Multi-Modal Route Planning in Road and Transit Networks

Master's Thesis

Daniel Tischner

University of Freiburg, Germany,

daniel.tischner.cs@gmail.com

August 8, 2018

Supervisor: Prof. Dr. Hannah Bast
Advisor: Patrick Brosi

Contents

1 Introduction **6**

2 Preliminaries **7**
- 2.1 Graph . . . 7
- 2.2 Tree . . . 9
- 2.3 Metric . . . 9

3 Models **12**
- 3.1 Road graph . . . 12
- 3.2 Transit graph . . . 14
- 3.3 Link graph . . . 17
- 3.4 Timetable . . . 19

4 Nearest neighbor problem **21**
- 4.1 Cover tree . . . 23

5 Shortest path problem **28**
- 5.1 Time-independent . . . 28
  - 5.1.1 Dijkstra . . . 28
  - 5.1.2 A* and ALT . . . 28
- 5.2 Time-dependend . . . 28
  - 5.2.1 Connection scan . . . 28
- 5.3 Multi-modal . . . 29
  - 5.3.1 Modified Dijkstra . . . 29
  - 5.3.2 Access nodes . . . 29
- 5.4 Other algorithms . . . 29

6 Evaluation **29**
- 6.1 Input data . . . 29
- 6.2 Experiments . . . 29
  - 6.2.1 Nearest neighbor computation . . . 29
  - 6.2.2 Uni-modal routing . . . 29
  - 6.2.3 Multi-modal routing . . . 29
- 6.3 Summary . . . 29

7 Conclusion **29**

References **30**

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date

Signature

Zusammenfassung

Wir präsentieren Algorithmen für multi-modale Routenplannung in Straßennetzwerken und Netzwerken des öffentlichen Personennahverkehrs (ÖPNV), so wie in kombinierten Netzwerken.

Dazu stellen wir das Nächste-Nachbar und das Kürzester-Pfad Problem vor und schlagen Lösungen basierend auf Cover Trees, ALT und CSA vor.

Des Weiteren erläutern wir die Theorie hinter den Algorithmen, geben eine kurze Übersicht über andere Techniken, zeigen Versuchsergebnisse auf und vergleichen die Techniken untereinander.

Abstract

We present algorithms for multi-modal route planning in road and public transit networks, as well as in combined networks.

Therefore, we explore the nearest neighbor and shortest path problem and propose solutions based on COVER TREEs, ALT and CSA.

Further, we illustrate the theory behind the algorithms, give a short overview of other techniques, present experimental results and compare the techniques with each other.

TODO: disable todos and macro highlights.

1 Introduction

Route planning refers to the problem of finding an *optimal* route between given locations in a network. With the ongoing expansion of road and public transit networks all over the world route planner gain more and more importance. This led to a rapid increase in research [3, 9, 15] of relevant topics and development of route planner software [13, 12, 21].

However, a common problem of most such services is that they are limited to one transportation mode only. That is a route can only be taken by a car or train but not by both at the same time. This is known as uni-modal routing. In contrast to that multi-modal routing allows the alternation of transportation modes. For example a route that first uses a car to drive to a train station, then a train which travels to a another train station and finally using a bicycle from there to reach the destination.

The difficulty with multi-modal routing lies in most algorithms being fitted to networks with specific properties. Unfortunately, road networks differ a lot from public transit networks. As such, a route planning algorithm fitted to a certain type of network will likely yield undesired results, have an impractical running time or not even be able to be used at all on different networks. We will explore this later in **Section 6**.

In this thesis we explore a technique with which we can combine an algorithm fitted for road networks with an algorithm for public transit networks. Effectively obtaining a generic algorithm that is able to compute routes on combined networks. The basic idea is simple, given a source and destination, both in the road network, we select *access nodes* for both. This are nodes where we will switch from the road into the public transit network. A route can then be computed by using the road algorithm for the source to its access nodes, the transit algorithm for the access nodes of the source to the access nodes of the destination and finally the road algorithm again for the destinations access nodes to the destination. Note that this technique might not yield the shortest possible path anymore. Also, it does not allow an arbitrary alternation of transportation modes. However, we accept those limitations since the resulting algorithm is very generic and able to compute routes faster than without limitations. We will cover this technique in detail in **Section 5.3.2**.

Our final technique uses a modified version of ALT [14] as road algorithm and CSA [10] for the transportation network. The algorithms are presented in **Section 5.1.2** and **Section 5.2.1** respectively. We also develop a multi-modal variant of DIJKSTRA [7] which is able to compute the shortest route in a combined network with the possibility of changing transportation modes arbitrarily. It is presented in **Section 5.3.1** and acts as baseline to our final technique based on access nodes.

We compute access nodes by solving the nearest neighbor problem. For a given node in the road network its access nodes are then all nodes in the transit network which are in the *vicinity* of the road node. We explore a solution to this problem in **Section 4**.

Section 3 starts by defining types of networks. We represent road networks by graphs only. For transit networks we provide a graph representation too. Both graphs can then be combined into a linked graph. The advantage of graph based models is that they are well studied and therefore we are able to use our multi-modal variant of DIJKSTRA to compute routes on them. However, we also propose a non-graph based representation for transit networks, a timetable. The timetable is used by CSA, an efficient algorithm for route planning on public transit networks. With that, our road and transit networks get incompatible and can not easily be combined. Therefore, we use the previously mentioned generic approach based on access nodes for this type of network.

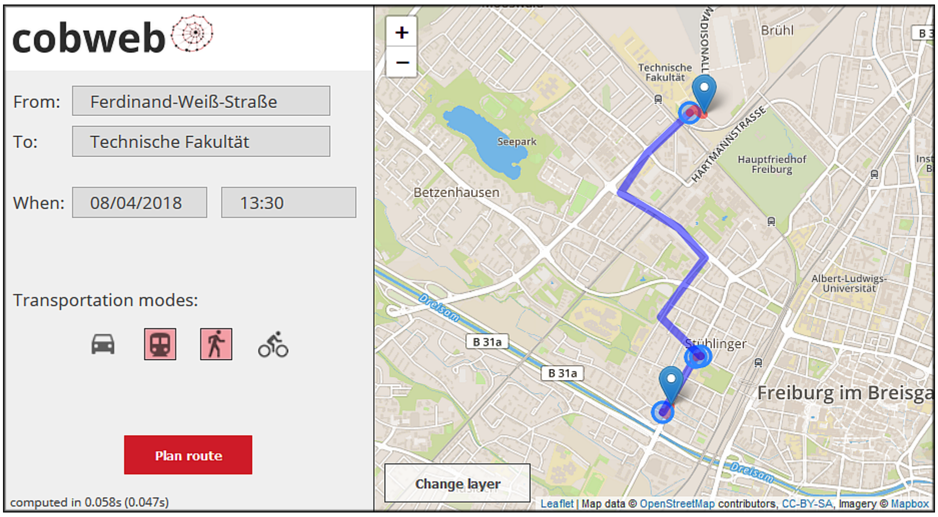

Fig. 1: Screenshot of COBWEBs [19] frontend, an open-source multi-modal route planner. It shows a multi-modal route starting from a given source, using the modes *foot-tram-foot-tram-foot* in that sequence to reach the destination.

Further, we implemented the presented algorithms in the COBWEB [19] project, which is an open-source multi-modal route planner (see **Fig. 1** for an image of its frontend). In **Section 6** we show our experimental results and compare the techniques with each other.

2 Preliminaries

Before we define our specific data models and problems we will introduce and formalize commonly reoccurring terms.

2.1 Graph

Definition 1. *A* graph G *is a tuple* (V, E) *with a set of nodes* V *and a set of edges* $E \subseteq V \times \mathbb{R}_{\geq 0} \times V$*. An* edge $e \in E$ *is an ordered tuple* (u, w, v) *with source node* $u \in V$*,* *a non-negative weight $w \in \mathbb{R}_{\geq 0}$ and a destination node $v \in V$.*

Note that **Definition 1** actually defines a *directed* graph, as opposed to an *undirected* graph where an edge like (u, w, v) would be considered equal to the edge of opposite direction (v, w, u) (compare to [11]). However, for transportation networks an undirected graph often is not applicable, for example due to one way streets or time dependent connections like trains which depart at different times for different directions.

In the context of route planning we refer to the weight w of an edge (u, w, v) as *cost*. It can be used to encode the length of the represented connection. Or to represent the time it takes to travel the distance in a given transportation mode.

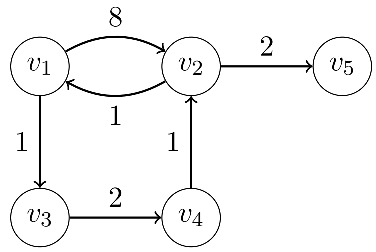

Fig. 2: Illustration of an example graph with five nodes and six edges.

As an example consider the graph $G = (V, E)$ with

$$V = \{v_1, v_2, v_3, v_4, v_5\} \text{ and}$$
$$E = \{(v_1, 8, v_2), (v_1, 1, v_3), (v_2, 1, v_1), (v_2, 2, v_5), (v_3, 2, v_4), (v_4, 1, v_2)\}.$$

which is illustrated by **Fig. 2**.

Definition 2. *Given a graph $G = (V, E)$ the function* src $: E \to V, ((u, w, v)) \mapsto u$ *gets the* source *of an edge. Analogously* dest $: E \to V, ((u, w, v)) \mapsto v$ *retrieves the* destination*.*

Definition 3. *A* path *in a graph $G = (V, E)$ is a sequence $p = e_1 e_2 e_3 \ldots$ of edges $e_i \in E$ such that*

$$\forall i : \text{dest}(e_i) = \text{src}(e_{i+1}).$$

The length *of a path is the amount of edges it contains, i.e. the length of the sequence. The* weight *or* cost *is the sum of its edges weights.*

An example for a path in the graph G would be

$$p = (v_1, 8, v_2)(v_2, 1, v_1)(v_1, 1, v_3).$$

Its length is 3 and it has a weight of 10.

2.2 Tree

Definition 4. *A* tree *is a graph $T = (V, E)$ with the following properties:*

1. *There is exactly one node $r \in V$ with no ingoing edges, called the* root*, i.e.*

$$\exists! r \in V \nexists e \in E : \mathrm{dest}(e) = r.$$

2. *All other nodes v have exactly one ingoing edge. The source p of this edge is called* parent *of v and v is called* child *of p:*

$$\forall v \in V : v \neq r \Rightarrow \exists! e \in E : \mathrm{dest}(e) = v.$$

Definition 5. *The* subtree *of a tree $T = (V, E)$ rooted at a node $r' \in V$ is a tree $T' = (V', E')$. $V' \subseteq V$ is the set of nodes that can be reached from r'. That is, all nodes that are part of possible paths starting at r'. Likewise $E' \subseteq E$ is the set of edges restricted to the vertices in V'. The root of T' is r'.*

Definition 6. *The* depth *of a node v in a tree $T = (V, E)$, denoted by* $\mathrm{depth}(v)$*, is defined as the amount of edges between the v and the root r. It is the length of the unique path p starting at r and ending at v.*
The height *of a tree is its greatest depth, i.e.*

$$\max_{v \in V} \mathrm{depth}(v).$$

And

$$\mathrm{children}(v) = \{c \in T | c \textit{ child of } v\}.$$

Trees are hierarchical data structures. Every node, except the root, has one parent. A node itself can have multiple children. Note that it is not possible to form a loop in a tree, i.e. a path that visits a node more than once. A node without children is called a *leaf*. TODO: Maybe add some illustrations here...

2.3 Metric

Definition 7. *A function $d : M \times M \to \mathbb{R}$ on a set M is called a* metric *iff for all $x, y, z \in M$*

$$\begin{aligned} d(x,y) &\geq 0, && \textit{non-negativity} \\ d(x,y) = 0 &\Leftrightarrow x = y, && \textit{identity of indiscernibles} \\ d(x,y) &= d(y,x) \textit{ and} && \textit{symmetry} \\ d(x,z) &\leq d(x,y) + d(y,z) && \textit{triangle inequality} \end{aligned}$$

holds.

Definition 8. *A* metric space *is a pair* (M, d) *where* M *is a set and* $d : M \times M \to \mathbb{R}$ *a metric on* M*.*

Definition 9. *Given a metric* d *on a set* M*, the distance of a point* $p \in M$ *to a subset* $Q \subseteq M$ *is defined as the distance from* p *to its nearest point in* Q*:*

$$d(p, Q) = \min_{q \in Q} d(p, q)$$

A metric is used to measure the distance between given locations. **Section 4** and **Section 5**, in particular **Section 5.1.2**, will make heavy use of this term.

There we measure the distance between geographical locations given as pair of *latitude* and *longitude* coordinates. Latitude and longitude, often denoted by ϕ and λ, are real numbers in the ranges $(-90, 90)$ and $[-180, 180)$ respectively, measured in degrees. However, for convenience we represent them in radians. Both representations are equivalent to each other and can easily be converted using the ratio $360° = 2\pi$ rad.

A commonly used measure is the *as-the-crow-flies* metric, which is equivalent to the euclidean distance in the euclidean space. **Definition 10** defines an approximation of this distance on locations given by latitude and longitude coordinates. The approximation is commonly known as equirectangular projection of the earth [16]. Note that there are more accurate methods for computing the great-circle distance for geographical locations, like the haversine formula [17]. However, they come with a significant computational overhead.

Definition 10. *Given a set of coordinates* $M = \left\{(\phi, \lambda) | \phi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \lambda \in [-\pi, \pi)\right\}$ *we define* asTheCrowFlies : $M \times M \to \mathbb{R}$ *such that*

$$((\phi_1, \lambda_1), (\phi_2, \lambda_2)) \mapsto \sqrt{\left((\lambda_2 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_2}{2}\right)\right)^2 + (\phi_2 - \phi_1)^2} \cdot 6371000.$$

As a next step we prove that asTheCrowFlies is indeed a metric on the set of coordinates.

Proposition 1. *The function* asTheCrowFlies *is a metric on its domain* M*.*

Proof. We need to prove that all four axioms hold. Let us first set

$$x = (\lambda_2 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_2}{2}\right)$$
$$y = \phi_2 - \phi_1$$

then the function simplifies to

$$\sqrt{x^2 + y^2} \cdot 6371000.$$

Obviously this can never resolve to a negative number since

$$\underbrace{\underbrace{\sqrt{\underbrace{x^2}_{\geq 0}+\underbrace{y^2}_{\geq 0}}}_{\geq 0}\cdot 6371000}_{\geq 0}.$$

For the second axiom we assume that $\text{asTheCrowFlies}\left((\phi_1,\lambda_1),(\phi_2,\lambda_2)\right)=0$ for an arbitrary pair of coordinates and follow

$$\begin{aligned}
&\sqrt{\left((\lambda_2-\lambda_1)\cdot\cos\left(\tfrac{\phi_1+\phi_2}{2}\right)\right)^2+(\phi_2-\phi_1)^2}\cdot 6371000 &&=0\\
\Leftrightarrow\quad &\sqrt{\left((\lambda_2-\lambda_1)\cdot\cos\left(\tfrac{\phi_1+\phi_2}{2}\right)\right)^2+(\phi_2-\phi_1)^2} &&=0\\
\Leftrightarrow\quad &\left((\lambda_2-\lambda_1)\cdot\cos\left(\tfrac{\phi_1+\phi_2}{2}\right)\right)^2+(\phi_2-\phi_1)^2 &&=0
\end{aligned}$$

At this point either both summands are 0 or one is the negative of the other. However, both summands must be positive due to the quadration. Because of that we follow

$$\begin{aligned}
&(\phi_2-\phi_1)^2 &&=0\\
\Leftrightarrow\quad &\phi_2 &&=\phi_1
\end{aligned}$$

and with that

$$\begin{aligned}
&\left((\lambda_2-\lambda_1)\cdot\cos\left(\tfrac{\phi_1+\phi_2}{2}\right)\right)^2 &&=0\\
\Leftrightarrow\quad &(\lambda_2-\lambda_1)\cdot\cos\left(\tfrac{2\phi_1}{2}\right) &&=0\\
\Leftrightarrow\quad &(\lambda_2-\lambda_1)\cdot\cos\left(\phi_1\right) &&=0.
\end{aligned}$$

Since $\phi_1\in\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$ it follows that $\cos(\phi_1)\neq 0$. As such

$$\begin{aligned}
&\lambda_2-\lambda_1 &&=0\\
\Leftrightarrow\quad &\lambda_2 &&=\lambda_1
\end{aligned}$$

and by that $(\phi_1,\lambda_1)=(\phi_2,\lambda_2)$, so the second axiom holds.

Symmetry follows quickly since

$$\begin{aligned}
\phi_1+\phi_2&=\phi_2+\phi_1\\
(\phi_2-\phi_1)^2&=(\phi_1-\phi_2)^2\\
\left((\lambda_2-\lambda_1)\cdot\cos\left(\frac{\phi_1+\phi_2}{2}\right)\right)^2&=(\lambda_2-\lambda_1)^2\cdot\cos^2\left(\frac{\phi_1+\phi_2}{2}\right)\\
(\lambda_2-\lambda_1)^2&=(\lambda_1-\lambda_2)^2.
\end{aligned}$$

The triangle inequality is a bit trickier, we choose three arbitrary coordinates $c_i = (\phi_i, \lambda_i)$ for $i = 1, 2, 3$ and start on the squared left side:

$$
\begin{aligned}
\text{asTheCrowFlies}^2(c_1, c_3) &= \left(\left((\lambda_3 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_3}{2}\right)\right)^2 + (\phi_3 - \phi_1)^2\right) \cdot 6371000^2 \\
&= \left(\left((\lambda_3 - \lambda_2 + \lambda_2 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_3}{2}\right)\right)^2 + (\phi_3 - \phi_2 + \phi_2 - \phi_1)^2\right) \cdot 6371000 \\
&= \Bigg(\left((\lambda_3 - \lambda_2)^2 + (\lambda_2 - \lambda_1)^2 + 2 \cdot ((\lambda_3 - \lambda_2) \cdot (\lambda_2 - \lambda_1))\right) \cdot \cos^2\left(\frac{\phi_1 + \phi_3}{2}\right) \\
&\quad + (\phi_3 - \phi_2)^2 + (\phi_2 - \phi_1)^2 + 2 \cdot ((\pi_3 - \phi_2) \cdot (\phi_2 - \phi_1))\Bigg) \cdot 6371000^2 \\
&= \dots \\
&\leq \left(\left(\left((\lambda_2 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_2}{2}\right)\right)^2 + (\phi_2 - \phi_1)^2\right) \cdot 6371000^2\right) \\
&\quad + \left(\left(\left((\lambda_3 - \lambda_2) \cdot \cos\left(\frac{\phi_2 + \phi_3}{2}\right)\right)^2 + (\phi_3 - \phi_2)^2\right) \cdot 6371000^2\right) \\
&\quad + 2 \cdot \left(\left(\left((\lambda_2 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_2}{2}\right)\right)^2 + (\phi_2 - \phi_1)^2\right) \cdot 6371000^2\right) \\
&\quad \cdot \left(\left(\left((\lambda_3 - \lambda_2) \cdot \cos\left(\frac{\phi_2 + \phi_3}{2}\right)\right)^2 + (\phi_3 - \phi_2)^2\right) \cdot 6371000^2\right) \\
&= (\text{asTheCrowFlies}(c_1, c_2) + \text{asTheCrowFlies}(c_2, c_3))^2
\end{aligned}
$$

TODO: continue (squared ineq holds without, cauchy schwarz ineq) or remove completely...

All four axioms hold, asTheCrowFlies is a metric on the set M. □

3 Models

This section defines the models we use for the different network types. We define a graph based representation for road and transit networks. Then both graphs are combined into a linked graph, making it possible to have one graph for the whole network. Afterwards an alternative representation for transit networks is shown.

3.1 Road graph

A road network typically is time independent. It consists of geographical locations and roads connecting them with each other. We assume that a road can be taken at any time, with no time dependent constraints (see **Section 2** of [9]).

Modeling the network as graph is straightforward, **Definition 11** goes into detail.

Definition 11. *A* road graph *is a graph $G = (V, E)$ with a set of geographic coordinates*

$$V = \{(\phi, \lambda) | \phi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \lambda \in [-\pi, \pi)\},$$

for example road junctions. There is an edge $(u, w, v) \in E$ iff there is a road connecting the location u with the location v, which can be taken in that direction. The weight w of the edge is the average time needed to take the road from u to v using a car, measured in seconds.

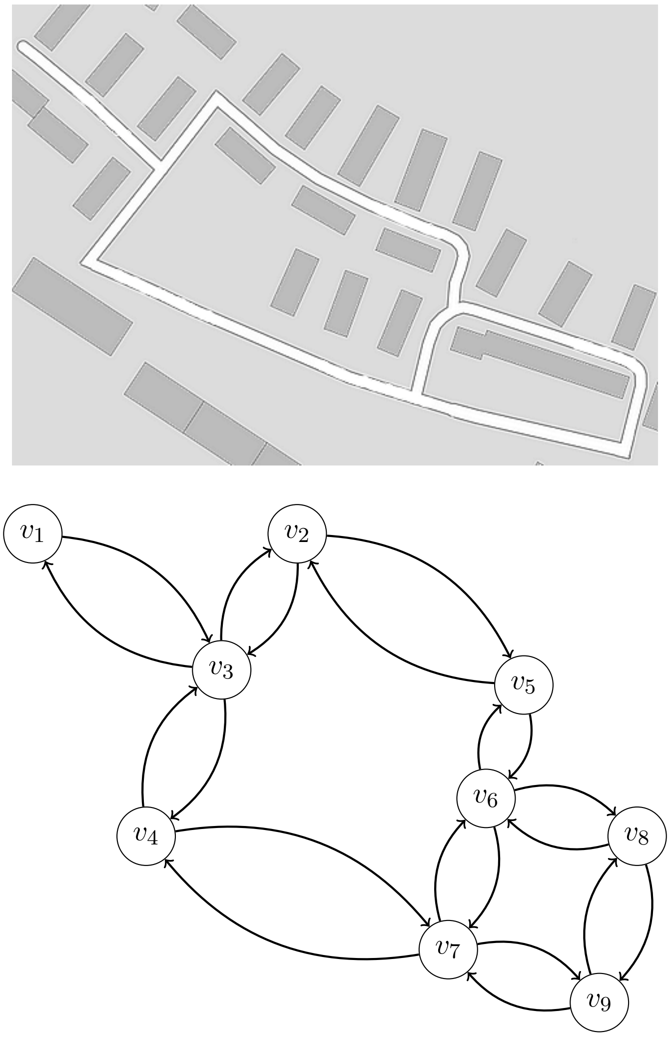

Fig. 3: Example of a road network with its corresponding road graph. White connections indicate roads, dark gray rectangles represent houses or other static objects. Geographical coordinates for each node as well as edge weights are omitted in the graph illustration.

Fig. 3 shows a constructed example road network with the corresponding road graph. Note that two way streets result in two edges, one edge for every direction the road can be taken.

Since edge weights are represented as average time it needs to take the road, it is possible to encode different road types. For example the average speed on a motorway is much higher than on a residential street. As such, the weight of an edge representing a motorway is much smaller than the weight of an edge representing a residential street.

While the example has exactly one node per road junction this must not always be the case. Typical real world data often consists of multiple nodes per road segment. However, **Definition 11** is still valid for such data as long as there are edges between the nodes if and only if there is a road connecting the locations.

3.2 Transit graph

Transit networks can be modeled similar to road graphs. The key difference is that transit networks are time dependent while road networks typically are not. For example an edge connecting *Freiburg main station* with *Karlsruhe main station* can not be taken at any time since trains and other transit vehicles only depart at certain times. The schedule might even change at different days.

The difficulty lies in modeling time dependence in a static graph. There are two common approaches to that problem (see [**9**, **15**, **3**]).

The first approach is called *time-dependent*. There edge weights are not static numbers but functions that take a date with time and compute the cost it needs to take the edge when starting at the given time. This includes waiting time. As an example assume an edge (u, c, v) with the cost function c. The edge represents a train connection and the travel time are 10 minutes. However, the train departs at 10:15 *am* am but the starting time is 10:00 *am*. The cost function thus computes a waiting time of 15 minutes plus the travel time of 10 minutes. Resulting in an edge weight of 25 minutes.

The main problem with this model is that it makes pre-computations for route planning very difficult as the starting time is not known in advance.

The second approach, originally from [**18**], is called *time-expanded*. There, idea is to remove any time dependence from the graph by creating additional nodes for every event at a station. A node then also has a time information next to its geographic location.

Definition 12. *A* time expanded transit graph *is a graph $G = (V, E)$ with a set of events at geographic coordinates*

$$V = \{(\phi, \lambda, t) | \phi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \lambda \in [-\pi, \pi), t \textit{ time}\},$$

for example a train arriving or departing at a train station at a certain time.

For a node $v \in V$, v_ϕ and v_λ denote its location and v_t its time.

There is an edge $(u, w, v) \in E$ *iff*

1. *there is a vehicle departing from* u *at time* u_t *which arrives at* v *at time* v_t *without stops in between, or*

2. v *is the node at the same coordinates than* u *with the smallest time* v_t *that is still greater than* u_t*. This edge represents exiting a vehicle and waiting for another connection. That is*

$$\forall v' \in V \setminus \{v\} : v'_\phi = u_\phi \wedge v'_\lambda = u_\lambda \wedge v'_t \geq u_t$$
$$\Rightarrow v'_t - u_t > v_t - u_t.$$

The weight w *of an edge* (u, w, v) *is the difference between both nodes times, that is*

$$w = v_t - u_t.$$

Note that weights are still positive since $v_t \geq u_t$ *always holds due to construction.*

Definition 12 defines such a time expanded transit graph and **Fig. 4** shows an example. For simplicity it is assumed that the trains have no stops other than shown in the schedule. The schedule lists four trains:

1. The ICE 104 which travels from Freiburg Hbf to Karlsruhe Hbf via Offenburg,
2. theRE 17024 connecting Freiburg Hbf with Offenburg,
3. the RE 17322 driving from Offenburg to Karlsruhe Hbf and
4. ICE ICE 79 which travels in the opposite direction, connecting Karlsruhe Hbf with Freiburg Hbf without intermediate stops.

As seen in the example the resulting graph has no time dependency anymore and is static, as well as all edge weights. The downside is that the graph size dramatically increases as a new node is introduced for every single event. In order to limit the growth we assume that a schedule is the same every day and does not change. In fact, most schedules are stable and often change only slightly, for example on weekends or at holidays. In practice hybrid models can be used for those exceptions.

However, the model still lacks an important feature. It does not represent *transfer buffers* [15, 3] yet. It takes some minimal amount of time to exit a vehicle and enter a different vehicle, possibly even at a different platform.

We model that by further distinguishing the nodes by arrival and departure events. In between we can then add transfer nodes which model the transfer duration. Therefore, the previous definition is adjusted and **Definition 13** is received.

Definition 13. *A* realistic time expanded transit graph *is a graph* $G = (V, E)$ *with a set of events at geographic coordinates*

$$V = \{(\phi, \lambda, t, e) | \phi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \lambda \in [-\pi, \pi), t \text{ time}, e \in \{\textit{arrival}, \textit{departure}, \textit{transfer}\}\},$$

$\longrightarrow$	Freiburg Hbf	Offenburg		Karlsruhe Hbf
	departure	arrival	departure	arrival
ICE 104	3:56 *pm*	4:28 *pm*	4:29 *pm*	4:58 *pm*
RE 17024	4:03 *pm*	4:50 *pm*		
RE 17322			4:35 *pm*	5:19 *pm*
$\longleftarrow$	arrival	departure	arrival	departure
ICE 79	8:10 *pm*			7:10 *pm*

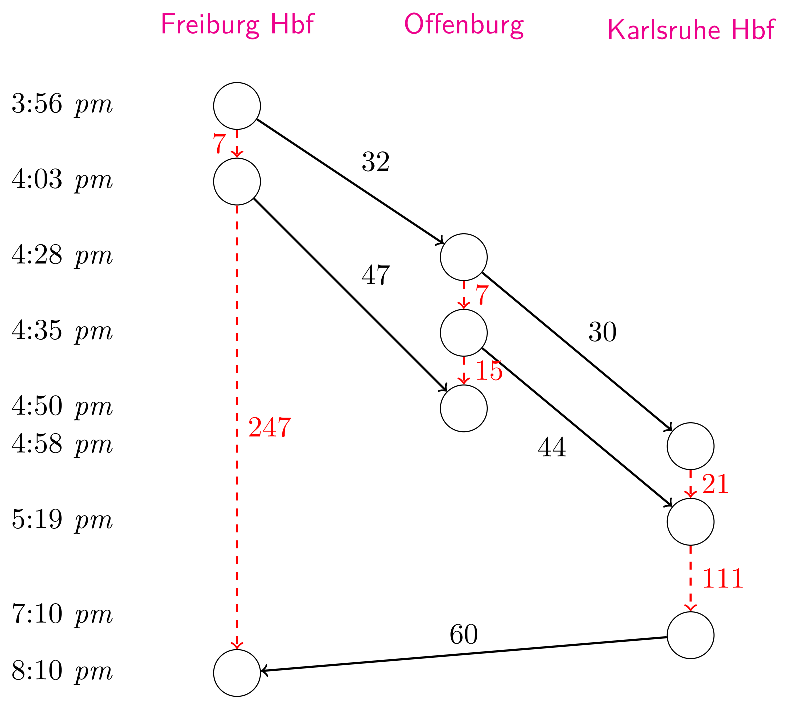

Fig. 4: Example of a transit network with its corresponding time expanded transit graph. The table shows an excerpt of a train schedule. Regular edges indicate a train connection and dashed edges waiting edges. Edge weights are measured in minutes.

for example a train arriving at a train station at a certain time.

A node $(\phi, \lambda, t, e) \in V$ *is an* arrival node *if* $e =$ arrival, *analogously it is a* departure node *for* $e =$ departure *and a transfer node for* $e =$ transfer. *For a node* $v \in V$, v_ϕ *and* v_λ *denote its location,* v_t *its time and* v_e *its event type.*

For every arrival node n *there must exist a transfer node* m *at the same coordinates such that* $m_t = n_t + d$ *with* d *being the average transfer duration at the corresponding stop.*

There is an edge $(u, w, v) \in E$ *iff*

1. $u_e = \textsf{departure} \wedge v_e = \textsf{arrival}$ *such that there is a vehicle departing from* u *at time* u_t *which arrives at* v *at time* v_t *without stops in between; or*

2. $u_e = \textsf{arrival} \wedge v_e = \textsf{departure}$ *such that* u *and* v *belong to the same connection. For example a train arriving at a station and then departing again; or*

3. $u_e = \textsf{arrival} \wedge v_e = \textsf{transfer}$ *such that* v *is the first transfer node at the same coordinates whose time* v_t *comes after* u_t*. That is*

$$\forall v' \in V \setminus \{v\} : v'_\phi = u_\phi \wedge v'_\lambda = u_\lambda \wedge v'_e = \textsf{transfer} \wedge v'_t \geq u_t$$
$$\Rightarrow v'_t - u_t > v_t - u_t.$$

   Such an edge represents exiting the vehicle and getting ready to enter a different vehicle; or

4. $u_e = \textsf{transfer} \wedge v_e = \textsf{transfer}$ *such that* v *is the first transfer node at the same coordinates whose time* v_t *comes after* u_t*, representing waiting at a stop; or*

5. $u_e = \textsf{transfer} \wedge v_e = \textsf{departure}$ *such that* u *is the last transfer node at the same coordinates whose time* u_t *comes before* v_t*, i.e.*

$$\forall u' \in V \setminus \{u\} : u'_\phi = v_\phi \wedge u'_\lambda = v_\lambda \wedge u'_e = \textsf{transfer} \wedge u'_t \leq v_t$$
$$\Rightarrow v_t - u'_t > v_t - u_t.$$

   An edge like this represents entering a different vehicle from a stop after transferring or waiting at the stop.

The weight w *of an edge* (u, w, v) *is the difference between both nodes times, that is*

$$w = v_t - u_t.$$

Fig. 5 shows how the transit graph from **Fig. 4** changes with transfer buffers.

The weight of edges connecting arrival nodes with transfer nodes is equal to the transfer duration, 5 minutes in the example. The transfer duration can be different for each edge. A transfer is now possible if the departure of the desired vehicle is after the arrival of the current vehicle plus the duration time. As seen in the example, edges connecting transfer nodes with departure nodes are present exactly in this case. A transfer from ICE 104 to RE 17322 in Offenburg is indicated by taking the edge to the first transfer node in Offenburg and then following the edge with cost 2 to the departure node of the train.

3.3 Link graph

In this section we examine how a road and a transit graph can be combined into a single graph such that all connections of the real network are preserved.

The approach is simple, selected nodes in the road network are connected to nodes

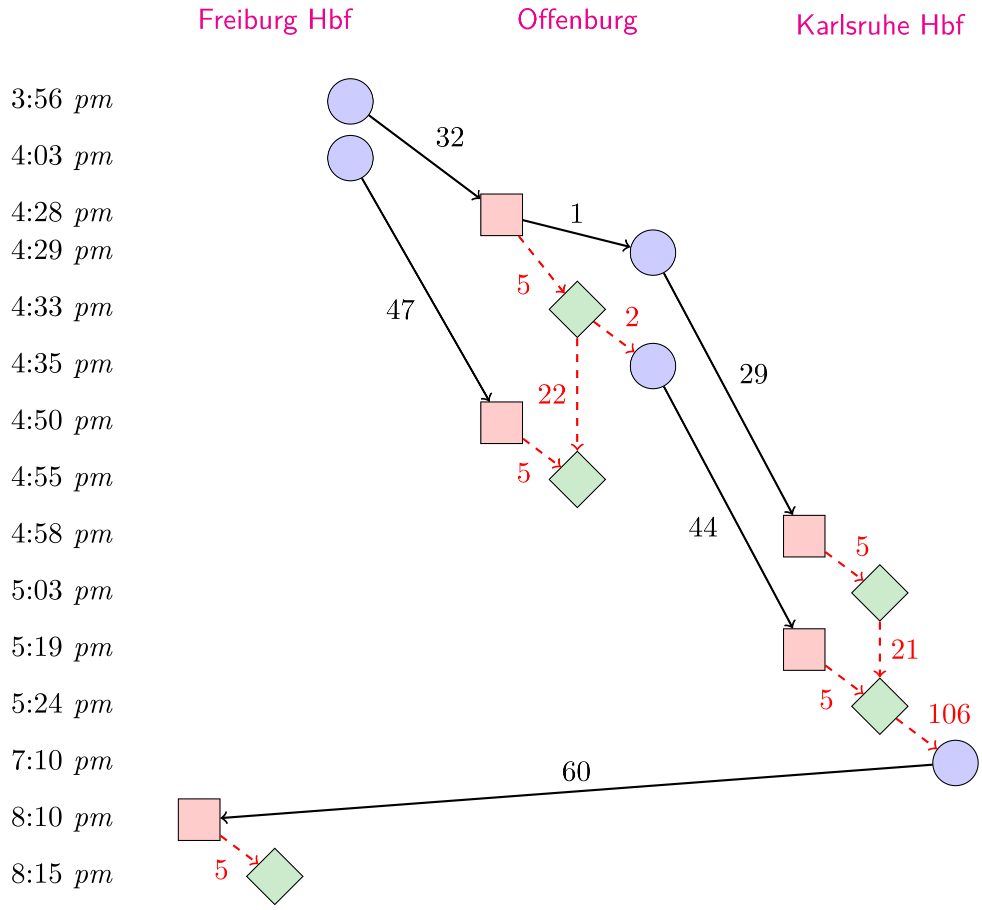

Fig. 5: Illustration of a realistic time expanded transit graph representing the schedule from **Fig. 4**. A transfer duration of 5 minutes is assumed at every stop. Rectangular nodes are arrival nodes, circular nodes represent departure nodes and diamond shaped nodes are transfer nodes. Regular edges indicate a train connection and dashed edges involve transfer nodes. Edge weights are measured in minutes.

of a certain stop in the transit network and vice versa. Since starting time is not known in advance the graph must connect a road node to all arrival nodes of a stop (compare to [8]).

In order to not miss a connection the transit graph must ensure that every connection starts with an arrival node. In **Fig. 5** this is not the case and all four trains start at a departure node. However, this is easily fixed by adding an additional arrival node to the beginning of every connection not starting with an arrival node already. The arrival nodes time is the same as the time of the departure node and both are connected by an edge with a weight of 0. **Definition 14** formalized the model.

Definition 14. *Assume a road graph* $R = (V_R, E_R)$*, a realistic time expanded transit* *graph $T = (V_T, E_T)$ where every connection in T starts by an arrival node and a partial function* link : $V_R \nrightarrow M$ *where M contains subsets $S \subseteq V_T$. For every element $S \in M$ with an arbitrary element $s \in S$ the following properties must hold:*

1. *All contained elements must be arrival nodes and have the same location than s, i.e.*

$$\forall s' \in S : s'_e = \textsf{arrival} \wedge s'_\phi = s_\phi \wedge s'_\lambda = s_\lambda.$$

2. *The set must contain all arrival nodes at the location of s, i.e.*

$$\nexists v \in V_T \setminus S : v_e = \textsf{arrival} \wedge v_\phi = s_\phi \wedge v_\lambda = s_\lambda.$$

Then a link graph *is a graph $L = (V_R \uplus V_T, E_R \uplus E_T \uplus E_L)$ with an additional set of link edges $E_L = V_R \times \mathbb{R}_{\geq 0} \times V_T$.*

There is an edge $(u, 0, v) \in E_L$ iff $\mathrm{link}(u)$ *is defined and $v \in \mathrm{link}(u)$.*

The function link can be obtained in different ways. For example by creating a mapping from a road node u to a stop S if u is in the vicinity of S according to the asTheCrowFlies metric.

Another straightforward possibility is to always connect a stop with the road node nearest to it. We will explore this problem in **Section 4**. An obvious downside of this approach is that the nearest road node might not always have a good connectivity to the road network. A solution consists in creating a road node at the coordinates of the stop as representative. The node can then be connected with all road nodes in the vicinity.

3.4 Timetable

Timetables [3] are non-graph based representations for transit networks. They consists of stops, trips, connections and footpaths.

Definition 15. *A* timetable *is a tuple (S, T, C, F) with stops S, trips T, connections C and footpaths F.*

A stop is a position where passengers can enter or exit a vehicle, for example a train station or bus stop. It is represented as geographical coordinate (ϕ, λ) with $\phi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \lambda \in [-\pi, \pi)$.

A trip is a scheduled vehicle, like the ICE 104 in the example schedule of **Fig. 4** or a bus.

In contrast to a trip a connection is only a segment of a trip without stops in between. For example the connection of the ICE 104 from Freiburg Hbf at 3:56 *pm* to Offenburg with arrival at 4:28 *pm*. It is defined as tuple $c = (s_{\mathsf{dep}}, s_{\mathsf{arr}}, t_{\mathsf{dep}}, t_{\mathsf{arr}}, o)$ with $s_{\mathsf{dep}}, s_{\mathsf{arr}} \in S$ representing the departure and arrival stop of the connection respectively.

Analogously t_{dep} is the time the vehicle departs at s_{dep} and t_{arr} when it arrives at s_{arr}. And $o \in T$ is the trip the connection belongs to.

Footpaths represent transfer possibilities between stops and are formalized as ordered tuple $(s_{\mathsf{dep}}, d, s_{\mathsf{arr}})$ with $s_{\mathsf{dep}}, s_{\mathsf{arr}} \in S$ the stops the footpath connects. The duration it needs to take the path by foot is represented by d, measured in seconds. Together with the set of stops S the footpaths build a graph $G = (S, F)$ representing directed edges between stops.

We require the following for the footpaths:

1. The footpaths must be transitively closed, that is

$$\exists (a, d_1, b), (b, d_2, c) \in F \Rightarrow (a, d_3, c) \in F$$

for arbitrary durations d_1, d_2, d_3.

2. The triangle inequality must hold for all footpaths:

$$\exists (a, d_1, b), (b, d_2, c) \in F \Rightarrow \exists (a, d_3, c) \in F : d_3 \leq d_1 + d_2$$

3. Every stop must have a self-loop footpath, i.e.

$$\forall s \in S \Rightarrow (s, d, s) \in F.$$

The duration d models the transfer time at this stop, as already seen in **Section 3.2**.

The first property can easily make the set of footpaths huge. However, it is necessary for our algorithms that the amount of footpaths stays relatively small. In practice, we therefore connect each stop only to stops in its vicinity and then compute the transitive closure to ensure it is transitively closed.

To familiarize more with the model we take a look at the schedule from **Fig. 4** again. The corresponding timetable consists of:

$$S = \{f, o, k\},$$

where f, o, k represent Freiburg Hbf, Offenburg and Karlsruhe Hbf respectively;

$$T = \{t_{104}, t_{17024}, t_{17322}, t_{79}\},$$

representing the four trains ICE 104, RE 17024, RE 17322 and ICE 79; the connections

$$\begin{aligned}
&(f, o, 3{:}56\ pm, 4{:}28\ pm, t_{104}),\\
&(o, k, 4{:}29\ pm, 4{:}58\ pm, t_{104}),\\
&(f, o, 4{:}03\ pm, 4{:}50\ pm, t_{17024}),\\
&(o, k, 4{:}35\ pm, 5{:}19\ pm, t_{17322}),\\
&(k, f, 7{:}10\ pm, 8{:}10\ pm, t_{79})
\end{aligned}$$

and at least the footpaths

$$(f, 300, f),$$
$$(o, 300, o),$$
$$(k, 300, k)$$

for transferring at the same stop with a duration of 300 seconds (5 minutes).

If we would decide that Offenburg is reachable from Freiburg Hbf by foot and analogously Karlsruhe Hbf from Offenburg, we would also need to add a footpath connecting Freiburg Hbf directly with Karlsruhe Hbf. Else the footpaths would not be transitively closed anymore.

4 Nearest neighbor problem

In this section we introduce the nearest neighbor problem, also known as nearest neighbor search (NNS). First, we define the problem. Then a short overview of related research is given, after which we elaborate on a solution called Cover Tree [5].

Definition 16. *Given a metric space (M, d) (see* ***Definition 8****) with $|M| \geq 2$ and a point $x \in M$, the* nearest neighbor problem *asks for finding a point $y \in M$ such that*

$$y = \underset{y' \in M \setminus \{x\}}{\arg\min}\, d(x, y').$$

The point y is called nearest neighbor *of x.*

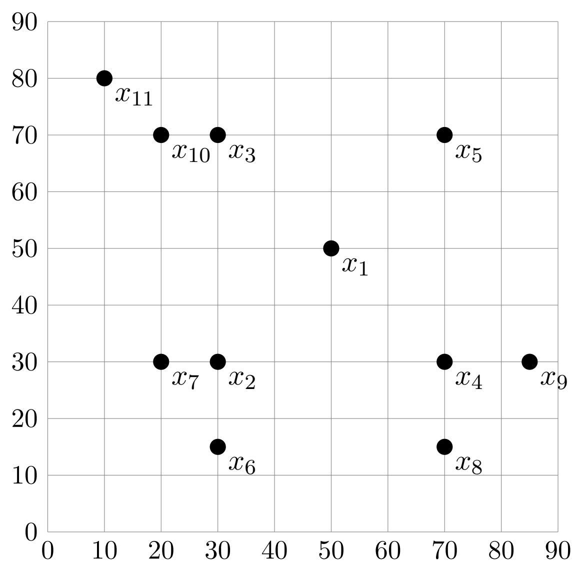

Fig. 6: Grid showing eleven points in the cartesian plane $\mathbb{R}^2$.

For following examples the toy data set shown in **Fig. 6** is introduced. It consists of the points

$$
\begin{aligned}
x_1 &= (50, 50),\\
x_2 &= (30, 30),\\
x_3 &= (30, 70),\\
x_4 &= (70, 30),\\
x_5 &= (70, 70),\\
x_6 &= (30, 15),\\
x_7 &= (20, 30),\\
x_8 &= (70, 15),\\
x_9 &= (85, 30),\\
x_{10} &= (20, 70),\\
x_{11} &= (10, 80).
\end{aligned}
$$

All points are elements of the cartesian plane $\mathbb{R}$. The euclidean distance d is chosen as metric on this set. For two dimensions it can be defined as:

$$d : \mathbb{R}^2 \times \mathbb{R}^2, ((x_1, y_1), (x_2, y_2)) \mapsto \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Informally d computes the *ordinary* straight-line distance between two points.

The nearest neighbor of x_5 is x_1 as

$$
\begin{aligned}
d(x_5, x_1) &= \sqrt{(50 - 70)^2 + (50 - 70)^2}\\
&= \sqrt{800}
\end{aligned}
$$

is smaller than all other distances to x_5, like

$$
\begin{aligned}
d(x_5, x_4) &= \sqrt{(70 - 70)^2 + (30 - 70)^2}\\
&= \sqrt{1600}.
\end{aligned}
$$

On the other hand, x_1 has four smallest neighbors:

$$d(x_1, x_2) = d(x_1, x_3) = d(x_1, x_4) = d(x_1, x_5)$$

Any of them is a valid solution to the nearest neighbor problem of x_1.

The search for a nearest neighbor is a well understood problem [**2, 1**] and has many applications. Without restrictions, solving the problem on general metrics was proven to require $\Omega(n)$ time [**2**], where n is the amount of points.

Typical approaches divide the space into regions, exploiting properties of the metric space. Common examples include K-D TREEs [4], VP TREEs [20], BK-TREEs [6] and COVER TREEs [5].

The problem also has a lot of variants. We elaborate on two of them:

Definition 17. *The* k-nearest neighbors *of a point* $x \in M$ *are the* k *closest points* $\{y_1, y_2, \ldots, y_k\} \subseteq M$ *to* x*. That is*

$$\begin{aligned} y_1 &= \underset{y' \in M \setminus \{x\}}{\arg\min}\, d(x, y'), \\ y_2 &= \underset{y' \in M \setminus \{x, y_1\}}{\arg\min}\, d(x, y'), \\ &\vdots \\ y_k &= \underset{y' \in M \setminus \{x, y_1, \ldots, y_{k-1}\}}{\arg\min}\, d(x, y'). \end{aligned}$$

Definition 18. *The* k-neighborhood *of a point* $x \in M$ *is the set*

$$\{y \in M \setminus \{x\} | d(x, y) \leq k\}.$$

4.1 Cover tree

Definition 19. *A* cover tree T *on a metric space* (M, d) *is a leveled tree* (V, E)*.*

The root is placed at the greatest level, denoted by $i_{max} \in \mathbb{Z}$*. The level of a node* $v \in V$ *is*

$$\text{lvl}(v) = k - \text{depth}(v).$$

The lowest level is denoted by i_{min}*. Every node* $v \in V$ *is associated with a point* $m \in M$*. We write* $\text{assoc}(v) = m$*. Nodes of a certain level form a cover of points in* M*. A cover for a level* i *is defined as*

$$C_i = \{m \in M | \exists v \in V : \text{lvl}(v) = i \wedge \text{assoc}(v) = m\}.$$

The following properties must hold

1. *For a level* i *there must not exist nodes which are associated with the same point* $m \in M$*:*

$$\nexists v, v' \in V : i = \text{lvl}(v) = \text{lvl}(v') \wedge v \neq v' \wedge \text{assoc}(v) = \text{assoc}(v')$$

   So each point can at most appear once per level.

2. $C_i \subset C_{i-1}$*. This ensures that, once a point was associated with a node in a level, it appears in all lower levels too.*

3. *Points are covered by their parents:*

$$\forall p \in C_{i-1} \exists q \in C_i : d(p, q) < 2^i$$

and the node v_p with $\mathrm{lvl}(v_p) = i \wedge \mathrm{assoc}(v_p) = p$ is the parent of the node v_q with $\mathrm{lvl}(v_q) = i - 1 \wedge \mathrm{assoc}(v_q) = q$.

4. *Points in a cover C_i have a separation of at least 2^i, i.e.*

$$\forall p, q \in C_i : p \neq q \Rightarrow d(p, q) > 2^i.$$

A cover tree [5] has interesting distance properties on its nodes which allows for efficient retrieval of nearest neighbors. The general approach is straightforward. Given a node v in the tree placed at level i, we know that all nodes of the subtree rooted at v are associated with points inside a distance of at most 2^i. This means that, if we search for a nearest neighbor, and traverse to a node v in the tree, all nodes underneath v are relatively close to v. So, if we already have a candidate for a nearest neighbor, with distance of d and v is already further away than $d + 2^i$; v and all nodes in its subtree can not improve the distance.

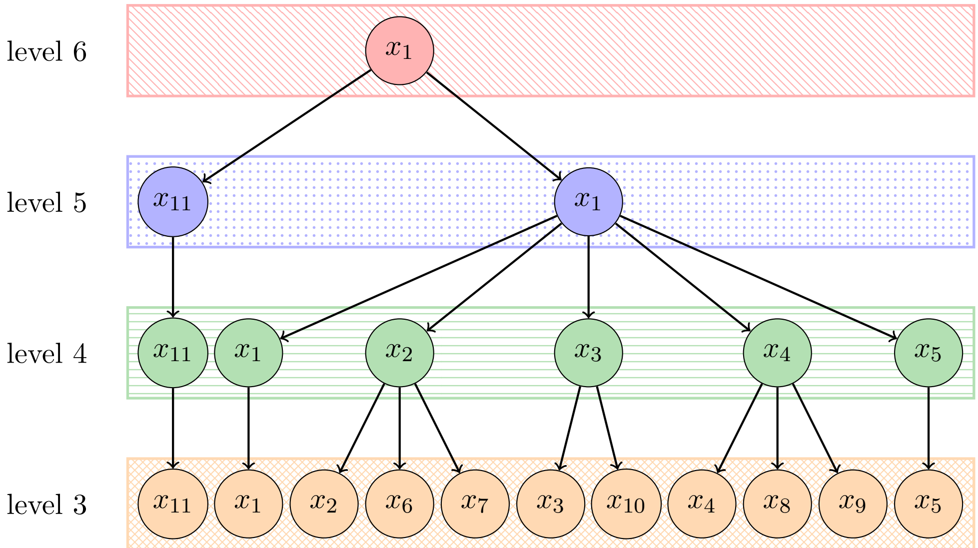

Fig. 7: Cover tree for the data set of **Fig. 6**. Nodes are vertically grouped by their levels and highlighted accordingly.

Fig. 7 shows a valid cover tree for the toy example illustrated by **Fig. 6**. The covers are

$$\begin{aligned} C_6 &= \{x_1\}, \\ C_5 &= \{x_1, x_{11}\}, \\ C_4 &= \{x_1, x_2, x_3, x_4, x_5, x_{11}\}, \\ C_3 &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}. \end{aligned}$$

Clearly the first property holds, there is no level where a x_i is associated with a node

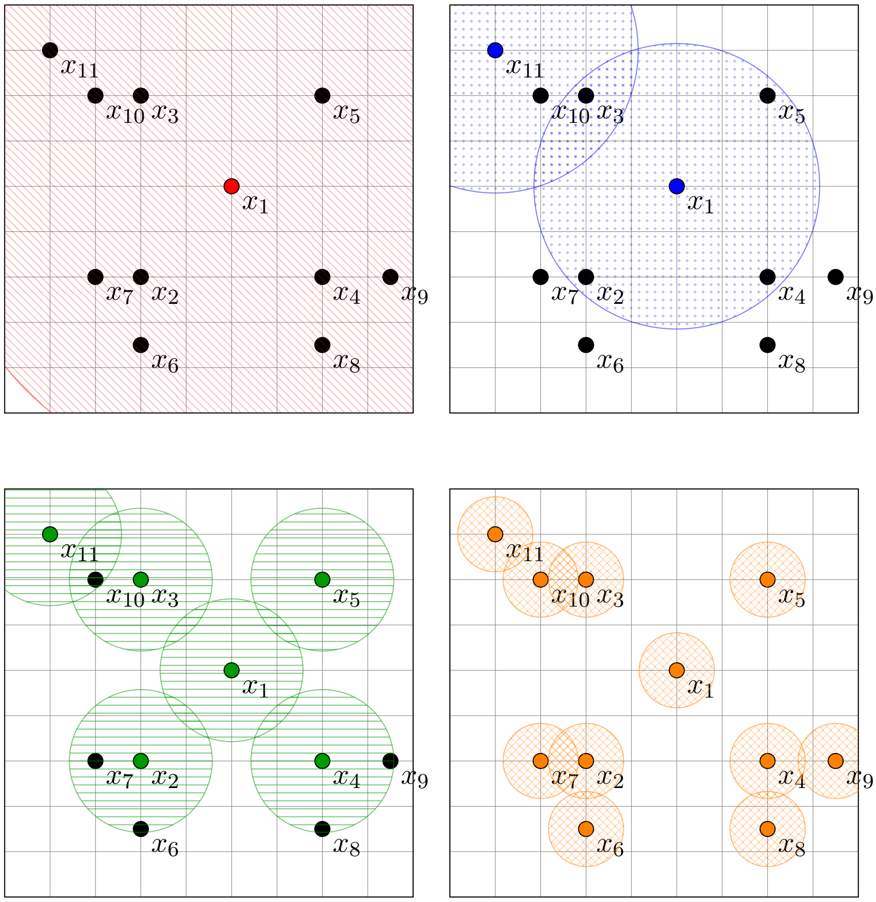

Fig. 8: Figure that shows the separation property for each level of the cover tree shown by **Fig. 7**. The levels are highlighted in the same manner than in the previous example. The levels are $6, 5, 4$ and 3 from top left to bottom right. The radii around the points have a size of $2^6, 2^5, 2^4$ and 2^3.

more than once. The second property holds too, it is

$$C_6 \subset C_5 \subset C_4 \subset C_3.$$

For the last two properties we take a look at **Fig. 8**. It illustrates the fourth property. The property states that all points in a cover C_i must have a distance of at least 2^i to each other. For level 6 this is trivial since the set only contains x_1. For level 5 it must hold that

$$d(x_1, x_{11}) = 50 > 32 = 2^5,$$

which is true. If this would not be the case, the figure would show the nodes included inside the circle around the other node. Analogously all nodes in C_4 and C_3 are separated enough from each other.

The third property can easily be confirmed using the figure too. It states that a node in level $i-1$ must be closer than 2^i to its parent. Obviously this holds for x_1 and x_{11} in level 5, as a radius of 2^6 around their parent x_1 covers all nodes. Likewise are x_1, x_2, x_3, x_4 and x_5 included in the circle around their parent x_1 with radius 2^5.

Note that it is not necessary that a node covers its whole subtree in its level. As example, we refer to x_1 in level 5 which does not cover x_{10}, as $d(x_1, x_{10}) > 2^5$, though it is part of the subtree rooted at x_1. The third property only demands that a parent covers all its direct children, not grandchildren or similar.

Algorithm 1: Inserting a point into a cover tree operating on a metric space (M, d).

input : point $p \in M$, candidate cover set $Q_i \subseteq C_i$, level i
output: true if p was inserted at level $i-1$, false otherwise

$Q := \{\texttt{children}(q) | q \in Q_i\}$;
if $d(p, Q) > 2^i$ **then**
  return false ; // Check separation
else
  $Q_{i-1} := \{q \in Q | \mathtt{d}(p, q) \leq 2^i\}$; // Covering candidates
  if $\neg\texttt{insert}(p, Q_{i-1}, i-1) \wedge \mathtt{d}(p, Q_i) \leq 2^i$ **then**
    pick any $q \in Q_i : \mathtt{d}(p, q) \leq 2^i$;
    append q as child to q;
    return true;
  else
    return false;

The cover tree is constructed using **Algorithm 1** with the maximal level i_{max} and the cover set C_k which only consists of the root. The algorithm is stated recursively, but can easily be implemented without recursion by descending the levels and only following relevant candidates.

A point p can be appended in level $i-1$ to a parent q in level i if the point has enough separation to all other nodes in this level, meaning more than 2^{i-1}, and is covered by the parent, that is a distance of less than 2^i. The algorithm searches such a point by descending the levels, computing the separation and appending it to a node if it also covers the point.

A search for a nearest neighbor follows a similar approach. **Algorithm 2** starts at the root and traverses the tree by following the children. The candidate set is refined by

Algorithm 2: Searching a nearest neighbor in a cover tree operating on a metric space (M, d).

input : point $p \in M$
output: a nearest neighbor to p in M

$Q_{i_{\mathsf{max}}} := C_{i_{\mathsf{max}}}$;
for i *from* i_{max} *to* i_{min} **do**
  $Q := \{\texttt{children}(q) | q \in Q_i\}$;
  $Q_{i-1} := \{q \in Q | \texttt{d}(p, q) \leq \texttt{d}(p, Q) + 2^i\}$;
return $\arg\min_{q \in Q_{i_{\mathsf{min}}}} \texttt{d}(p, q)$;

only following children which are closer than

$$d(p, Q) + 2^i.$$

There, the distance to the set represents the distance of the currently best candidate. Nodes in the subtree rooted at a child can maximally be 2^i closer than the child itself. Therefore, take a look at **Fig. 8** where x_2 is maximally 2^5 closer to x_7 than x_1, else it would not be covered by its parent x_1. Because of that the algorithm only follows children which can have nodes in their subtree that improve over the currently best candidate. Other children are rejected.

Note that the algorithm must track down all levels, as another node could show up in the lowest level because of the separation property.

Algorithm 3: Searching the k-nearest neighbors in a cover tree operating on a metric space (M, d).

input : point $p \in M$, amount $k \in \mathbb{N}$
output: k-nearest neighbors to p in M

$Q_{i_{\mathsf{max}}} := C_{i_{\mathsf{max}}}$;
for i *from* i_{max} *to* i_{min} **do**
  $Q := \{\texttt{children}(q) | q \in Q_i\}$;
  perform a k-partial sort of Q, ascending in $\texttt{d}(p, q)$;
  let q' be the k-th element of Q;
  $Q_{i-1} := \{q \in Q | \texttt{d}(p, q) \leq \texttt{d}(p, q') + 2^i\}$;
perform a k-partial sort of $Q_{i_{\mathsf{min}}}$, ascending in $\texttt{d}(p, q)$;
return first k elements of $Q_{i_{\mathsf{min}}}$;

The cover tree can also be used to efficiently compute the k-nearest neighbors or the

Algorithm 4: Computing the k-neighborhood by using a cover tree which operates on a metric space (M, d).

input : point $p \in M$, radius $k \in \mathbb{R}_{\geq 0}$
output: k-neighborhood of p in M

$Q_{i_{\mathsf{max}}} := C_{i_{\mathsf{max}}}$;
for i *from* i_{max} *to* i_{min} **do**
  $Q := \{\texttt{children}(q) | q \in Q_i\}$;
  $Q_{i-1} := \{q \in Q | \texttt{d}(p, q) \leq k + 2^i\}$;
return$\{q \in Q_{i_{\mathsf{min}}} | \texttt{d}(p, q) \leq k\}$;

k-neighborhood. In order to compute the k-nearest neighbors, **Algorithm 3** extends the range bound from the currently best candidate to the k-th best candidate. Likewise does **Algorithm 4** extend the bound to the given range k instead of involving candidate distances.

For other operations and a detailed analysis of the cover tree, as well as its complexity and a comparison against other techniques, refer to [5].

5 Shortest path problem

Blabla

5.1 Time-independent

Blabla

5.1.1 Dijkstra

Blabla

5.1.2 A* and ALT

Blabla

5.2 Time-dependend

Blabla

5.2.1 Connection scan

Blabla

5.3 Multi-modal

Blabla

5.3.1 Modified Dijkstra

Blabla

5.3.2 Access nodes

Blabla

5.4 Other algorithms

Blabla

6 Evaluation

Blabla

6.1 Input data

Blabla

6.2 Experiments

Blabla

6.2.1 Nearest neighbor computation

Blabla

6.2.2 Uni-modal routing

Blabla

6.2.3 Multi-modal routing

Blabla

6.3 Summary

Blabla

7 Conclusion

Blabla

References

[1] Mohammad Reza Abbasifard, Bijan Ghahremani, and Hassan Naderi. A survey on nearest neighbor search methods. 2014.

[2] Alexandr Andoni. Nearest neighbor search : the old , the new , and the impossible by alexandr andoni. 2009.

[3] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks*, pages 19–80. Springer International Publishing, Cham, 2016.

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[5] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 97–104, New York, NY, USA, 2006. ACM.

[6] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, April 1973.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[8] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating multi-modal route planning by access-nodes. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 587–598, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[9] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. *Engineering Route Planning Algorithms*, pages 117–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[10] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *CoRR*, abs/1703.05997, 2017.

[11] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[12] Wei Dong. An overview of in-vehicle route guidance system. In *Australasian Transport Research Forum*, volume 2011. Citeseer, 2011.

[13] R. L. French. Historical overview of automobile navigation technology. In *36th IEEE Vehicular Technology Conference*, volume 36, pages 350–358, May 1986.

[14] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[15] Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, pages 67–90, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] John P. Snyder. Flattening the earth: Two thousand years of map projections. 10 1994.

[17] C. C. Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1):38–40, 1957.

[18] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *J. Exp. Algorithmics*, 5, December 2000.

[19] Daniel Tischner. Cobweb. `https://github.com/ZabuzaW/Cobweb`, 2018.

[20] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993.

[21] Yilin Zhao. *Vehicle location and navigation systems*. 1997.