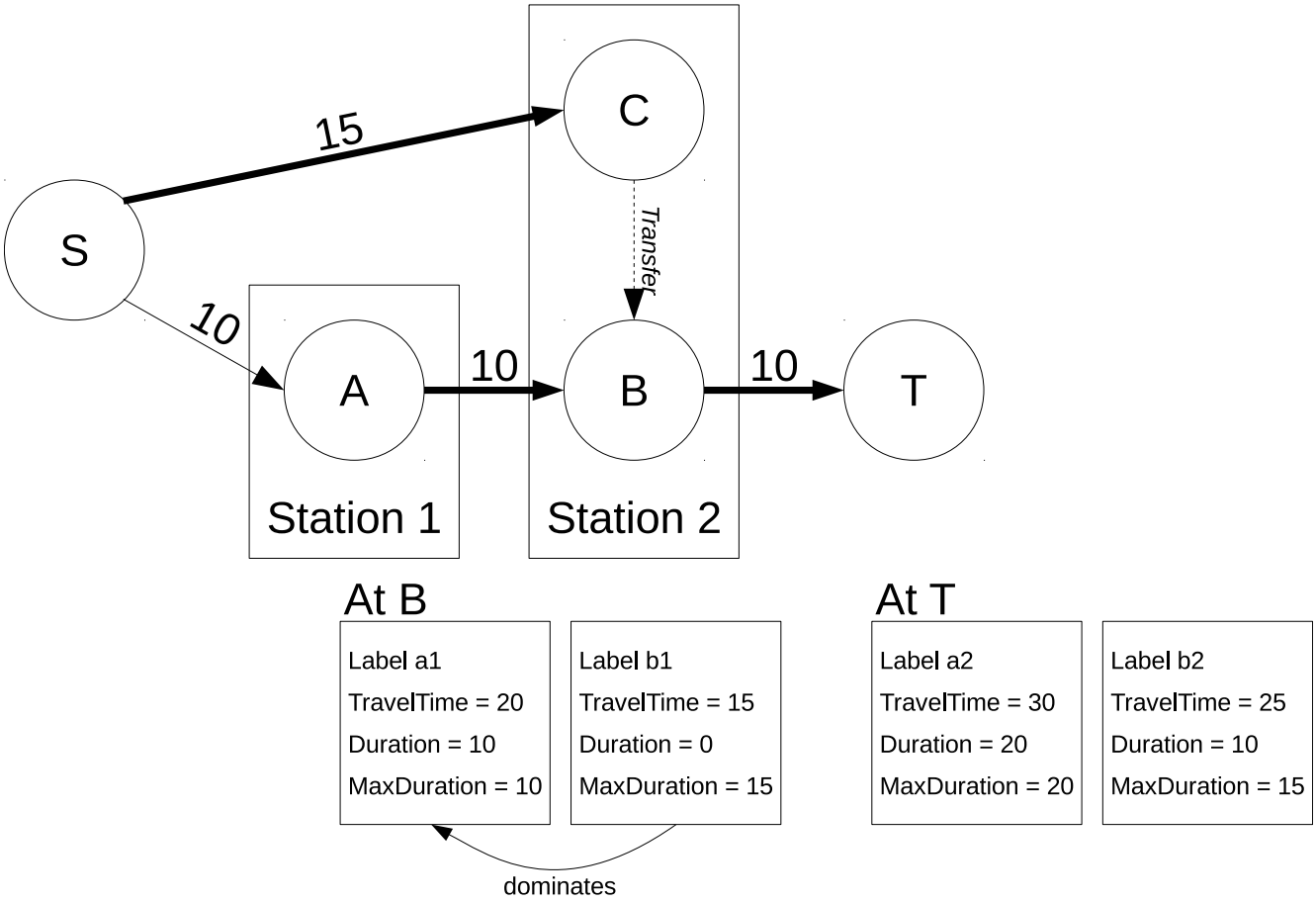


**Figure 3.2:** Example that shows two journeys from station S to station T. The bold arrows indicate that a train with the search attribute is driving. For Node B and T are the two labels given that are created while searching.

Because the attribute duration needs to be maximized, an upper bound can be used in the same way as a lower bound for a minimization problem. Label B then dominates A iff

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &B.amd \geq A.ub \wedge \\ &(\exists x \in SC_{old} : A.x < B.x \vee \\ &B.amd > A.ub) \end{aligned} \tag{3.6}$$

### Duration

The attribute duration value gives the customer the possibility to decide how long he wishes to have the attribute available in a train. This also means, when there are two journeys that both exceed the attribute duration value but one has a higher value, the one with the lower can still dominate the other. That is because there is no reason to accept a higher travel time, price, or more transfers if the duration, the attribute was wished for, is already reached.

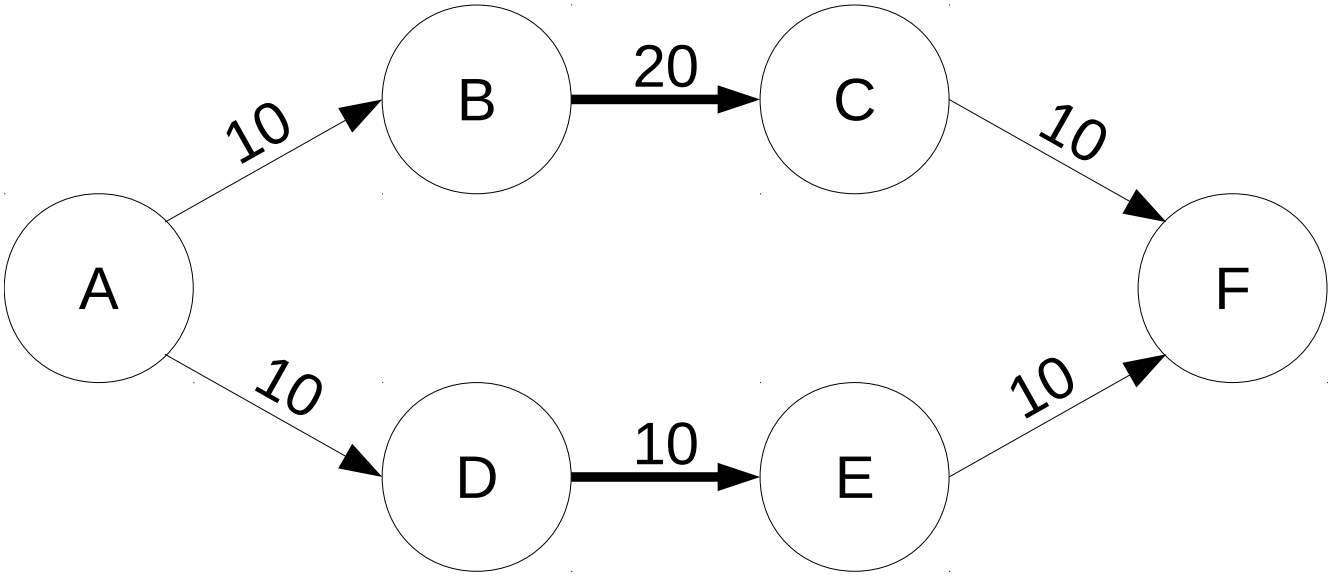


**Figure 3.3:** Example that shows two paths from station A to station F. The bold arrows indicate that a train with the wanted attribute is driving.

A good example where this is useful is when the customer wants to take a meal in the train. He knows he will not require more than 30 minutes for this. A journey with 30 min restaurant is enough then, respectively one with more restaurant time but in general worse values for other search criteria would make no sense for the customer. Another example is when looking for night trains. Every person has different preferences in how much sleep he requires. Thus, it is conceivable that some customers only want to sleep for 6 hours while others prefer 8 or more. Because the range can be set differently for every search, every customers preferences can be reached.

#### CreateLabel

The duration value acts as an upper bound. A value that is bigger or equals the search duration value is as good as any other value that is also bigger or equal. So setting the duration value as an upper bound for $ad$ seems to be a reasonable choice. We simply extend Equation 3.2 by restricting the calculated $ad_{new}$.

$$ad_{new} = \begin{cases} min\{duration, ad_{old} + d\} & \textbf{if transfer occures} \\ min\{duration, d\} & \textbf{else} \end{cases} \tag{3.7}$$

#### Dominates with lower bounds

Similar to Equation 3.7, the upper bound needs to be limited to $duration$.
Equation 3.5 has to be updated to

$$A.ub = min\{duration, max\{A.amd, A.ad + endTime - now\}\}$$

Setting a relatively small duration value can have a major impact on the performance because as soon as a label that has reached the upper bound is at the target node, it is able to dominate other labels with worse values in the other criteria.

**Example**

Lets have a look at an example to understand the impact of different duration values. Figure 3.3 shows a graph with route nodes. The arrows again represent trains that drive for a given duration. Bold arrows represent trains that have the wanted attribute. Journeys from A to F are looked for. A quick look shows that there are two paths from A to F. The upper one A → B → C → F has a total of 20 minutes attribute duration and the lower one A → D → E → F a total of 10 minutes attribute duration. The lower path requires only 30 minutes travel time so it can never be dominated by the upper that requires 40 minutes. On the other hand the upper one has 10 minutes more attribute duration. Setting the search duration to a value smaller or equals than 10 makes the upper path dispensable because both reach the same *amd* value while the upper requires more travel time. It has then no more advantage compared to the lower one. For every duration value bigger than 10 the upper path will be shown in the result because it now has a higher *amd* than the lower path and can therefore not be dominated anymore.

## MinimumDuration

The customer is now able to define his preferred duration for which he wants the attribute to be available. In a next step it would not be a bad idea if he is also able to define the minimum time for which the attribute should be available. Again we can find good examples for restaurant and night train why this would be an useful option. If the meal takes at least half an hour, any journey that has fewer restaurant duration and does get dominated by other journeys is not of interest anymore. The same applies for the night train. It is only reasonable that the customer wants to set a lower bound that filters out every journey that provides a night train for fewer than for example four hours.

As mentioned, journeys that do not provide the minimum attribute duration but are not dominated by other journeys, are still shown. That means that theorem 1 is still true.

The minimum duration value $minDuration$ acts as a lower bound. While checking for domination between two labels at a route node that is not the target node we can not apply $minDuration$ in any way because the attribute duration values for the future labels could still change. However, because it effects the domination with target labels, it can be applied for the upper bound prediction and the direct compare of two labels at the target node itself.

**Domination with lower bound**

We need to achieve that the duration values only take affect when their value is bigger than $minDuration$. Because the calculated value from Equation 3.5 is an upper bound of the $minDuration$ for this label, the adaptation can be applied a same way. This can be simply achieved by subtracting $minDuration$ and taking the maximum of this value and zero. By this, every value that is lower than $minDuration$ is set to 0.
We first state, for any label $L$ it is

$$L.amd_{new} = max\{0, L.amd_{old} - minDuration\}$$

and

$$L.ad_{ub,new} = max\{0, L.ad_{ub} - minDuration\}$$

Now, from Equation 3.6 the following equation can be deducted. Label B dominates A iff

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &B.amd_{new} \geq A.ad_{ub,new} \wedge \\ &(\exists x \in SC_{old} : A.x < B.x \vee \\ &B.amd_{new} > A.ad_{ub,new}) \end{aligned} \tag{3.8}$$

**DominatesHard**

The same can be applied when comparing labels at the target node. Equation 3.4 has to be adapted to

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &B.amd_{new} \geq A.amd_{new} \wedge \\ &(\exists x \in SC_{old} : A.x < B.x \vee \\ &B.amd_{new} > A.amd_{new}). \end{aligned} \tag{3.9}$$

**Example**

Lets see how a different $minDuration$ can affect the resulting journeys. We again assume a graph like the one in Figure 3.3 is given and we look for journeys from node A to node F. Setting $minDuration$ to a value higher than 20 will make the upper path expendable because it can only provide at most 20 minutes of attribute duration. Respectively for every $minDuration$ smaller or equal than 20, the upper path can not be dominated by the lower one.

### Tolerance factor

The last search criterion for attributes, we introduce here, is the $toleranceFactor$. The $toleranceFactor$ says how much more travel time compared to additional attribute duration a customer would tolerate. This follows the idea that journeys, that are shown because they have a higher attribute duration but require a much higher travel time, are not really of interest for the customer. Nobody would accept a 60 minutes longer journey just to have 10 minutes more of, for example, on-board restaurant time. The same can be applied for a night train. Just like the minimum duration, implementing the tolerance factor only requires changes in $dominatesHard$ and $dominatesLowerBound$.

**DominatesHard**

Equation 3.10 that emerges from Equation 3.9 contains the last changes for a domination check between two labels at the target node.

Label $B$ dominates $A$ iff

$$\begin{aligned} &\forall x \in SC_{old} : A.x \leq B.x \wedge \\ &A.travelTime - B.travelTime \geq toleranceFactor \cdot (A.amd_{new} - B.amd_{new}) \wedge \\ &(\exists x \in SC_{old} : A.x < B.x \vee \\ &A.travelTime - B.travelTime > toleranceFactor \cdot (A.amd_{new} - B.amd_{new})) \end{aligned} \tag{3.10}$$

$A.travelTime - B.travelTime$ is the additional travel time of the corresponding journey of label $A$ compared to the one of label $B$. $A.amd_{new} - B.amd_{new}$ on the other hand describes the additional attribute duration that $A$ can offer. Multiplying this value with $toleranceFactor$, that is higher than one, increases the overall value on the right side of the term making it more likely to fail and thus likelier to prevent a domination of $A$ by $B$. We note that we do not need to consider the case that $A.travelTime - B.travelTime < 0$ because then $\forall x \in SC_{old} : A.x \leq B.x$ would fail what makes the term false anyway.

#### Dominates lower bound

Equation 3.8 can be adapted to:

Label $B$ dominates $A$ iff

$$\begin{gathered}\forall x \in SC_{old} : A.x \leq B.x \wedge \\ A.travelTime_{lb} - B.travelTime_{lb} \geq toleranceFactor \cdot (A.ad_{ub,new} - B.amd_{new}) \wedge \\ (\exists x \in SC_{old} : A.xB.x \vee \\ A.travelTime_{lb} - B.travelTime_{lb} > toleranceFactor \cdot (A.ad_{ub,new} - B.amd_{new})).\end{gathered} \tag{3.11}$$

Because we are doing a lower bound comparison, we have to use the lower bound travel times from each label. It gives the shortest possible travel time from the current node at which the label is to the target node. Therefore $A.travelTime_{lb} - B.travelTime_{lb}$ is a lower bound for the additional travel time of $A$ compared to $B$. Then again $A.ad_{ub,new} - B.amd_{new}$ defines the upper bound for the additional attribute time. The term makes sure that only in the case that a lower bound has a bigger value than an upper bound, a domination can occur.

This change can have a major impact on the performance with queries with source and target station that are far apart. It allows to exclude journeys that are dominant but have a too high travel time. Usually this affects attributes that can not be excluded that easy with a preferred or a minimum duration because those values are chosen too high. For example the night train falls under this category where $minDuration$ and $duration$ usually are bigger than 4 hours. In Section 4.3.2 is described how this affects the search in practice.

#### Example

Again the result of a query on the graph from Figure 3.3 may vary depending on the chosen $toleranceFactor$. Lets consider that $minDuration$ is 0 and $duration$ a value over 20 so that they do not affect the attribute duration. The upper path is valid as long as $toleranceFactor$ is equally or higher than 1 and becomes expendable for any value lower than 1. This is due to the fact that the upper path requires a 10 minute longer travel time but also provides 10 minutes more of attribute duration. The different search criteria for attributes are also dependent of each other. If $duration$ would be chose to a value lower than 20, $toleranceFactor$ can be chosen higher to still accept the upper path. Only a $duration$ lower than 10 makes the upper path expendable, independent of $toleranceFactor$ (because the lower path already provides 10 attribute duration).

### 3.2.2 Validation

We have seen that implementing temporal bounded attributes only requires to change the domination criteria from equations 3.3, 3.11, and 3.10. Furthermore, we have shown why the search with the changes still finds under the given restrictions optimal journeys. Thus, assuming that the algorithm without any changes finds optimal solutions, our approach does this as well.

## 3.3 Optimizations for Time-dependent Graph Models

In the Pareto Dijkstra algorithm from Section 2.4, when popping the next element from the queue, only one label for the next departing train is created. This indeed provides the optimal results for travel time, number of transfers, and price because of the constraint that no trains of one route may overtake each other. With attributes this situation changes a little bit. It is possible that, although the next departing train does not have the wanted attribute, there may be a train departing later that has it. Taking a train later makes mostly sense when the attribute is indispensable, when the tolerance factor is high enough to allow a higher travel time for the cost of more attribute duration or the waiting time for the next train is considerably low.

### Implementation

Further departing trains are only of interest if the current train does not have the wanted attribute and if the other trains may have the attribute. In a preprocessing part, a bitfield $b_{av}$ is used to describe all attributes that occurred in any LightConnection at the edge. The current state of still required attributes are saved in another bitfield $b_{req}$.

In Algorithm 5 is the pseudo code of the algorithm. At first the next label is created and inserted in the queue. This is done by $createLabel$. In line 3, $b_{req}$ is updated. $\wedge_b$ and $\textbf{not}_\textbf{b}$ are the bitwise operators for $\wedge$ and **not**. All bits that are 1 at $b_{av}$ are set to 0 at $b_{req}$. The while loop from line 4 to 10 iterates over the next LightConnection objects as long as there are LightConnection objects in the edge available with required but not yet found attributes. If one is found, $b_{req}$ is again updated. The loop stops if no more attributes are required ($b_{req} \neq (00000000)$) or there is no more LightConnection object. The second case only occurs if there was a LightConnection object with the wanted attribute before the search time, but non after.

```
1  Function void createNewLabel(label, edge, nextConnection, b_req)
2      label.createLabel(edge, nextConnection);
3      b_req = b_req ∧_b not_b b_av;
       // find next train with at least one common attribute
4      while b_req ≠ (00000000) ∧ next train available do
5          nextConnection = getNextDepartingTrain();
6          if not b_req ∧_b b_av = (00000000) then
7              label.createLabel(edge, nextConnection);
8              b_req = b_req ∧_b not_b b_av;
9          end
10     end
```

**Algorithm 5:** Algorithm createNewLabel: Creates a label at the target station of a given edge.

# 4 Computational Study

In this chapter we analyse the results of randomly generated queries with attributes. First we examine for each attribute how the queries should be generated so that they are as realistic as possible. This involves fundamental assumptions and graph analyses. Then we compare the results from 10,000 queries with attribute condition to 10,000 without, with regard to, among others, the distance of source and target station. In Section 4.3 a runtime analysis is done for all queries with attributes. The queries are executed on a real 2 weeks long timetable from *Deutsche Bahn AG*.

## 4.1 Bike Transportation

Here we analyse how the changes take affect on real queries with bike transportation for the timetable search. In the provided timetable of all LightConnections only 5.03% have bike transportation. But if we only take trains into account, more than 97.13% have it. To get realistic examples we only created queries of stations that are reachable by trains of the classes from table 4.2. Other transportation vehicles are for example buses and trams.

| Train Classes | total amount | bike | restaurant | night train |
|---|---|---|---|---|
| ICE THA TGV RJ | 128,044 | 30 | 74,095 | 0 |
| EC IC EX D | 127,744 | 76,784 | 42,509 | 285 |
| CNL EN AZ | 54,896 | 6,800 | 164 | 10,331 |
| IRE REX RE IR X DPX E Sp | 828,796 | 768,143 | 325 | 0 |
| DPN R DPF RB Os | 2,916,192 | 2,833,103 | 717 | 1100 |
| All Classes | 4,055,672 | 3,684,860 | 117,810 | 11,716 |

**Table 4.1:** Amount of LightConnection objects with the specific attribute for all together and each train class separately.

| Train Classes | total | bike | restaurant | night train |
|---|---|---|---|---|
| ICE THA TGV RJ | 3.16% | 0.02% | 57.87% | 0.00% |
| EC IC EX D | 3.15% | 60.11% | 33.28% | 0.22% |
| CNL EN AZ | 1.35% | 12.39% | 0.30% | 18.82% |
| IRE REX RE IR X DPX E Sp | 20.44% | 92.68% | 0.04% | 0.00% |
| DPN R DPF RB Os | 71.90% | 97.15% | 0.02% | 0.04% |
| All Classes | 100% | 90.86% | 2.90% | 0.29% |

**Table 4.2:** Ratio of LightConnection objects with the specific attribute to all LightConnection objects for each train class separately and all classes together. The ”total” column gives the ratio of LightConnections objects of the class to the sum of LightConnection objects of all classes.

For the search with bike transportation, the graph is directly restricted by taking out all connections that do not support it. If the optimal path from source to target does contain connections that do not have the attribute, different paths must be found. Sometimes the results

are a lot worse, for example when faster trains like ICEs can not be used. Table 4.4 shows the result of a search with bike transportation. In comparison the result for the same query without bike transportation is shown in table 4.3. Without bike transportation the fastest journey takes only 380 minutes. But with bike transportation the travel time is increased to 1101 minutes. The problem here is that due to the fact that the fast connection could not be taken, it was not possible to find a journey within one day. Instead, it is necessary to take a CNL that travels respectively slow for multiple hours. Table 4.2 shows the probability for each train class to have the specific attribute (Table 4.1 gives the corresponding total amounts). As we can see, ICEs nearly never allow bike transportation while REs and RBs allow it most of the times.

In some cases even more journeys are found because the optimal path that would dominate others does not have the attribute. In table 4.5 is an example for such a case. Both journeys with bike transportation would have been dominated by the one found that does not have bike transportation in every connection.

| Name | aTime | dTime | TC | Attributes |
|---:|---|---|---|---|
| Hamburg-Altona(S) | 13:31 | 13:31 | S | bike |
| Hamburg-Harburg(S) | 13:55 | 13:55 | S | bike |
| Hamburg-Harburg | 14:02 | 14:05 | ICE | |
| München Hbf | 19:41 | 19:41 | ICE | |
| München Hbf (tief) | 19:51 | | | |

**Table 4.3:** A journey for a query with source station Hamburg-Altona(S) and target station München Hbf(tief). Each row represents an arrival and a departure at the given times with a train of the train class TC and the specified attribute.

| Name | aTime | dTime | TC | Attributes |
|---|---|---|---|---|
| Hamburg-Altona(S) | 13:54 | 13:54 | S | bike |
| Hamburg Hbf (S-Bahn) | 14:08 | 14:08 | S | bike |
| Hamburg Hbf | 14:16 | 16:41 | IRE | bike |
| Berlin Hbf (tief) | 19:35 | 21:35 | CNL | bike |
| Berlin Wannsee | 21:51 | 21:53 | CNL | bike |
| München Hbf | 07:05 | 07:05 | CNL | bike |
| München Hbf (tief) | 07:15 | 07:15 | | |

**Table 4.4:** A journey for a query with source station Hamburg-Altona(S) and target station München Hbf(tief). Additionally, bike transportation was set as a requirement.

### Journeys

To analyse this further, we created 10,000 queries starting from a random station and ending in another random one. The number of queries is equally distributed over the distance between the source and target stations for each query. In Figure 4.1, the number of queries for each number of found journeys is shown. What first catches the eye is that if we look for bike transportation, the likelihood that no connection at all is higher. In this example, over 35% of the queries will

| With bike | | | Without bike | | |
|---|---|---|---|---|---|
| TT | Transfers | Price | TT | Transers | Price |
| 863 | 4 | 20904 | 803 | 4 | 20424 |
| 1040 | 5 | 20748 | | | |

**Table 4.5:** Result of a query with start time 18.09.2015 19:43:00 from Dingden to Attnang-Puchheim. In each row are for all found journeys travel time, transfers, and price listed, on the left for journeys with bike transportation and on the right for the one without.

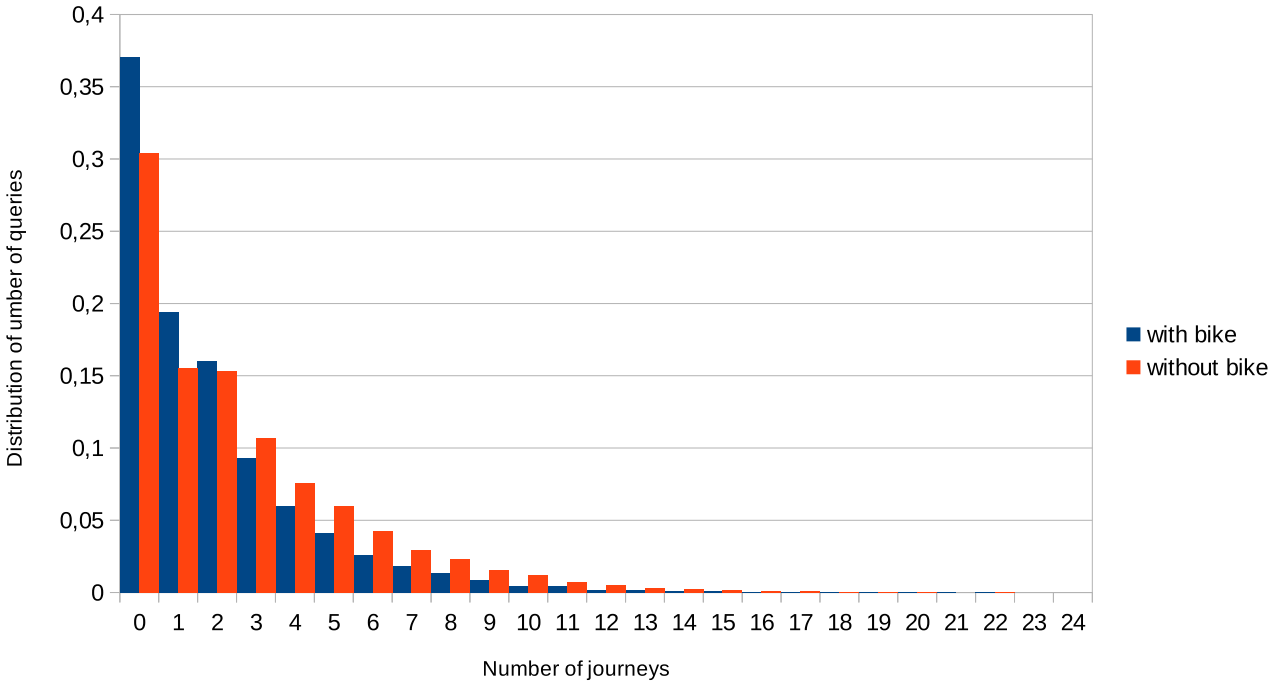


**Figure 4.1:** Shows the distribution of the number of journeys for queries with and without bike transportation.

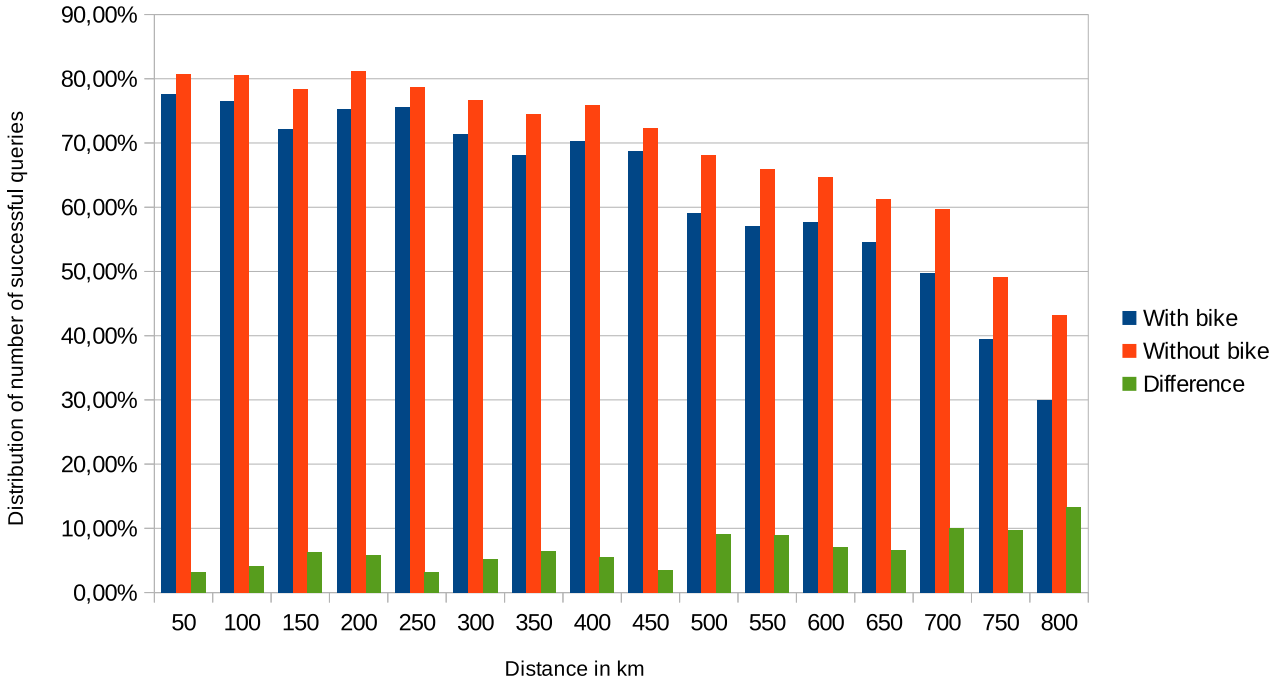


**Figure 4.2:** Shows successful queries regarding the distance with and without bike transportation.

not find any journey between the two stations. If bike transportation is not chosen, under 30% of the queries fail. In general, we can say that the number of found journeys is lower for queries with bike transportation.

## Distance

In this chapter we consider if and how the distance between the source and target station plays a role for the success rate of queries. Figure 4.2 shows the rate of successful queries for queries whose distance of source and target stations are within the given ranges (the first one is 0 to 50 km, the second 50 to 100 and so on). As it would be expected, the probability decreases with the distance. While it is around 80% for distances up to 200 km it sinks to 43.23% for searches without and 29.89% for searches with bike transportation. Furthermore, the success rate of a search with bike transportation seems to be always a bit lower than the one without.

### Search criteria

The following statistics give information on how the search for bike transportation influences the three search criteria travel time, number of transfers, and price. To make the data comparable, only successful queries for a search with bike transportation are used. Those searches are obviously also successful for a search without bike transportation. From all found journeys for a query only the best search criterion was chosen. The goal is to optimize at least one criterion and it is not guaranteed that there is a journey that has optimal values for all mentioned search criteria. This approach is chosen because of the problem that journeys from different queries would not be comparable due to the fact that they are optimized for different criteria. Restricting ourself to one criterion makes the journeys a lot easier to compare.

### Travel time

If the customer wants to travel by bike, he will in average be travelling for 570.33 minutes. If he chooses to travel without bike it takes him 478.39 minutes what is an increase of 19.22%. Figure 4.3 shows the average travel time in minutes of journeys grouped by the distance between source and target station. It is no surprise that the travel time grows for both searches proportionally with the distance. In every case the journeys with only bike transportation take longer. This is because if there is no attribute constraint, the search is not restricted to connections with bike transportation and can therefore find better paths sometimes. For small distances up to 100 km the difference is not too high while it goes up to 100 to 150 minutes for distances above 300 km.

### Transfers

In average it takes 3.09 transfers to travel from source to target station if a bike is carried along. Without a bike it only takes 2.85 transfers. That is an increase of 8.42%. Something similar can be found for the average number of transfers in Figure 4.4. For small ranges bike transportation requires only a few more transfers while the number grows faster for bigger ranges.

### Price

Travelling with a bike costs in average 7.68% more with 12,195.6 compared to travelling without bike that costs 11,325.3. Figure 4.5 shows the average price that needs to be paid for the journey. The graph for the price is very similar to the one for the travel time what is due to the fact that the price is, among others, calculated from the distance. Though it does not grow as fast, because usually the faster trains do not provide bike transportation while the slower cheaper ones do. The statistic only take the best times into account so most likely the expensive fast ones do not appear in the price statistics.

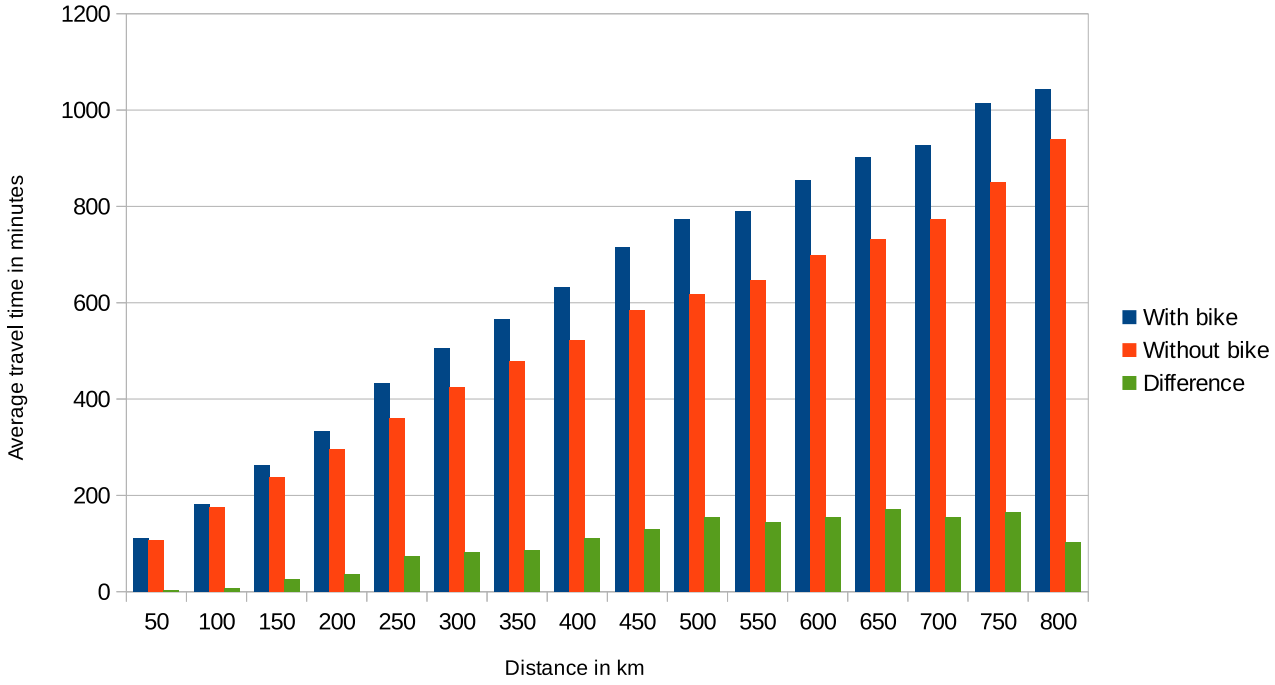


**Figure 4.3:** Shows from the responses from 10,000 queries with and without bike transportation the average travel time. For each set of journeys per query only the smallest travel time is used.

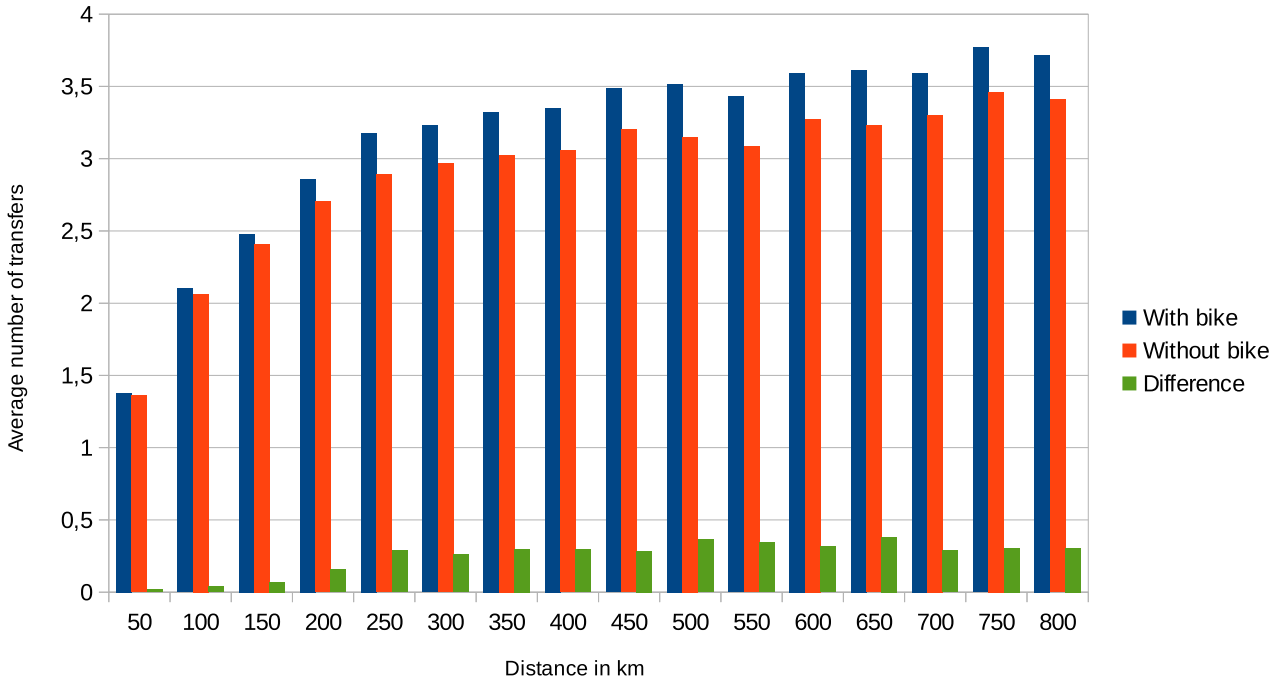


**Figure 4.4:** Shows from the responses from 10,000 queries with and without bike transportation the average number of transfers. For each set of journeys per query only the smallest travel time is used.

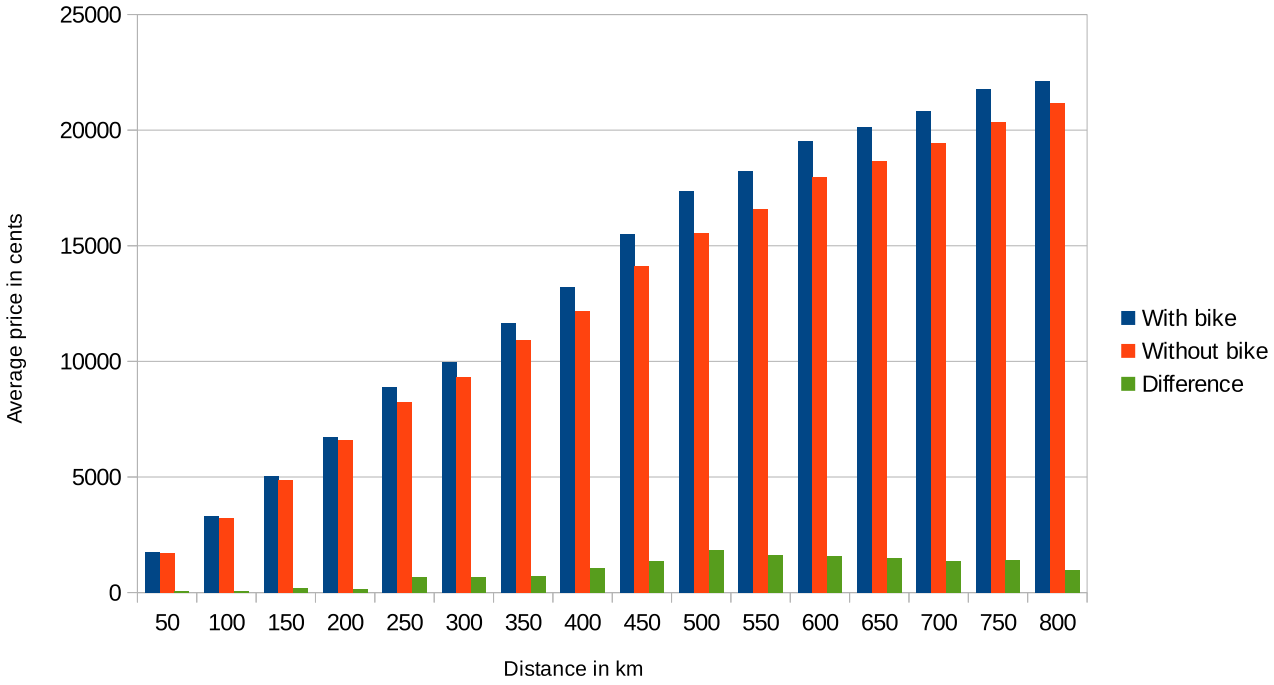


**Figure 4.5:** Shows from the responses from 10,000 queries with and without bike transportation the average travel time. For each set of journeys per query only the smallest price is used.

## 4.2 Temporal Bounded Attributes

In this chapter we analyse certain aspects about the attributes restaurant and night train. We first have a look at how to choose the extended search criteria for attributes to generate realistic queries with a high success rate. After that we analyse the results of queries generated with those parameters.

### 4.2.1 Graph Analysis

When generating queries we need to ask ourself when the customer would probably search for an attribute. In a first step, we can assume that no one would search for a restaurant at night or a night train at day.

#### Restaurant

In Figure 4.6 we can see the distribution of LightConnection objects with the attribute restaurant in the graph for every hour of the day. Again only connections of a type from table 3.2 are considered. While at the day approximately 3.5 % of all trains have the attribute, the number decreases rapidly when reaching 23 o'clock and starts first rising again at 5 o'clock. Already at 7 o'clock 2.5% is reached where it takes 4 hours until 11 o'clock to reach the 3.5% limit in which it sticks during the whole day. The time from 7 to 10 is probably planed for breakfast. The demand is here not as high than lunch and dinner from 11 to 22.

Still, a search for a train with restaurant from 7 to 10 seems reasonable whereas the chances from 11 to 23 are the highest. From 2 to 4 the probability is less than 0.2% what makes it highly unlikely to find a journey with restaurant at night.

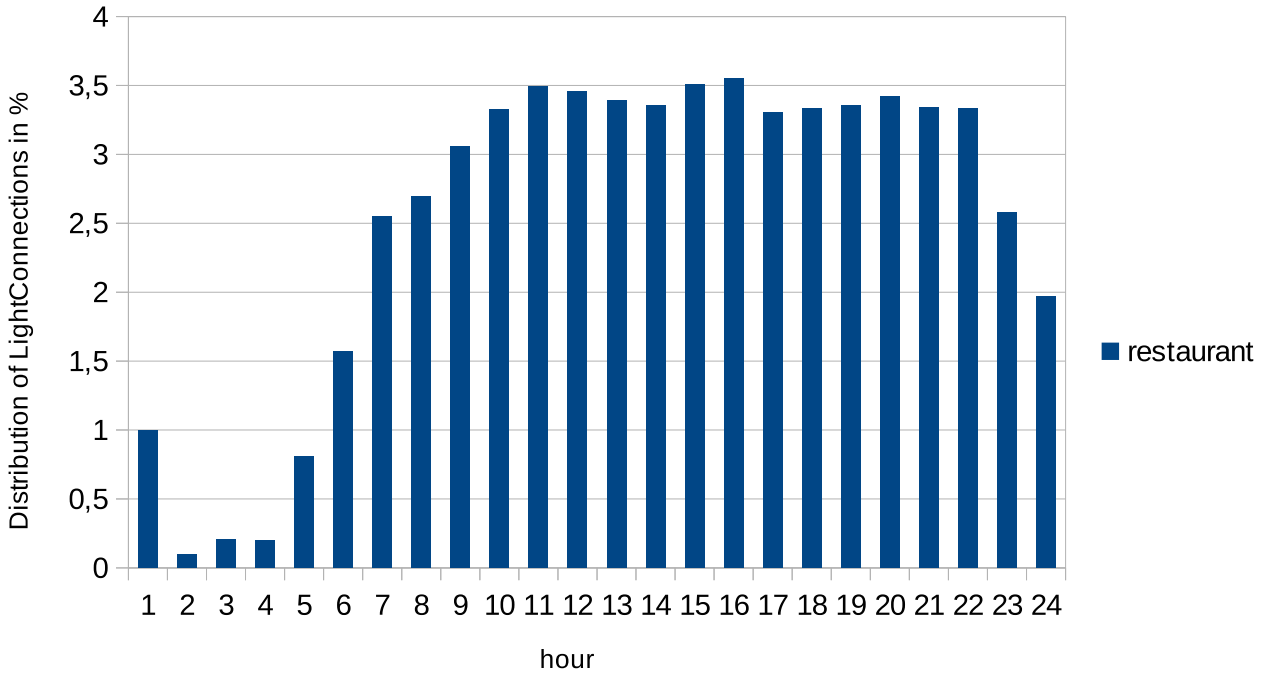


**Figure 4.6:** The restaurant occurrences distributed over the day

### Night train

Figure 4.7 shows the same as Figure 4.6 but for night trains. Here, we have opposite occurrences than for restaurant. During the day nearly no night train is available. Starting at 21 o'clock the distribution of night trains begins to raise slowly. From 1 until 5 it rapidly increases up to 14% and falls within one hour to 2% again.

The high distribution can be explained with the fact that in general at night fewer trains drive. From all those trains that are left of course more trains are night trains. Because of that we should find good results at times starting from 21 until 8 o'clock.

### Remaining Attribute Search Criteria

Table 4.6 gives an overview how the remaining attribute search criteria can be chosen. To include as many customer preferences as possible, we gave unspecific regulations by saying that a random value between two boundaries should be chosen.

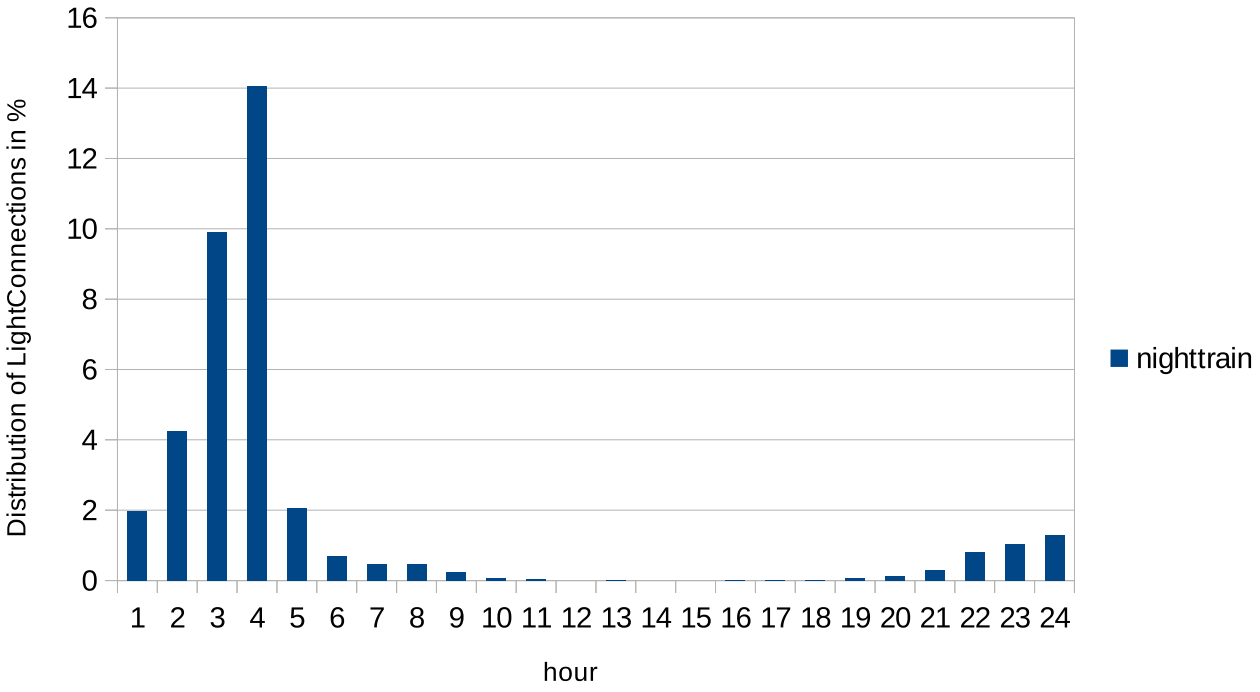


**Figure 4.7:** The night train occurrences distributed over the day

| Attribute | StartTime | EndTime | Duration | MinDuration | ToleranceFactor |
|---|---|---|---|---|---|
| Restaurant | 420 to 1080 | $t_{start}$ + ($d$ to $d$+60) | 30 to 90 | 25 to 45 | 1 |
| Night train | 1260 to 1320 | 420 to 540 | 360 to 480 | 240 | 4 |

**Table 4.6:** The table shows rules on how the extended search criteria for queries with restaurant or night train are chosen. If a range $a$ to $b$ is given then a random number between $a$ and $b$ will be used. $t_{start}$ is the chosen start time for this queries and $d$ the chosen duration.

### 4.2.2 Search Results Analysis

In this chapter we analyse how the additional found journeys that fulfil the attribute requirement perform in comparison to the ones without restrictions. Again, we consider searches with the attributes restaurant and night train. We first look at the results for searches with and without of each attribute and then have a closer look at the effects on travel time, number of transfers, and price for each separately.

| | restaurant | night train |
|---|---|---|
| MinDuration | 74.44% | 74.47% |
| Duration | 62.37% | 42.03% |

**Table 4.7:** Shows the percentage of queries with restaurant or night train that have at least one result with an attribute duration higher than MinDuration or even higher than Duration.

Table 4.7 shows the percentage of successful queries for which $minDuration$ and $duration$ is reached. For 74.44% of all successful queries with the attribute restaurant, at least one journey is in the response that has a attribute duration bigger than the minimum duration, and for 62.37% the attribute duration is equal to the requested attribute duration. To analyse this further, we first have to ask ourself from which factors the found attribute duration time depends. As we have seen for bike transportation, the distance could play an important role. For short distances the search is probably more likely to fail because if the minimum requested attribute duration is higher than the travel time itself circuitous routes have to be taken. Those are obviously not always available or they require a significantly higher travel time. In Figure 4.8 is the probability for finding a journey that has at least the minimum attribute duration for each distance shown. Here, we can see that indeed for bigger distances the probability raises up to 80 to 90%.

At first it seems that at least for the minimum duration for both attributes always nearly equally good results are found. In the next paragraphs we give other factors beside the distance that can influence the success rate for finding journeys with an minimal requested attribute duration. We show that the reasons for the similar values can be completely different.

#### Minimum duration value

The obvious factor that influences the probability of finding journeys that have at least the minimum attribute duration is the $minDuration$ value itself. A higher value is harder to reach. As we can see in table 4.6 restaurant has a minimum duration by at least a factor 5 lower than night train.

#### Tolerance factor

The tolerance factor plays also an important role. All journeys that have a higher travel time, compared to the additional attribute time, are removed from the resulting set. The higher the value, the higher is the probability to find a journey with the minimum attribute duration. A night train query has a 4 times higher tolerance factor.

### Attribute specific factors

Other factors that are hard to measure play also a role, for example the distribution of the attributes on the different train classes from table 4.2. In general trains like ICEs and ICs that provide a restaurant are faster than night trains. Therefore, the fastest journey to the target station has even without an specific attribute search in many cases already restaurant time because ICEs and ICs are used more often.

Without going into more detail we conclude that, although, the results for restaurant and night train at first seem to be very similar, the factors on which both attributes are depending influence them in a totally different way.

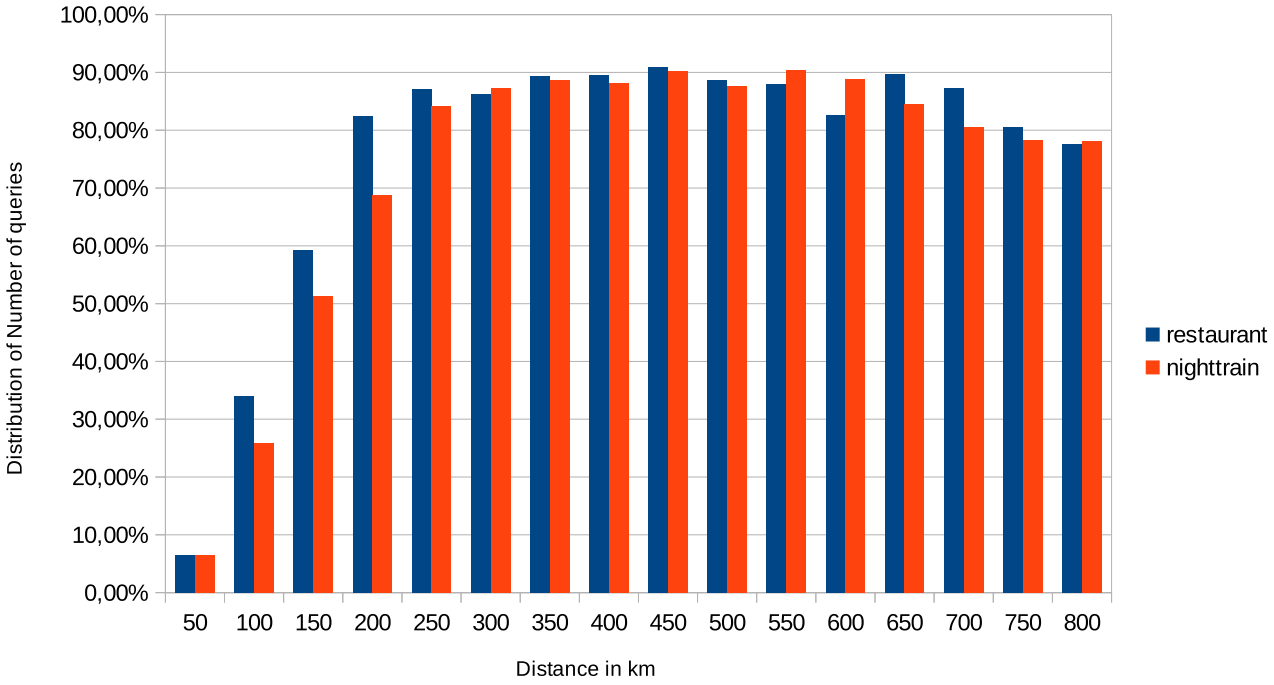


**Figure 4.8:** Shows the distribution of the number of queries which resulting journeys have at least the requested minimum attribute duration with restaurant and with night train over the distance.

## Search Criteria

Here we analyse how good the found journeys that fulfil the attribute constraint are in terms of travel time, number of transfers, and price. For this purpose we created for every attribute for each search criterion a graph that shows, depending on the distance, the average value of the search criterion. The average is calculated with a moving average of order 100. Furthermore, the moving average for the difference between both searches is shown. For each graph we used 20,000 separately created queries, 10,000 with and 10,000 without attribute constrain.

### Restaurant

We first have a look at the responded journeys from queries with the attribute restaurant. The figures 4.9, 4.10, and 4.11 show the moving average of travel time, number of transfers, and

price for each distance. In table 4.8 are the average values and the increase rate. The search criteria are the minimum values of each journey for each query. We only used queries in which the attribute duration is above the minimum attribute duration. This filters the for this statistic irrelevant values out.

As we can see in Figure 4.9, the travel time is unaffected by a restaurant search. Both graphs are nearly identical. Just a slight tendency of the search with attribute being always a bit higher than the one without can be seen.

For the number of transfers and price in Figure 4.10 and 4.11 both searches are again equal with the exception that the difference is higher. While the difference of the number of transfers is relatively constant for every distance, we can see for the price a slight linear regression over distance. This can be explained by the fact that ICEs and ICs that provide restaurants have a one time surcharge. For bigger distances that surcharge gets compensated by the higher prices. It should be noted that other factors play a role as well, like the fact that train classes like RB and RE are cheaper for short distances because of better local linkages. An ICE can be cheaper than a RB when the RB needs to drive inconvenient paths that are in sum longer than the direct one the ICE drives.

The nearly identical travel times make the search results very attractive because an on-board restaurant brings not only comfort but also the opportunity to save the time of taking a meal.

Table 4.9 shows an example of the resulting journeys for a query with on-board restaurant. The search with restaurant finds not only the results of the search without but also one additional connection. This one is only 10 cents more expensive but completely provides the requested 55 minutes restaurant time.

| | with restaurant | without restaurant | increase rate |
|---|---|---|---|
| avg. travel time | 518.86 | 513.96 | 0.95% |
| avg. transfers | 3.23 | 3.02 | 6.81% |
| avg. price | 14312.00 | 13071.40 | 9.49% |

**Table 4.8:** Average values for travel time in minutes, number of transfers, and price in cents of the results of a search with 10,000 queries with and without restaurant

| With restaurant | | | | Without restaurant | | | |
|---|---|---|---|---|---|---|---|
| TT | Transfers | Price | Attribute duration | TT | Transfers | Price | Attribute duration |
| 936 | 4 | 21478 | 0(0%) | 936 | 4 | 21478 | 0(0%) |
| 936 | 4 | 21488 | 55(100%) | 936 | 5 | 21395 | 0(0%) |
| 936 | 5 | 21395 | 0(0%) | 957 | 5 | 20038 | 0(0%) |
| 957 | 5 | 20038 | 0(0%) | 1003 | 5 | 19798 | 0(0%) |
| 1003 | 5 | 19798 | 0(0%) | - | - | - | - |

**Table 4.9:** Resulting journeys of two queries, one with restaurant one without, that looked for a connection from Wutöschingen to Groß Ammensleben. Each journey has a travel time (TT), transfers, price in cents, and an attribute duration. The attribute search properties are the following: duration = 55 min, minDuration = 32 min, toleranceFactor = 1.

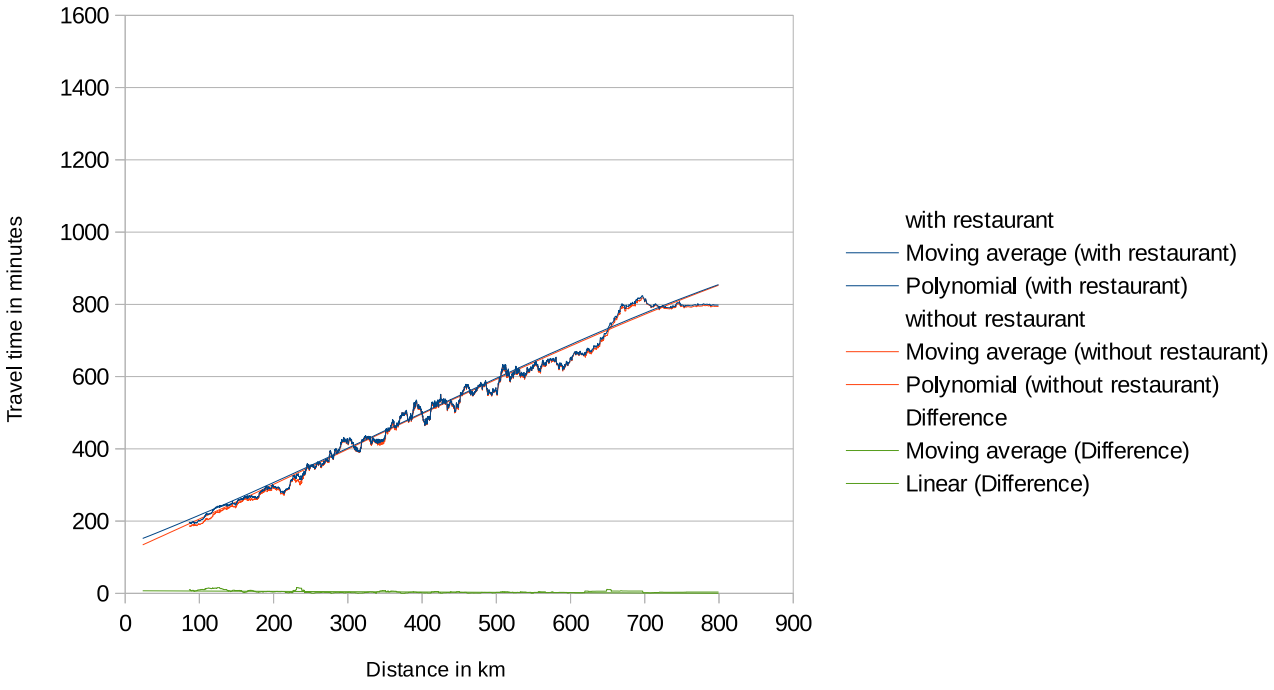


**Figure 4.9:** The travel time of journeys from 10,000 queries with and without restaurant over the distance

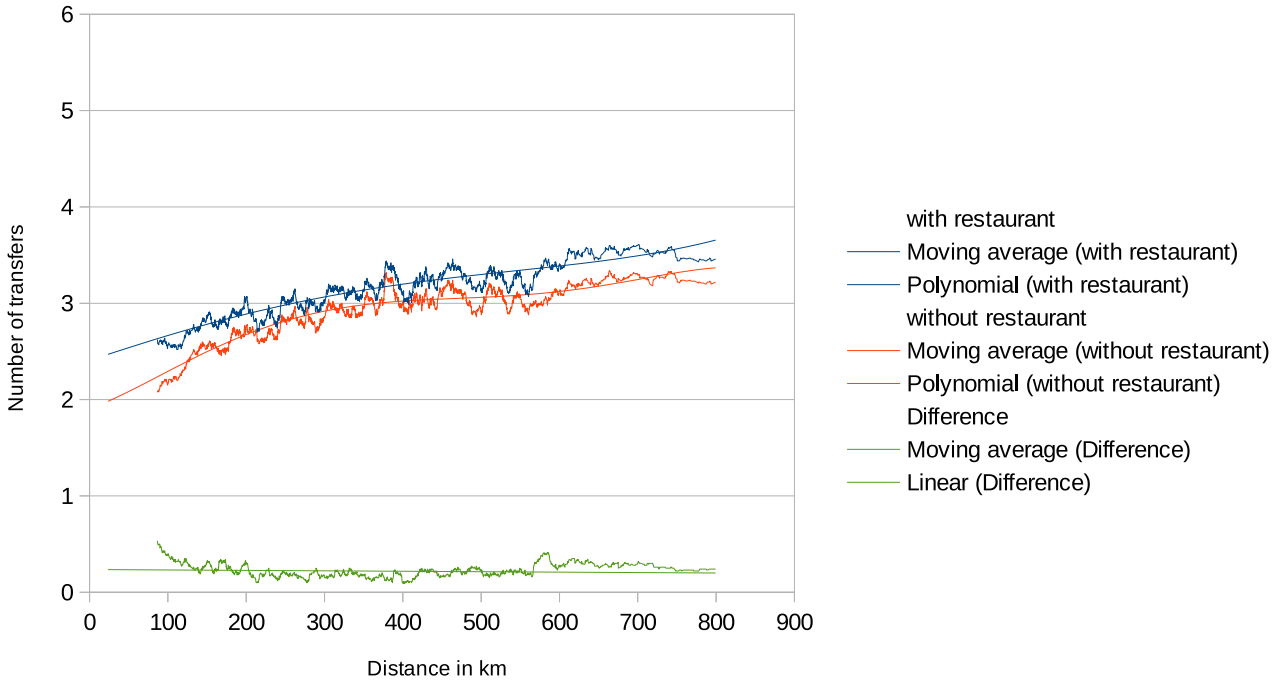


**Figure 4.10:** The number of transfers of journeys from 10,000 queries with and without restaurant over the distance

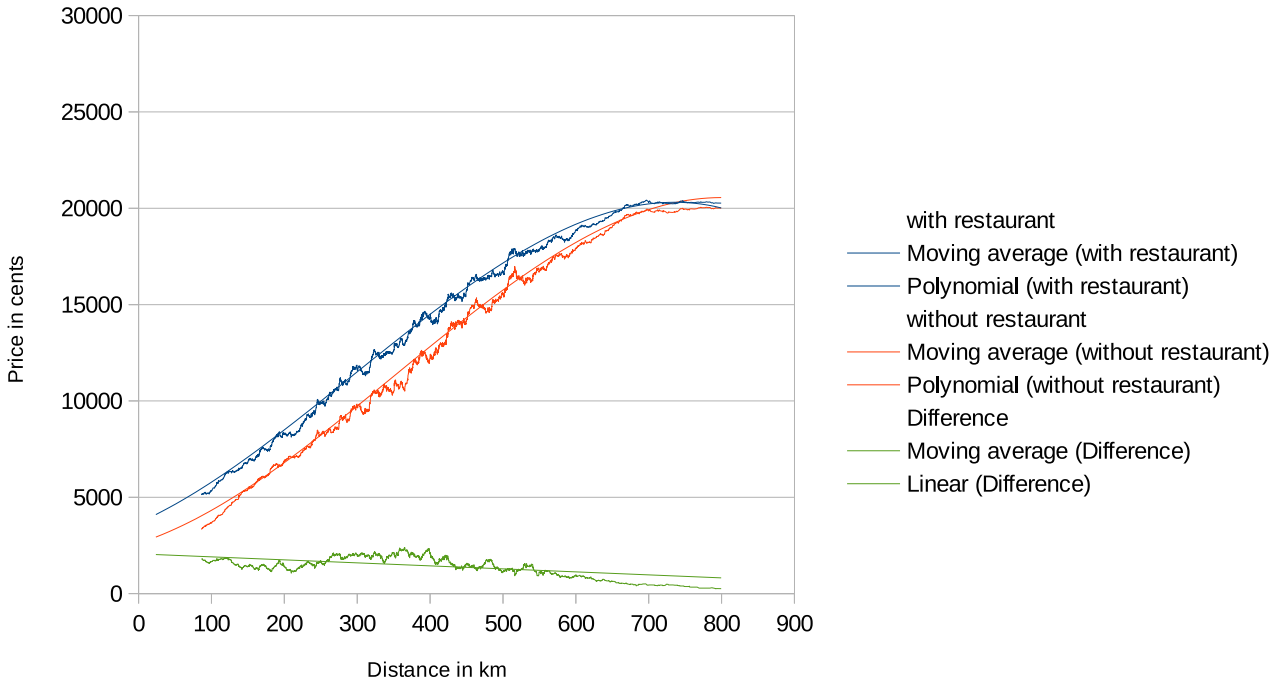


**Figure 4.11:** The price of journeys form 10,000 queries with and without restaurant over the distance

### Night train

Now we analyse the same for night trains. The figures 4.12, 4.13, and 4.14 show again the travel time, number of transfers, and price of a search with and without night train for each distance and table 4.12 respectively shows the average for all values.

When we compare the average increase of travel time, number of transfers, and price to the one of the other attributes we see that the trade off for the customer may be worse for this attribute. But we have to keep in mind that the service has to be available for a much longer time and has probably more importance to the customer than for example an on-board restaurant. Therefore a customer probably bears a not as good of a trade-off.

The first distinction is that the difference for the travel time is a lot bigger. The higher travel time up until 400 km can be explained by the fact that not nearly enough night trains are available to find direct journeys that do not require circuitous routes. After that point we can see a constant approach of both graphs what can be explained by the fact that probably only night trains travel bigger distances at night.

The number of transfers from Figure 4.13 shows a similar behaviour.

The price in Figure 4.14 follows the same trend as travel time and number of transfers but in a more radical fashion. The difference for short distances is a lot higher but therefore it adjusts faster. The customer need to pay for the higher distance that is caused by the circuitous route that needs to be taken.

In Table 4.10 we see an example where the last two journeys of the query with night train search have a much higher travel time than the others. The last journey even reaches the preferred attribute duration. In Table 4.11 a better example is given. The customer has to travel 360 minutes longer, has an additional transfer, and must pay roughly 30 Euros more but therefore has 385 minutes night train time.

| With night train | | | | Without night train | | | |
|---|---|---|---|---|---|---|---|
| TT | Transfers | Price | Attribute duration | TT | Transfers | Price | Attribute duration |
| 177 | 2 | 7601 | 0 | 177 | 2 | 7601 | 0 |
| 177 | 3 | 7010 | 0 | 177 | 3 | 7010 | 0 |
| 213 | 3 | 6021 | 0 | 213 | 3 | 6021 | 0 |
| 213 | 4 | 5949 | 0 | 213 | 4 | 5949 | 0 |
| 816 | 5 | 20528 | 404 | - | - | - | - |
| 848 | 5 | 20784 | 429 | - | - | - | - |

**Table 4.10:** Resulting journeys of two queries, one with night train one without, that looked for a connection from Bad Sooden-Allendorf to Karlstadt(Main). Each journey has a travel time (TT), transfers, price in cents, and an attribute duration. The attribute search properties are the following: duration = 429 min, minDuration = 240 min, toleranceFactor = 4.

| With night train | | | | Without night train | | | |
|---|---|---|---|---|---|---|---|
| TT | Transfers | Price | Attribute duration | TT | Transfers | Price | Attribute duration |
| 1013 | 4 | 22104 | 0(0%) | 1013 | 4 | 22104 | 0(0%) |
| 1373 | 5 | 24984 | 385(91%) | - | - | - | - |

**Table 4.11:** Resulting journeys of two queries, one with night train one without, that looked for a connection from from Eichen(Kr Siegen) to Sluknov. Each journey has a travel time (TT), transfers, price in cents, and an attribute duration. The attribute search properties are the following: duration = 421 min, minDuration = 240 min, toleranceFactor = 4.

| | with nighttrain | without nighttrain | increase rate |
|---|---|---|---|
| avg. travelTime | 809.72 | 692.63 | 16,90% |
| avg. transfers | 3.60 | 3.30 | 8,97% |
| avg. price | 18832.20 | 15250.30 | 23,49% |

**Table 4.12:** Average values for travel time in minutes, number of transfers, and price in cents of the results of a search with 10,000 queries with and without night train

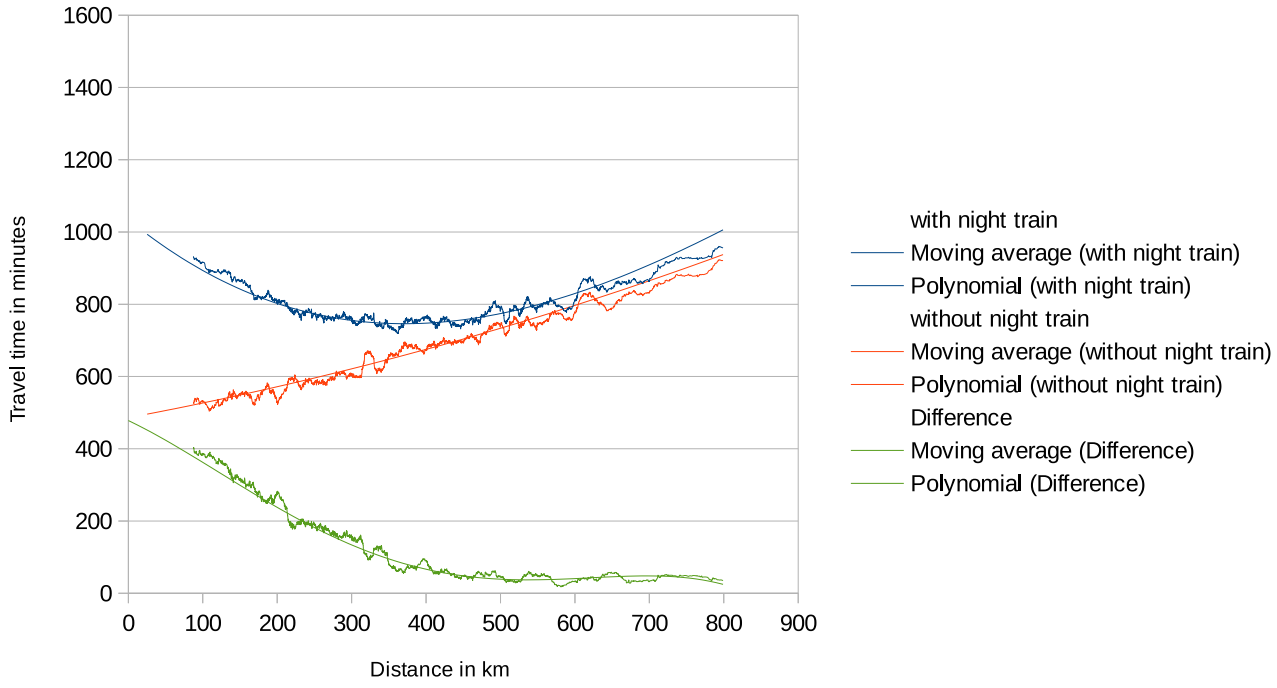


**Figure 4.12:** The travel time of journeys from 10,000 queries with and without night train over the distance

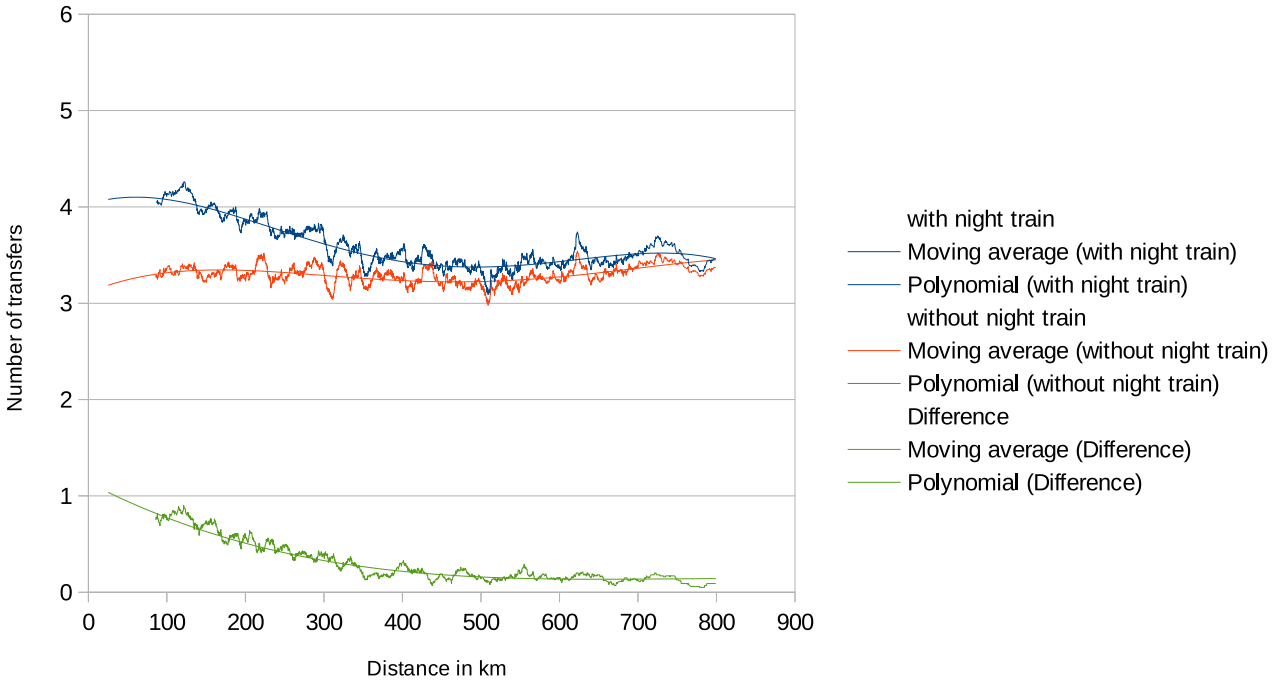


**Figure 4.13:** The number of transfers of journeys from 10,000 queries with and without night train over the distance

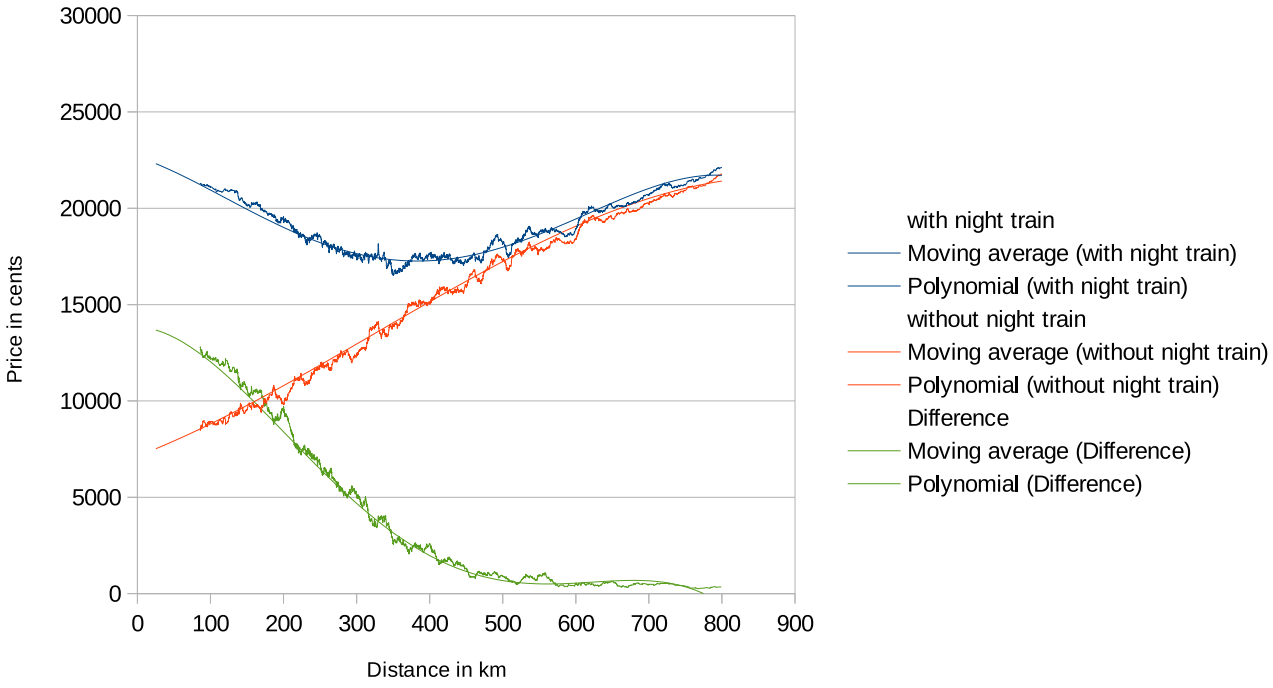


**Figure 4.14:** The price of journeys from 10,000 queries with and without night train over the distance

## 4.3 Runtime analysis

For the following computations we use an Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz (4 CPUs). To make use of all four cores, parallel computing with 4 threads is activated. We generated for each attribute 10,000 queries. To compare the runtime we additionally created the same attributes but without any attribute constraints. The distance of source and target station is up to 800 km equally distributed, otherwise the stations are randomly generated.

Table 4.15 shows the, per query, average resulting search times in ms for each search with and without attributes. The table is split into 3 parts. The overall average search time is the total average search time for a query. The search consists of a preprocessing part where the lower bounds graphs are initialized and the actual multi criteria search. The relevant search time is the average search time for each query but without the lower bound preprocessing. Factor simply shows for overall and relevant by which factor the search time of a search without attribute constraint changes to the one with. The three Figures 4.16, 4.17, and 4.18 show the average relevant search time over the distance for each search with and without attribute.

The reason for deferring between overall and relevant search time is because, as the name indicates, the preprocessing is in no way affected by the modifications in this thesis and is therefore not directly relevant. It is the most time consuming part so the overall search time changes are not that big.

### 4.3.1 Bike Transportation

It is no surprise that a search for bike transportation reduces the search time. After all, the search is now executed on a graph with only edges with at least one train that supports bike transportation. Although, as we have seen in Section 4.1, most of trains support it, all the other transportation classes often do not. Queries with source and target station, that are closer to each other, require less search time. Here, the difference between queries with and without attribute is also not that big. At higher distances the difference increases but we observe that the search time does not increase anymore from distances of 500 to 600 km.

### 4.3.2 Indispensable Attributes

The indispensable attributes restaurant and night train do always add search time because the search finds additionally to the journeys without attribute the one with. In average the relevant search time is more than doubled.

The search times for queries with restaurant are for small distances equally high and then diverge constantly. Here, the success rate from Figure 4.8 matters again. Because the tolerance factor and the duration between start and end time, in which the attribute should occur, is smaller, labels are dominated earlier by which potential unnecessary journeys are excluded earlier.

For queries with night trains the search time behaves for small distances differently as the difference to the queries without attribute is already high. Because of a higher tolerance factor and wider ranges between start and end time more journeys are possible that use bigger circuitous routes. And because its less likely to find a really good journey with a high attribute duration and small travel time, it is not that easy to dominate worse labels. For bigger distances the difference is not that high as it is for searches with restaurant.

In general, strangely queries with restaurant require a lot more calculation time than those with night trains. One explanation is that the queries with night trains, search for trains at night when fewer trains drive. Although, at first the same amount of edges are inspected, the higher waiting times increase the travel time more what results faster in a domination of the label. Because the searches without attributes are differing in a same way, we assume that this is an attribute independent effect.

| | Overall avg. search time | | Relevant avg. search time | | Factor | |
|---|---|---|---|---|---|---|
| Attribute | With attribute | Without attribute | With attribute | Without attribute | Overall | Relevant |
| restaurant | 1,852.75 | 1,614.84 | 381.89 | 190.90 | -14.73% | -100.05% |
| nighttrain | 1,660.02 | 1,533.27 | 221.26 | 104.67 | -8.27% | -111.39% |
| bike | 1,522.40 | 1,611.04 | 83.51 | 186.33 | 5.50% | 55.18% |

**Figure 4.15:** Average search time in ms of 10,000 queries for each attribute

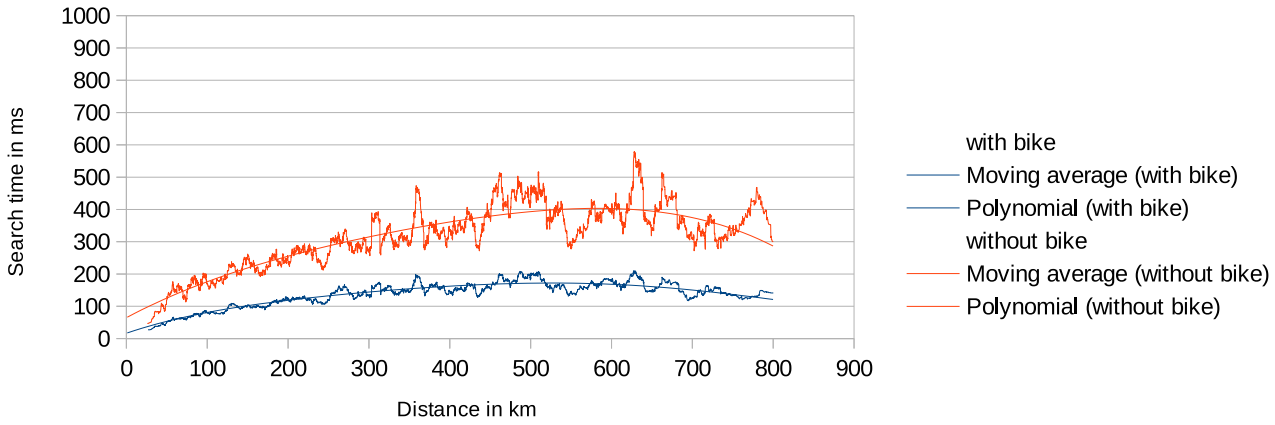


**Figure 4.16:** Search time for 10,000 queries with and without bike dependent of the distance

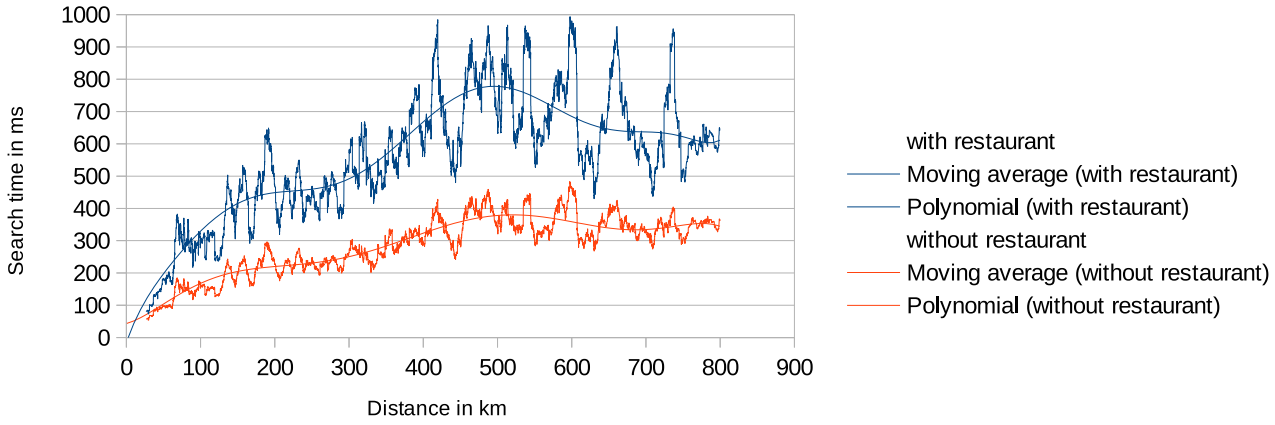


**Figure 4.17:** Search time for 10,000 queries with and without restaurant dependent of the distance

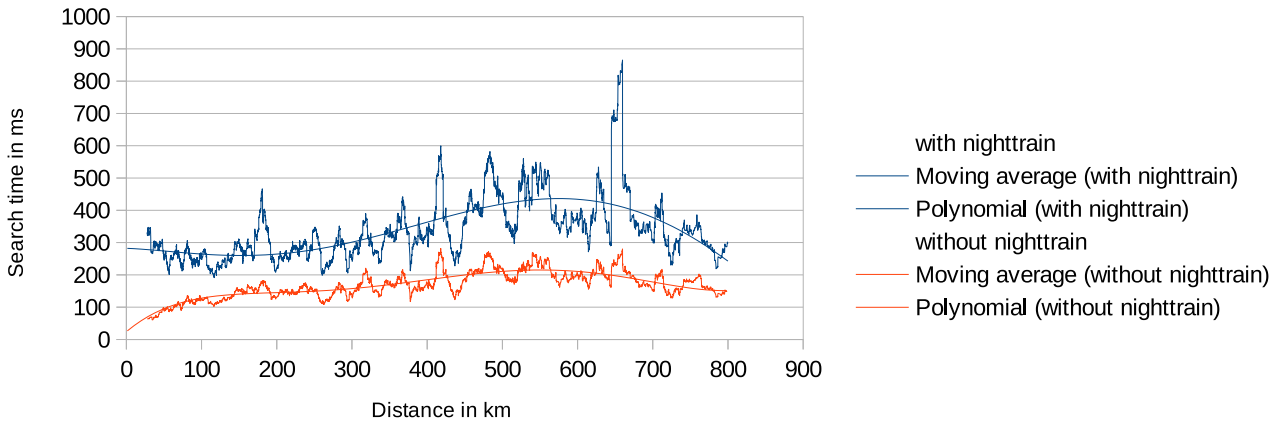


**Figure 4.18:** Search time for 10,000 queries with and without night train dependent of the distance

# 5 Conclusion

In this thesis, we successfully integrated attributes as a new search criterion in queries for timetable information. We could validate that the extended search finds indeed under the given conditions still the optimal journeys. This applies for both attribute search models.

We assessed that a search with bike transportation fails in average 5% to 10% more often then a search without. Furthermore, the number of transfers and the price are increasing by 7% to 8% and the travel time by 18% to 19%. Searches with restaurant had a slightly higher travel time while the number of transfers and the price increase by 7% to 10%. The biggest trade-off was the one for a night train search. Here, the increase for travel time, number of transfers, and price is 16.9%, 8.97%, and 23.49% respectively. We can assume that in many situations the compromise that has to be made by the customer is acceptable.

For bike transportation and on-board restaurant the trade-off for travel time, number of transfers, and price is unaffected by the distance of source and target station. An improvement can therefore not be gained by selecting stations that are farer or nearer to each other. For night train we had a different situation. Here, queries with source and target station that are further afar lead indeed to results that have in average not as high travel time and price.

The runtime analysis has shown that the relevant search time for temporal bounded attributes is doubled. But still, the overall time was increased by 10 to 15% what sounds justifiable considering the time the customer saves because he does not have to look manually. We saw that over the distance the additionally required search time reduces constantly to a point where no extra search time is required. For indispensable attributes the relative search time is even halved. Therefore, the scalability is given.

In total we can say that the approach in this thesis is an useful extension for a time table system because it gives the customer the ability to easily specify his search and fit it better to his needs while the search time is still within reasonable limits. As we have seen, the results deliver often an attractive trade-off, what makes this approach also useful in practice.

## Bibliography

[Dis07] Yann Disser. Multi-criteria search for optimal train connections using the time-dependent graph model. Master's thesis, TU Darmstadt, 2007.

[GSMH11] Thorsten Gunkel, Mathias Schnee, and Matthias Müller-Hannemann. How to find good night train connections. *Networks*, 57(1):19–27, 2011.

[Gü15] Felix Gündling. Modelling and speedup techniques for an intermodal travel information system. Master's thesis, TU Darmstadt, 2015.

## List of Figures

## List of Tables

## List of Algorithms