Public Transp (2017) 9:7–32
DOI 10.1007/s12469-016-0127-x

ORIGINAL PAPER

Line pool generation

Philine Gattermann[1] · **Jonas Harbering**[1] · **Anita Schöbel**[1]

Accepted: 18 June 2016 / Published online: 15 July 2016

Abstract Finding the lines and their frequencies in public transportation is the well-studied *line planning problem*. In this problem, it is common to assume that a line pool consisting of a set of potential lines is given. The goal is to choose a set of lines from the line pool that is convenient for the passengers and has low costs. The chosen lines then form the *line plan* to be established by the public transportation company. The line pool hence has a significant impact on the quality of the line plan. The more lines are in the line pool, the more flexible can we choose the resulting line plan and hence increase its quality. It hence would be preferable to allow all possible lines to choose from. However, the resulting instances of the line planning problem become intractable if all lines would be allowed. In this work, we study the effect of line pools for line planning models and propose an algorithm to generate 'good' line pools. To this end, we formally introduce the *line pool generation problem* and investigate its properties. The line pool generation problem asks for choosing a subset of paths (the line pool) of limited cardinality such that in a next step a good line concept can be constructed based on this subset. We show that this problem is NP-hard. We then discuss how reasonable line pools may be constructed. Our approach allows to construct line pools with different properties and even to engineer the properties of the pools to fit to the objective function of the line planning model to be used later on. Our numerical experiments on close-to real-world data show that the quality of a line plan significantly depends on the

✉ Philine Gattermann
p.gattermann@math.uni-goettingen.de

Jonas Harbering
jo.harbering@math.uni-goettingen.de

Anita Schöbel
schoebel@math.uni-goettingen.de

[1] University of Göttingen, Lotzestraße 16-18, 37083 Göttingen, Germany

underlying line pool, and that it can be influenced by the parameters of our approach.

Keywords Line pool · Line planning · Public transport · Algorithm design

1 Introduction

In public transportation planning the construction of the lines to be used plays a key role for all subsequent planning stages, such as setting the frequencies, finding the timetable and so on. Hence, the scope of line planning should be investigated thoroughly and limitations should be fathomed. Line planning has been extensively investigated in the literature, see Schöbel (2012), Kepaptsoglou and Karlaftis (2009) for surveys on the topic. The line planning problem is defined as follows:

Given a public transportation network $PTN = (V, E)$ with its set of stops or stations V and its set of direct connections E, a *line* is a path in the PTN. It is common to assume that a *line pool* $\mathcal{L}$ is given consisting of potential lines l, each of them with associated costs $cost_l$ for operating a vehicle on this line. The goal is to choose a *line concept*, i.e., a subset of lines together with their frequencies f_l, such that some given demand is satisfied. There are many possible models for determining a line concept which have been published in the literature. In the so-called *cost models*, the objective function minimizes the costs $\sum_{l \in \mathcal{L}} f_l cost_l$ of the lines chosen (see, e.g., Zwaneveld et al. 1996; Claessens et al. 1998) while in the *direct travelers approach* the goal is to maximize the number of direct travelers [as introduced in Dienst (1978) and further investigated in Bussieck et al. (1996); Bussieck (1998)]. *Travel time models* minimize the approximated traveling time over all OD-pairs (see Schöbel and Scholl 2006; Nachtigall and Jerosch 2008). In all these models it is assumed that the line pool is given beforehand as input. This may be very restrictive, in particular if the line pool is small.

Instead of choosing lines from a pre-specified line pool, it would hence be preferable to allow all possible paths through the public transportation network as lines. A pool consisting of all these possible lines would be by far too large. For the example network on which our results are based (which is similar to the German railway network) such a line pool contains more than one million lines. Firstly, the generation of these potential lines and secondly finding an optimal solution to the line planning problem within such a large line pool would be numerically intractable for most line planning models. Hence, one idea is to generate the lines within the line planning process. Only few work has been published on this simultaneous generation of lines. Borndörfer et al. (2007) developed a model in which lines are generated during the optimization within a column generation approach. However, the subproblem of generating the lines turns out to be a longest path problem which is NP-hard in general, hence the model is computationally hard to solve.

The following two early approaches already contain elements of the line pool generation problem and should hence be mentioned here: In Silman et al. (1974) an algorithm for line planning is proposed for which a line pool, called route set in this

paper, is computed. This is done by repeatedly adding lines to the already existing route set which improve the existing set of lines from a passenger oriented perspective. Mandl (1979) iteratively computes different sets of feasible lines which could also be interpreted as line pools. In the first iteration, lines are generated as shortest paths between any pair of leaves of the network. These lines are then adapted by a neighborhood search technique in subsequent steps.

We also mention that in the transportation (or transit route) network design problem (see Curtin and Biba 2011; Fan and Mumford 2010; Guan et al. 2006 for examples and Kepaptsoglou and Karlaftis 2009; Farahani et al. 2013 for overviews), it is common to compute the lines in a first step and assign frequencies to them later on. If the frequencies are allowed to be zero, the first step could also be interpreted as line pool generation. However, the difference is that in the network design problem, usually all lines generated in the first step are used, and hence only few lines are generated. This is far from the intention of our paper in which we generate a large and broad set of lines and allow the existing line planning algorithms in a second step to choose the ones which should really be used.

In this work we present an approach which lies between the two extremes of either having a fixed line pool or of generating the lines within the planning process, namely, we discuss how suitable line pools may be constructed in a pre-step before the actual line planning phase. Our approach allows to generate lines that serve certain purposes. Using our line pools within different line planning models, the influence of the line pool on the resulting optimal solution can be understood better such that we are able to give recommendations on properties lines should have in order to allow a good line concept.

2 Line planning in public transportation

The idea is to split the line planning phase into the following two steps:

Phase 1 Construct a line pool $\mathcal{L}$.
Phase 2 Take $\mathcal{L}$ as input for the line planning model which should be used for finding a line concept.

2.1 Phase 2: Finding a line concept

Before we deal with the generation of a line pool in Phase 1, we look at Phase 2 and describe formally what a line concept is and when it is feasible. Let a public transportation network $PTN = (V, E)$ with nodes V and edges E be given, and assume that we already know a line pool $\mathcal{L}$ of potential lines. A *line concept* assigns a frequency $f_l \in \mathbb{N}_0$ to every line l in the line pool. If $f_l = 0$ then the line l is not used and hence not part of the line plan. If $f_l > 0$ then the line is operated with the frequency f_l, i.e., f_l times in the planning period. As already mentioned in the introduction there are many different models for finding a line concept (see Schöbel 2012 for a survey), common objective functions include minimizing the costs,

maximizing the number of direct travelers or minimizing some approximation of the traveling time. Also the constraints differ from model to model. As Schöbel (2012) points out, there is one basic set of constraints which is used in most models for line planning, namely the edge frequency constraints: In order to allow all passengers to travel, it is common to assume a *lower frequency bound* $f_e^{\min}$ on each edge $e \in E$. It specifies the least amount of vehicles that have to traverse each edge. Analogously, also an *upper frequency bound*, denoted by f_e^{max} for all $e \in E$, is often given. This number represents the maximum number of vehicles that are allowed to traverse each edge and may model capacity or security restrictions. For a line concept with frequencies $f_l \in \mathbb{N}_0$ for all $l \in \mathcal{L}$ it is then required that

$$f_e^{min} \leq \sum_{\substack{l \in \mathcal{L}: \\ e \in l}} f_l \leq f_e^{max} \qquad \forall\, e \in E. \tag{1}$$

We finally repeat three line planning models which are used later on for evaluating our line pools:

(LP0) Basic line planning problem:

$$\text{Find } f_l \in \mathbb{N}_0 \text{ for all } l \in \mathcal{L} \text{ such that } (1) \text{ is satisfied.}$$

(LP-Cost) Cost model: Given cost values $cost_l$ for operating a vehicle on line l for all lines in $\mathcal{L}$, a simple cost model is given as follows.

$$\min\left\{\sum_{l \in \mathcal{L}} f_l\, cost_l \text{ s.t. } (1) \text{ is satisfied and } f_l \in \mathbb{N}_0 \text{ for all } l \in \mathcal{L}.\right\}$$

(LP-Direct) Direct passengers model: A direct passenger is a passenger who can travel between his origin and his destination on a preferable path (the preferable paths are given beforehand, e.g., as the shortest paths) without a transfer.

$$\max\{\text{number of direct passengers s.t. } (1) \text{ is satisfied and } f_l \in \mathbb{N}_0 \text{ for all } l \in \mathcal{L}.\}$$

Note that in order to count a passenger as a direct passenger the *capacity* of the vehicles in combination with the frequencies has to be high enough.

2.2 Phase 1: The line pool generation problem

We now turn our attention to Phase 1: We assume that any cycle free path in the PTN is a valid line. Let $\mathcal{L}^0$ be the set of all cycle free paths in the PTN. We then aim to find a set $\mathcal{L} \subseteq \mathcal{L}^0$.

When constructing such a line pool, two conflicting goals need to be considered: On the one hand, the line pool should be large enough to allow the line planning model in Phase 2 to find a good line concept. On the other hand, the larger the line

pool is, the more complex it is to solve the line planning model in Phase 2. Hence, we require the number of lines in the line pool to be bounded by a value K to ensure tractability of Phase 2. To ensure that the line pool is 'good enough' is much harder. A minimal requirement for a good line pool is that it should at least be *feasible* for the line planning problem in Phase 2. Since there are so many different models for line planning it is unrealistic to guarantee feasibility for all possible models. However, the edge frequency constraints (1) are a minimal requirement which is used in most line planning models, and which we hence consider for designing a line pool.

We can now define the *line pool generation problem* for given K as follows.

(**LPool**) Given the PTN=(V, E), lower and upper frequency bounds $f_e^{\min} \le f_e^{\max}$ for all $e \in E$, and the set $\mathcal{L}^0$ of all cycle free paths in the PTN, find a set $\mathcal{L} \subseteq \mathcal{L}^0$ such that $|\mathcal{L}| \le K$ and such that there exist $f_l \in \mathbb{N}_0$ for all $l \in \mathcal{L}$ with

$$f_e^{min} \le \sum_{\substack{l \in \mathcal{L}: \\ e \in l}} f_l \le f_e^{max} \qquad \forall\, e \in E.$$

Using binary variables

$$x_l = \begin{cases} 1 & \text{if line } l \text{ is chosen to be in the line pool} \\ 0 & \text{otherwise} \end{cases}$$

and variables $f_l \in \mathbb{N}_0$ to ensure the existence of a line concept, LPool can be formulated as the following integer program. To this end, let $M \ge \max_{e \in E} f_e^{\max}$.

$$\text{Find} \quad x_l \tag{2}$$

$$\text{s.t.} \sum_{l \in \mathcal{L}^0} x_l \quad \le \quad K \tag{3}$$

$$\sum_{\substack{l \in \mathcal{L}^0: \\ e \in l}} f_l \quad \ge \quad f_e^{min} \quad \forall\, e \in E \tag{4}$$

$$\sum_{\substack{l \in \mathcal{L}^0: \\ e \in l}} f_l \quad \le \quad f_e^{max} \quad \forall\, e \in E \tag{5}$$

$$f_l \quad \le \quad x_l M \quad \forall l \in \mathcal{L}^0 \tag{6}$$

$$f_l \quad \in \quad \mathbb{N}_0 \quad \forall l \in \mathcal{L}^0 \tag{7}$$

$$x_l \quad \in \quad \{0, 1\} \quad \forall l \in \mathcal{L}^0 \tag{8}$$

Note that this is a feasibility problem only. We are mainly interested in the variables x_l since these variables determine the line pool $\mathcal{L} = \{l : x_l = 1\}$. However, a line pool is only feasible if there exists at least one line concept satisfying the basic

constraint (1). This is ensured by constraints (4) and (5) and our additional variables f_l. The number of lines in the pool is bounded by constraint (3). (6) finally is needed to ensure that $f_l = 0$ if $x_l = 0$, i.e., if the line is not in the pool.

In order to compare this formulation with classic line planning models, we look at the special case for $K \geq |\mathcal{L}^0|$. In this case, constraint (3) is redundant. We hence may set $x_l = 1$ for all lines $l \in \mathcal{L}^0$ meaning that also (6) can be dropped from the IP formulation. Consequently, we are only left with the variables f_l, i.e., in the special case of $K \geq |\mathcal{L}^0|$ (2)–(8) is equivalent to

$$\text{Find} \quad f_l \tag{9}$$

$$\text{s.t.} \sum_{\substack{l \in \mathcal{L}^0: \\ e \in l}} f_l \quad \geq \quad f_e^{min} \quad \forall\, e \in E \tag{10}$$

$$\sum_{\substack{l \in \mathcal{L}^0: \\ e \in l}} f_l \quad \leq \quad f_e^{max} \quad \forall\, e \in E \tag{11}$$

$$f_l \quad \in \quad \mathbb{N}_0 \quad \forall l \in \mathcal{L}^0. \tag{12}$$

which is the basic line planning problem LP0 from above for the line pool $\mathcal{L}^0$. If $\mathcal{L}^0$ is given as an arbitrary set of lines, it is known that this problem is NP-hard, see Claessens et al. (1998). However, here $\mathcal{L}^0$ has a special form: it consists of all cycle free paths. This makes the problem (9)–(12) easily solvable: Namely, choose a line

$$l_e := \{e\}$$

for each edge $e \in E$ which consists of the respective edge only, and set

$$f_{l_e} := f_e^{\min} \quad \text{for all } e \in E.$$

The question now arises if LPool is NP-hard for smaller K even in the case that $\mathcal{L}^0$ contains all cycle free paths. The following theorem answers this question.

Theorem 2.1 *Problem LPool is NP-hard even if* $K = 1$.

Proof Consider the following reduction of the directed Hamiltonian Path Problem to LPool.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a digraph. Construct an undirected PTN $G = (V, E)$ to determine whether a Hamiltonian Path exists in $\mathcal{G}$. Define

$$\begin{aligned} V &= \{v_{in}, v_{out} : v \in \mathcal{V}\}, \\ E &= E_1 \cup E_2, \\ E_1 &= \{\{v_{out}, u_{in}\} : (v, u) \in \mathcal{E}\}, \\ E_2 &= \{\{v_{in}, v_{out}\} : v \in \mathcal{V}\}. \end{aligned}$$

The edge set E_1 corresponds to the edges in the original graph while the set E_2 corresponds to the vertices and is used to give the orientation of the edges in $\mathcal{E}$ to the

corresponding ones in E_1.

Let $\mathcal{L}^0$ be the set of all possible cycle free paths in G. Set $K = 1$ such that only one line may be chosen for the pool $\mathcal{L}$. Upper frequency bounds are not needed. The lower frequencies $f_e^{\min}$ are set to 0 for all edges in E_1 and to 1 for all edges in E_2. Therefore, all edges connecting in- and outgoing vertices belonging to the same original vertex have to be used once.

Claim *There exists a solution to LPool in G if and only if there exists a Hamiltonian Path in $\mathcal{G}$ (see Fig. 1 for an illustration).*

$\Rightarrow$ Let $\mathcal{L}$ be a solution to LPool. Then $\mathcal{L} = \{P\}$ for some path P through the PTN which contains all edges in E_2. In the path P each node has at most degree two and each edge in E_2 is used once. As each node is exactly incident to one edge in E_2, the edges of P are alternately from E_1 and E_2. Since the edges in E_1 go from an outgoing to an incoming vertex, the path P in G can be transformed into a directed path P' in $\mathcal{G}$ by omitting the edges in E_2. As P uses each edge in E_2, P' traverses each node in $\mathcal{V}$. Therefore, P' is a Hamiltonian Path.
$\Leftarrow$ For a Hamiltonian Path P in $\mathcal{G}$ construct the corresponding undirected path P' in G. Therefore, change the edges from $\mathcal{E}$ into the corresponding ones in E_1 and insert the edges in E_2 for each vertex. This is a cycle free path in G which contains all edges in E_2 as P traverses all vertices $\mathcal{V}$. Using P' as line pool with frequency $f_{P'} = 1$ is, thus, a feasible solution for LPool.

□

Note that the problem is also NP-hard if we allow *all* simple paths in $\mathcal{L}^0$ instead of only cycle free paths.

3 Solution approach for generating a line pool

The algorithm we propose aims at generating line pools which are not too large, allow feasible line concepts and deliver good lines. Our algorithm iteratively adds lines and stops as soon as a feasible line concept is found. The quality of the

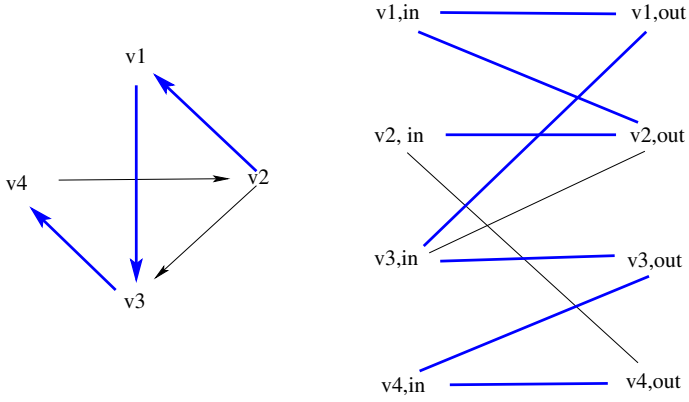


Fig. 1 Reduction of Hamiltonian Path to LPool in the proof of Theorem 2.1

resulting lines cannot be determined beforehand, but is evaluated experimentally in Sect. 4.

As input, our algorithm needs a PTN $= (V, E)$. Furthermore, passenger information given as an OD matrix $C = (C_{uv})_{u,v \in V}$ where C_{uv} is the number of passengers who wish to travel from u to v as well as lower and upper frequency bounds $f_e^{min} \leq f_e^{max}$ for all edges E are required. The algorithm is then based on the idea of repeatedly generating paths on minimal spanning trees (MST) of the PTN and adding them to the line pool. The algorithm stops when a feasible line concept is contained in the line pool and hence tends to find line pools which are rather small. Consequently, we do not use a maximum number of lines as input, but determine the size of the line pool afterwards.

3.1 Generation of valid lines

Let a non-empty set of lines $\mathcal{L}$ be given. If this set of lines is not a feasible solution to the line pool generation problem, we have to add further lines to it. A line l to be added to $\mathcal{L}$ has to satisfy constraints on itself and constraints which involve the line l together with the current line pool $\mathcal{L}$. In order to generate lines we proceed as follows:

(1) We identify *terminals*.
(2) We determine a spanning tree of the PTN with its leaves.
(3) We generate lines which connect either two leaves, or two terminals, or a terminal with a leaf.

Let $\mathcal{L} \neq \emptyset$ be a set of lines. For each node we compute its *line degree*

$$deg(v, \mathcal{L}) = |\{l \in \mathcal{L} : v \in l\}|.$$

Let $deg(\mathcal{L})^{\max}$ be the maximum number of lines passing through the same node. Then a node is called a *terminal* if

$$deg(v, \mathcal{L}) \geq \texttt{terminal\_degree} \cdot deg(\mathcal{L})^{\max}$$

for some given parameter $\texttt{terminal\_degree} \in (0, 1)$. Note that in the very first iteration, we have $\mathcal{L} = \emptyset$. In this case we use the node degree within the PTN instead of the line degree to determine terminals.

In the next step we determine a minimal spanning tree. A *minimal spanning tree* is a connected subgraph G' of the PTN with $G' = (V, E') \subseteq G$, $|E'| = |V| - 1$ and $\sum_{e \in E'} w_e$ minimal. It can be found, e.g., with Kruskal's algorithm (Kruskal 1956).

With the edge weights for the spanning tree problem we can influence which edges are likely to be contained in the tree. The weight of the edges is changed in each iteration. We denote w_e as the *weight* and l_e as the *length* of edge $e \in E$.

A line l' is called *valid* for a line pool $\mathcal{L}$ and an MST if

- $l' \notin \mathcal{L}$

- it is a terminal-terminal, terminal-leaf or leaf-leaf path
  - if l' is a leaf-leaf path: the euclidean distance between the two end nodes of l' is at least `min_distance`
  - else: l' contains at least `min_edges`.

We end this section by giving a motivation for using leaf-terminal lines: Looking at subgraphs of a tree consisting of a terminal-leaf connection yields a linear graph. In our heuristic, the edges of this linear graph form a single line, with no other line starting or ending in between. In the following proposition we give a reason why such a line should be added to the line pool (if it is valid). This reason is based on the objective function used for finding the line concept in Phase 2: namely, such a line is likely to be included in a line concept since it has low costs (compared to its length) and is also one with the maximum number of direct travelers.

Proposition 3.1 *Let the PTN be a linear graph* $G = (V, E)$, $f_e^{min} > 0 \ \forall e \in E$, $K = \infty$, $\mathcal{L}$ *the set of all paths in* G *and the line-costs are of the form* $cost_l = F + \sum_{e \in E} c_e$. *Here* F *is a fixed cost-term and* c_e *are edge-dependent costs.*

Let l *be the line between the two leaves of the linear PTN, i.e., the line consisting of all edges. Then there*

(1) *exists an optimal solution to LP-Cost which contains line* l *with frequency* $f_l = \min_{e \in E} f_e^{min}$
(2) *exists an optimal solution to LP-Direct which contains line* l *if the upper edge frequencies* f_e^{max} *are large enough.*

Proof

(1) Consider an optimal solution to LP-Cost with $f_l < \min_{e \in E} f_e^{min}$. Then there exists a set of lines in $\mathcal{L} \backslash \{l\}$ with positive frequencies, such that each edge is covered at least once. Let $\mathcal{L}' = \{l_1, \ldots, l_m\}$ be such a set, where additionally no line is contained in another one and the lines are ordered according to their first edge. Now consider the set of lines which are intersections of two consecutive lines

$$\mathcal{L}'' = \{l_1 \cap l_2, l_2 \cap l_3, \ldots, l_{m-1} \cap l_m\} \cup \{l\}.$$

It is easy to see that each edge is covered by $\mathcal{L}'$ as often as by $\mathcal{L}''$. Increasing the frequencies of each line in $\mathcal{L}''$ and decreasing the frequencies of each line in $\mathcal{L}'$ leads to a new solution which can be shown to be feasible and optimal by some calculations and yields a larger value for f_l. Iterating this process until $f_l = \min_{e \in E} f_e^{min}$ proves the claim.

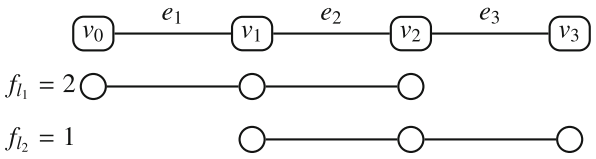
(2) The second assertion is clear since every OD-pair in the linear graph can travel without a transfer if line l is contained within the line concept.

□

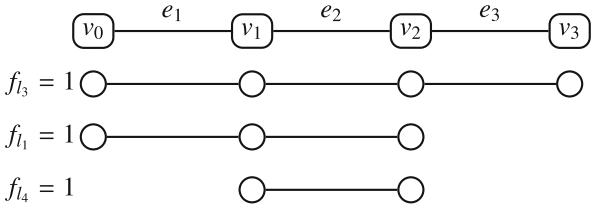
Remark 3.2 We can apply the idea of the proof of Proposition 3.1 to any set of lines covering the same connected component of a linear PTN. Using this approach recursively gives us an optimal solution to LP-Cost in which for all pairs of lines l_1, l_2 with positive frequencies we have that either $l_1 \subseteq l_2$ or $l_2 \subseteq l_1$ or $l_1 \cap l_2 = \emptyset$. For details, see Gattermann (2015).

The following example shows the limitation of adding the longest path as a line when the direct traveler model is concerned. Here, upper frequency bounds and relatively small vehicle capacities may lead to shorter lines being preferred.

Example 3.3 Consider a linear PTN consisting of four nodes $V = \{v_0, \ldots, v_3\}$ and three edges $E = \{e_1, e_2, e_3\}$. Let the upper and lower frequencies for each edge be identical with $f_{e_1}^{\min} = f_{e_1}^{\max} = 2$, $f_{e_2}^{\min} = f_{e_2}^{\max} = 3$ and $f_{e_3}^{\min} = f_{e_3}^{\max} = 1$. Let the capacity of the vehicle be one and the demand $C_{u,v}$ between stations u and v be given as $C_{v_0,v_2} = 2$, $C_{v_1,v_3} = 1$ and $C_{u,v} = 0$ otherwise. Consider the lines $l_1 = (e_1, e_2)$ and $l_2 = (e_2, e_3)$ with the corresponding line concept $f_{l_1} = 2$, $f_{l_2} = 1$ and $f_l = 0$ for all other possible lines l. This line concept allows all passengers to travel directly, so the total number of direct travelers is three.



If the longest line $l_3 = (e_1, e_2, e_3)$ is used with frequency $\bar{f}_{l_3} = 1$, the best possible line concept is $\bar{f}_{l_3} = 1$, $\bar{f}_{l_1} = 1$ and $\bar{f}_{l_4} = 1$ with $l_4 = (e_2)$. Due to the capacity constraint this line concept allows only two direct travelers and is therefore not optimal.



3.2 Generation of a line pool

We finally can describe our algorithm for generating a line pool. The generation of the single lines to be added is based on the theoretical results of the previous section.

Algorithm 1 Main function for generating a line pool

```
1: function MAIN
2:     pool= ∅
3:     preferred_edges= ∅
4:     while pool infeasible and number_of_iterations not reached do
5:         if first loop then
6:             Calculate shortest paths for all passengers in PTN.
7:             For each edge determine the number of passengers using it.
8:             Add the pass_edge_ratio most used edges to preferred_edges.
9:         else
10:            if feasible line concept (ℒ, f) found then
11:                return Line pool pool
12:            else
13:                Set f̄^min = f^min
14:                while no feasible line concept found do
15:                    Find a line concept (ℒ̄, f̄) with f̄^min in (1).
16:                    if no such line concept exists then
17:                        set f̄_e^min = f̄_e^min − 1 for all e ∈ E.
18:                    end if
19:                end while
20:            end if
21:            for edge e ∈ E with Σ_{l∈pool:e∈l} f̄_l < f_e^min do
22:                preferred_edges=preferred_edges ∪{e}
23:            end for
24:        end if
25:        while pool increased and preferred_edges ≠ ∅ do
26:            Find MST in PTN, with
27:            w_e = 0 for all e ∈ preferred_edges and
28:            w_e = l_e for all e ∉ preferred_edges.
29:            Determine leaves and terminals based on terminal_degree.
30:            POOL_FROM_MST(pool, MST, terminals, preferred_edges)
31:        end while
32:    end while
33: end function
```

The algorithm consists of an outer loop which checks if the line pool found so far is feasible, and if not, determines the parameters for the inner loop, in which spanning trees are constructed and used to generate lines.

In the outer loop (see Algorithm 1) we start with routing all passengers through the network in order to see which edges are used by many passengers. The `pass_edge_ratio`% most used edges are set to be preferred, i.e., their weight for determining an MST is set to zero. All other edges keep their physical length as weight. We then determine a first MST from which we generate valid lines in the inner loop.

All other iterations of the outer loop start with a non-empty line pool. In each iteration we check if this line pool is feasible according to (1). If it is, the algorithm stops. Otherwise we want to identify edges which should be used for generating new lines. To this end, we decrease the lower frequencies until we find a feasible line concept within our pool (lines 14–19). This line concept is not feasible for at least one

of the original lower edge frequency constraints. We determine all edges e for which (1) is not satisfied (line 21) and add them to the preferred edges with zero weight in order to make it likely that they are included in the next spanning tree (line 22). In each step of the *inner loop* (25–31) we determine a spanning tree and identify terminal nodes.

Algorithm 2 Function computing lines based on one MST

```
1: function POOL_FROM_MST(pool, MST, terminals, preferred_edges)
2:     Initialize unfinished_lines = {terminals} ∪ {leafs}
3:     for line ∈ unfinished_lines do
4:         Delete line from unfinished_lines
5:         stop ← last stop in line
6:         for edge ∉ line incident to stop do
7:             Create new_line l′ by adding edge to line.
8:             if new_line l′ is not valid then
9:                 Add new_line l′ to unfinished_lines.
10:            else if |{l ∈ ℒ ∪ {l′} : e ∈ l}| ≤ f_e^max · max_num_cover and
11:                    new_line l′ contains an edge from preferred_edges then
12:                Add new_line l′ to pool.
13:                for edge e ∈ new_line l′ do
14:                    if ⌈f_e^min / min_num_cover⌉ ≤ |{l ∈ pool : e ∈ l}| then
15:                        Remove e from preferred_edges
16:                    end if
17:                end for
18:            end if
19:        end for
20:    end for
21: end function
```

We then use the MST and the terminals to generate valid lines in Algorithm 2. This is done by adding additional edges to non-valid lines until they reach the minimal length `min_edges` required for paths starting or ending at a terminal. A valid line is only added to the line pool if it contains at least one preferrable edge, and if it does not increase the frequency of its edges too much, i.e., if

$$|\{l \in \mathcal{L} : e \in l\}| \leq f_e^{max} \cdot \texttt{max\_num\_cover}$$

holds. If a line is added which contains a preferred edge e we might want to remove edge e from the list of preferred edges. This is done if

$$\left\lceil \frac{f_e^{min}}{\texttt{min\_num\_cover}} \right\rceil \leq |\{l \in \mathcal{L} : e \in l\}|$$

is fulfilled. After a spanning tree is processed, a new spanning tree is constructed and the process is repeated until there are no preferred edges left, or no further line can be added.

Afterwards the feasibility of the pool is checked by solving a line planning problem using the pool. The algorithm returns an error if no feasible line concept can be constructed. Note that the pool may only be infeasible if the maximal number of iterations of the outer loop is reached.

3.3 Summary of algorithm parameters

We summarize the list of parameters which control the algorithm.

- `pass_edge_ratio`: Used in the first iteration to obtain preferred edges. More precisely, the `pass_edge_ratio` percent of most used edges are chosen as preferred edges.
- `min_num_cover`: For each edge, the lower frequency bound divided by `min_num_cover` bounds the number of lines containing this edge, i.e., for all $e \in E$ we require $\left\lceil \frac{f_e^{min}}{\texttt{min\_num\_cover}} \right\rceil \leq |\{l \in \mathcal{L} : e \in l\}|$.
- `terminal_degree`: Deviation from maximum line degree such that a node is labeled terminal.
- `min_edges`: Minimum number of edges per line (only for lines from or to a terminal).
- `min_distance`: Minimum euclidean distance between start and end station (only for lines from and to a leaf).
- `max_num_cover`: For each edge, the upper frequency bound times `max_num_cover` bounds the maximal coverage of this edge, i.e., for all $e \in E$ we require $|\{l \in \mathcal{L} : e \in l\}| \leq f_e^{max} \cdot \texttt{max\_num\_cover}$.

Whereas the first two parameters are only used to determine the edges which are preferred in the computation of the MSTs the other parameters are related to the validity of a line.

3.4 Determining the costs

In order to use the line pool generated by our approach for line planning in Phase 2, we have to add costs for each line in the pool. The costs of a line are influenced by various parameters from which we take the following three into account:

- length of line (`costs_length`)
- number of edges traversed (`costs_edges`)
- fixed costs (`costs_fixed`)

Then the costs for a line $l \in \mathcal{L}$ are computed as

$$cost_l = \texttt{costs\_length} \sum_{e \in l} l_e + \texttt{costs\_edges}\,|\{e \in l\}| + \texttt{costs\_fixed}.$$

4 Results

In order to study the performance of Algorithm 1 we apply it to generate line pools in a first step. In the second step we used the cost model and the direct travelers model on the generated line pools.

We used two different network types. The first are star shaped networks. In these graphs we can use an algorithm of Gattermann (2015) to optimally solve the cost model even if all possible paths are allowed as lines. This gives the best possible objective function value which can be used as lower bound for our two-step approach. Our second network is close to the network of the German railways. It is used to show its general applicability for realistically sized and shaped instances.

4.1 Results for stars

Our first data instance set comprises different star shaped graphs.

4.1.1 Data set

We generated 100 star graphs as follows. For every $n \in \{10, 100, 250, 500\}$ we generated 25 stars with a number of edges (and hence nodes) chosen randomly from $\{1, \ldots, n\}$. The upper frequency bound is set to $f_e^{max} = 20$ for all edges and the lower frequency bound is randomly determined between 0 and 20. The costs of the edges are aslo randomly chosen with values between 0 and 10 and the fixed costs of a line are set to 10.

4.1.2 Experiments

The parameter setting for the star shaped graphs is according to Table 1. Note that the parameters `min_num_cover` and `max_num_cover` take two different values dependent on the respective instance. The test instance with at most 500 edges per graph requires too much memory to be computable with the setting used for smaller instances. Hence, we set `min_num_cover` to 4 and `max_num_cover` to 1 here. For all other instances `min_num_cover` and `max_num_cover` are set to 1 and 5, respectively, to allow many lines in the pool. The parameter `min_edges` is set to 1 to allow lines consisting of only one edge and the parameter `min_distance_leaves` is set to 0 to allow lines consisting of any pair of edges. In stars, the parameters `ratio_od` and `node_degree_ratio` do not have any effect on the algorithm as there is only one possible minimal spanning tree and only one possible terminal. Therefore, these parameters are not changed during the tests.

Table 1 Parameters for Algorithm 1 on randomly generated stars

Parameter name	Parameter value
`min_num_cover`	1 (4)
`max_num_cover`	5 (1)
`number_of_iterations`	5
`pass_edge_ratio`	0.5
`terminal_degree`	0.5
`min_edges`	1
`min_distance`	0

4.1.3 Evaluation parameters

For every star shaped graph, we can easily compute the optimal objective value of LP-Cost for a line pool which contains all possible simple paths (see Gattermann 2015). This value is then compared to the objective value of LP-Cost for the line pool obtained from applying Algorithm 1 to the graph.

4.1.4 Results

The results of testing Algorithm 1 on the randomly generated stars are shown in Table 2.

We observe that especially for smaller stars (up to 100 edges) the heuristic solution found by Algorithm 1 is very good. In more than 75 % of all cases the optimal solution is found by the heuristic and the objective values of the heuristic solutions deviate from the optimal objective value on average only between 1 and 3 % and at most by 21 %. When using larger stars (up to 250 edges) this quotient increases but the objective values still lie on average only 14 % and at most 27 % above the optimal values. That the quality of the line pool gets worse when proceeding to stars with up to 500 edges is mainly due to changing the parameters `min_cover_factor` and `max_cover_factor` which leads to considerably fewer lines in the pools and hence to line pools with reduced quality: On average we obtain 697.28 lines for stars consisting of 250.96 edges opposed to 1242.84 lines for stars consisting of 139.36 edges. But in turn, the decreased number of lines also leads to a decrease in runtime.

Overall, the heuristic Algorithm 1 works quite well especially for smaller stars, although this depends on choosing the parameters such that enough lines are added to the pool. We remark that in order to achieve a good approximation of the optimal solution, only a fraction of all lines is needed in the pool. The discrepancy between the possible number of lines and the number of lines which make up the pools shows that Algorithm 1 is a good example for line pool generation as a preprocessing step for line planning. Especially for larger PTNs the number of lines which have to be considered when finding a line concept is reduced such that further computations are achievable in a reasonable time.

Table 2 Results for randomly generated stars

Max. number of edges	10	100	250	500
Av. number of edges	4.72	39.68	139.36	250.96
Number of optimal solutions	19	20	4	0
Ratio of optimal solutions	0.76	0.80	0.16	0
Av. deviation from optimal value	1.03	1.01	1.14	1.36
Max. deviation from optimal value	1.21	1.15	1.27	1.41
Av. number of undirected lines	12.60	308.52	1242.84	697.28
Av. number of possible undirected lines	17.64	1177.40	11,957.32	39,580.60
Av. runtime in seconds	5.36	34.08	97.28	90.64

4.2 Results for the German railway network

4.2.1 Data set

The other data set is generated from the German high speed railway network. It consists of $|V| = 250$ nodes and $|E| = 326$ edges. The frequency bounds are given as $f_e^{min} \in [0, 5]$ and $f_e^{max} = 15$ for all $e \in E$. Different settings for each of the seven parameters listed in Sect. 3.3 are tested.

4.2.2 Experiments

Experiments on the Germain railway network are done with LinTim (see Goerigk et al. 2015). Based on the data set and one specific parameter setting a line pool is generated. The resulting line pool is then taken as input for the cost line planning model LP-Cost and the direct travelers line planning model LP-Direct, see Sect. 2.1. These IP programs are solved with FICO Xpress. We obtain the objective function value for both models: the minimal possible costs and the maximal possible number of direct travelers.

The algorithm has been executed with a variety of different parameter patterns. In total 624 different settings have been tested, and for each of them a line pool has been generated.

4.2.3 Evaluation parameters

The quality of a line pool is evaluated by two rather opposing parameters. The first parameter gives the lowest possible costs which are determined as the objective function value of the cost-optimal line concept. The second parameter reflects the passengers' point of view. Here, we measure the number of direct travelers that can travel on their shortest paths in the PTN in the direct-travelers optimal solution. Let a line pool $\mathcal{L}$ be given.

- Costs: We solve LP-Cost using the costs $cost_l$ for each line $l \in \mathcal{L}$ according to Sect. 3.4.
- Direct Travelers: We solve LP-Direct. To this end, preferable paths are needed. In our computation we use for each OD-pair $(u, v) \in C$ only one preferable path, namely the shortest path p from u to v in the PTN. The union $\mathcal{P} = \bigcup p$ gives the set of paths for all OD-pairs. Based on the paths $\mathcal{P}$, the number of direct travelers on each $p \in \mathcal{P}$ is computed. For a more detailed description of the definition and optimization of direct travelers we refer to Bussieck et al. (1996).

4.2.4 Results

Feasibility The first result shows the reliability of the algorithm on the German railway network. Most of the line pools (87 %) obtained by any of the parameter settings are feasible solutions to LPool, i.e., they contain a feasible line concept.

We further investigated which particular parameter settings have a recognizable influence on the feasibility.

In Fig. 2 the percentage of infeasible line pools in relation to the iteration of Algorithm 1 is depicted. Until iteration 3 no line pool is feasible whereas from iteration 7 on the percentage of feasible line pools only increases slightly. Hence, for most parameter settings between 4 and 7 iterations are needed to receive a feasible line pool.

In Figs. 3 and 4 the influence of the parameters `min_distance` and `terminal_degree` on the feasibility of the line pools is studied. It can be recognized that the percentage of feasible line pools is influenced by the choice of these parameters. E.g., requiring that the end nodes of leaf-leaf lines are far from each other, i.e., `min_distance` $= 15$, results in 39 % of infeasible line pools after 6 iterations. Setting `min_distance` to a smaller value, e.g., 5, only 2 % of infeasible line pools are obtained after 7 iterations.

The percentage of feasible line pools also depends on the setting of `terminal_degree`. While a value of 0.9 results in 31 % of infeasible line pools after 7 iterations, setting the parameter to 0.7 results in only 10 % of infeasible line pools.

Considering all tested settings, minus the ones with `min_distance` $= 15$ and `terminal_degree` $= 0.9$, the percentage of feasible line pools after 7 iterations reaches 95 % and even 97 % after 16 iterations. The other parameter settings do not show any significant effects on the feasibility of the resulting pools.

Running times In addition to the feasibility, also the running times of the subsequent line planning models are important. The computation times for subsequent line planning models are low as the number of lines in the pools range between 115 and 342. The range of computation time for the cost line planning model is from 1.0 to 49.0 s, with a mean of 1.59 s. For the direct travelers model, the computation time ranges between 6.0 and 94.0 s and is on average 11.95 s.

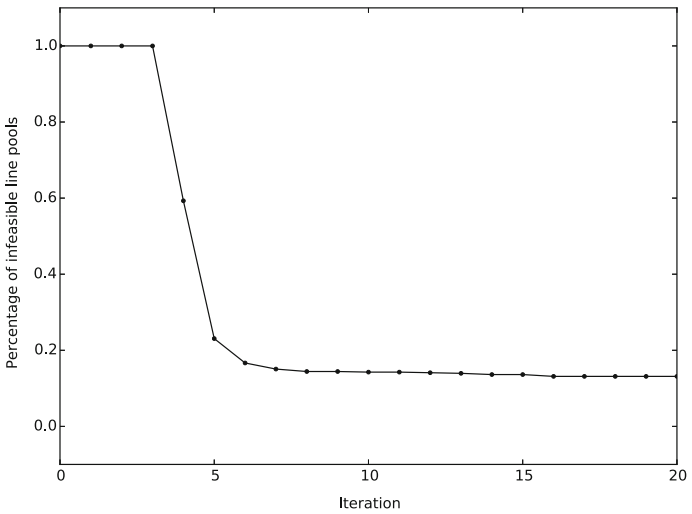


Fig. 2 Feasibility of all line pools

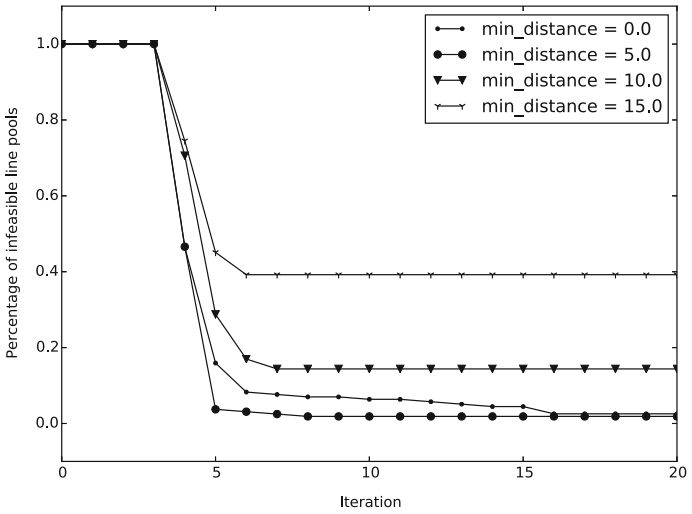


Fig. 3 Feasibility of line pools under parameter `min_distance`

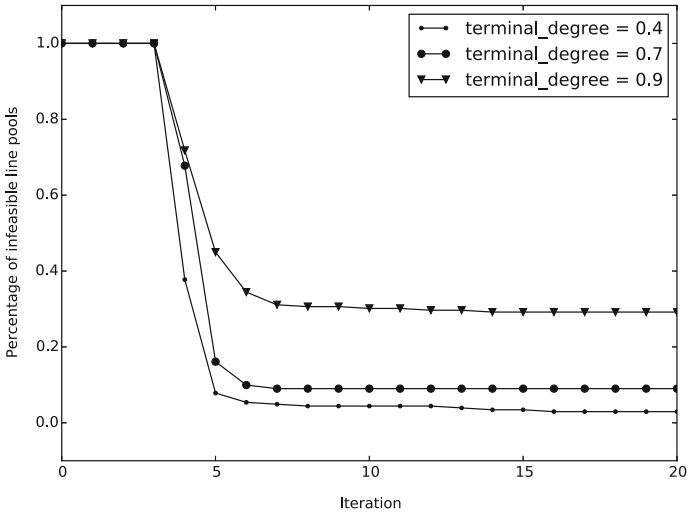


Fig. 4 Feasibility of line pools under parameter `terminal_degree`

What are the influences of the parameters? We now want to study the impact of different algorithm parameters on the evaluation parameters. As described, the resulting line pools are used as input for the line planning problem using the cost model and the direct travelers approach. The result is indicated in Fig. 5. Each point in this figure reflects a line pool. It shows the objective values of the cost model and of the direct travelers model. In order to understand the influence of the parameters we additionally show three attributes for each point which represent the parameters

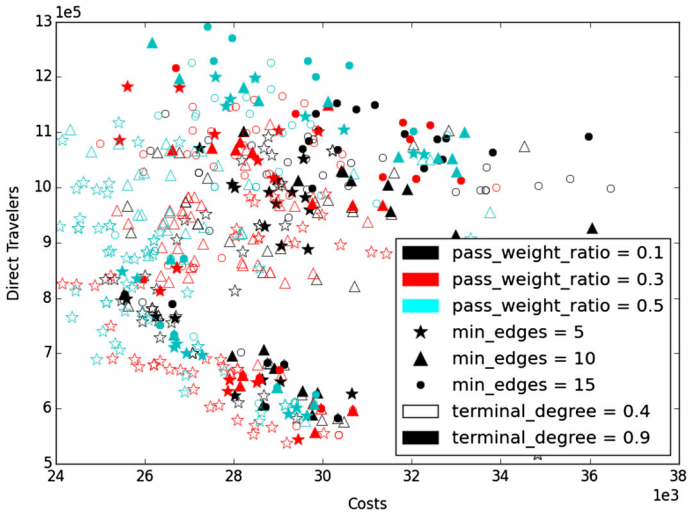


Fig. 5 Different line pools evaluated for costs and direct travelers

used for generating the respective line pool. A point is specified by a particular setting for `min_edges` (star, triangle, circle), `pass_edge_ratio` (black, red/grey, cyan/light grey) and `terminal_degree` (contour, filled).

How long should a line be? We start with an analysis of the parameter `min_edges` which specifies how many edges are required in a line.

Firstly, in order to show the implication of the choice of `min_edges`, we study how many edges the lines in the line pools have. For every pool we evaluate the number of edges in the shortest line and the average number of edges per line in the pool. The question is how these values are influenced by the parameter `min_edges`.

Setting `min_edges` to 5 results in an average number of 10.09 edges per line and the shortest line having 2 edge. Increasing `min_edges` to 10 the average number of edges per line also increases to 11.82 whereas the shortest line still has 2 edges. Finally, setting `min_edges` to 15 gives the same number of edges in the shortest line and the average number of edges in a line again raises to 13.89. The same structure can indeed be found in the resulting line concepts.

While the shortest line always contains 2 edges, the average number of edges per line increases if `min_edges` is increased. Analyzing these shortest lines, we note that only leaf-leaf lines are not restricted in the number of edges. Hence, lines shorter than `min_edges` are only possible if leaf-leaf lines exist whose endpoints have a larger euclidean distance than `min_distance`. If the parameter `min_distance` is set to a larger number, then increasing `min_edges` in fact also leads to more edges in the shortest line. This is confirmed by our experiments: Choosing `min_distance` as 5 we obtain that the shortest line contains 5 edges for any setting of `min_edges`. Increasing `min_distance` to 15 the shortest line contains exactly as many edges as specified by `min_edges`.

We summarize that there is a clear effect of the parameter min_edges on the average number of edges per line. The minimal number of edges per line is then influenced by both parameters min_edges and min_distance.

We now study the implication of the parameter min_edges to our evaluation parameters. A clear correspondence to the number of direct travelers can be observed. A value of 5 for min_edges results in an average number of direct travelers of 838,761.65. The values 10 and 15 result in average direct travelers of 901,603.33 and 958,793.89, respectively. It indicates that the more edges are required per line the higher the number of direct travelers until some upper bound is reached. In our computations it turned out that setting min_edges to 20 gives no significant improvement in relation to min_edges = 15. A relation between the parameter min_edges and the costs of the cost optimal line concept cannot be recognized.

How many terminals should be generated? We now turn our attention to the parameter terminal_degree which determines how many terminals are generated in each step. Figure 5 shows that generating less terminals produces better line pools in terms of the direct travelers. The relation is understandable as lines have to combine either leaf and leaf, terminal and leaf or terminal and terminal nodes. Reducing the number of terminals has two effects: First, fewer terminals lead to longer lines and longer lines are more likely to be direct connections for passengers than shorter lines. Moreover, the less terminals are available, the more leaf to leaf lines are obtained, which allow more passengers to travel directly instead of transferring at terminal nodes. Indeed, choosing terminal_degree as 0.4 the average number of direct travelers results in 850,925.84. Restricting the number of terminals by using a value of 0.7 for terminal_degree, an average number of 928,457.54 direct travelers is obtained. Increasing the value for terminal_degree further does not seem to make an impact as the average number of direct travelers for terminal_degree $= 0.9$ is 927,327.91.

Figure 5 does only show a slight correlation between the parameter terminal_degree and the costs of the cost optimal line concept. Choosing terminal_degree as 0.4 leads to average costs of 28,109.94, while average costs of 29,011.20 are obtained by setting this parameter to 0.7. Setting the paramter to 0.9 finally results in an average cost of 29,804.12. Hence, small values of terminal_degree seem to have the tendency of reducing the costs of the cost optimal line concept.

How many lines should pass through an edge? The parameters min_num_cover and max_num_cover have different effects. The parameter min_num_cover ensures that each edge is traversed by enough lines. The higher this parameter, the fewer lines are needed to cover each edge. Hence, with increasing value of min_num_cover the average number of lines per pool decreases. This seems to have a negative effect on the average number of direct travelers as the following numbers show: For min_num_cover $= 1$, the average number of lines in the pool is 481.49 and the average number of direct travelers is 950,488.90 while for min_num_cover $= 2$ (or 4) the line pool size is on average 358.60 (306.12) and the number of direct travelers is on average 887,568.32 (857,291.88). An effect on the costs can also be recognized. min_num_cover $= 1$ results in average costs

of 27,698.52. Increasing this value to 2 (and 4), also the costs increase to 29,273.31 (and 29,574.48). This may be due to the fact that the lines get shorter and hence more expensive compared to their lengths.

The parameter `max_num_cover` specifies the maximum number of lines per edge. Our experiments show that this parameter does not influence the solution quality. It is also recognizable that lines under the setting `max_num_cover` $= 3$ have slightly less edges than for `max_num_cover` $= 5$. The difference between the averages is about 0.8 edges but this difference does not show an impact on the solution quality.

How far should start- and endpoint of a line be apart from each other? We already saw some effects of the parameter `min_distance` (specifying the minimal euclidean distance between the start - and the endpoint of a line) together with the minimal number of edges of a line. Further results concerning this parameter are depicted in Fig. 6. In particular, we tested the outcome if this condition is completely ommitted, i.e., if also lines are allowed that connect two endpoints that are close to each other (and hence may make very large detours). To this end we compared the solutions for `min_distance` $= 0$ with `min_distance` $= 10$.

The figure shows that allowing lines between any pair of leaf nodes (i.e., for the case of `min_distance` $= 0$) has no effect on the costs (except of some outliers with high cost values in the case of `min_distance` $= 10$) but decreases the best possible number of direct travelers. The average number of direct travelers for `min_distance` $= 0$ is 694,184.77 while the average over all experiments with `min_distance` > 0 was 985,411.94.

Other observations Finally, the parameter `pass_edge_ratio` does not seem to have a strong impact on any evaluation parameter. This can be explained, since the parameter is only important in the first outer loop, where according to this

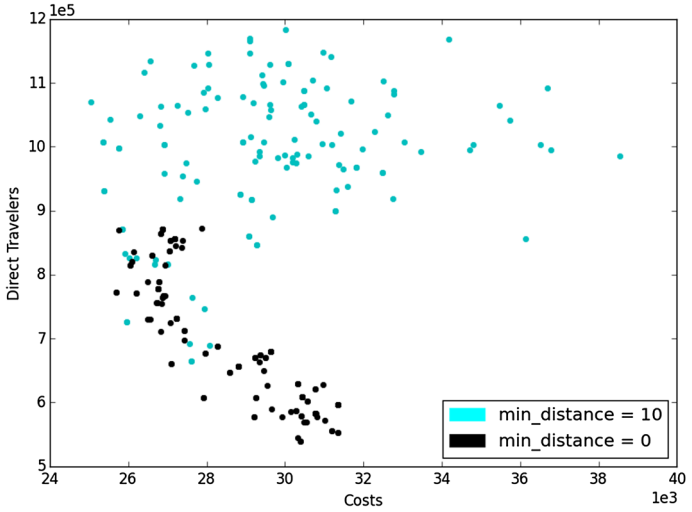


Fig. 6 Different line pools evaluated for costs and direct travelers, color coding by `min_distance` parameter

parameter passenger weighted edges are preferrably considered in the minimal spanning tree.

We also investigated if there is a correspondence between the number of lines in the line pools we generated and the quality of the evaluation parameters. It turned out that the costs get only a little bit better for larger line pools while there is no clear relation to the number of direct travelers.

Summary for the German railway network Our experiments show a strong dependency between the quality of the line plan and the parameters used in the line pool generation algorithm. For receiving a good line concept it is hence crucial *which* lines have been added to the line pool, i.e., not only the quantity of lines in the pool but also their quality plays an important role as Fig. 7 shows. This underlines the fact that it is possible to generate good line pools of moderate size. Our main findings are as follows:

- Increasing `min_edges`, i.e., requiring a high minimum number of edges per line improves the number of direct travelers (up to some bound).
- Increasing `min_num_cover`, i.e., the number of lines that have to traverse an edge, reduces the number of direct travelers and increases the costs.
- Increasing `min_distance` (i.e., allowing only lines between stations with a minimal euclidean distance) leads to more direct travelers.
- Increasing `terminal_degree` (i.e., allowing less terminals) increases the number of direct travelers and has a weak tendency of decreasing the costs.
- The values of `max_num_cover` and `pass_edge_ratio` influence the line pool, but there is no clear correlation to the objective function values.
- Line pools with many lines do not necessarily allow better line concepts. The specific choice of lines (determined by the algorithm parameters) is more important.

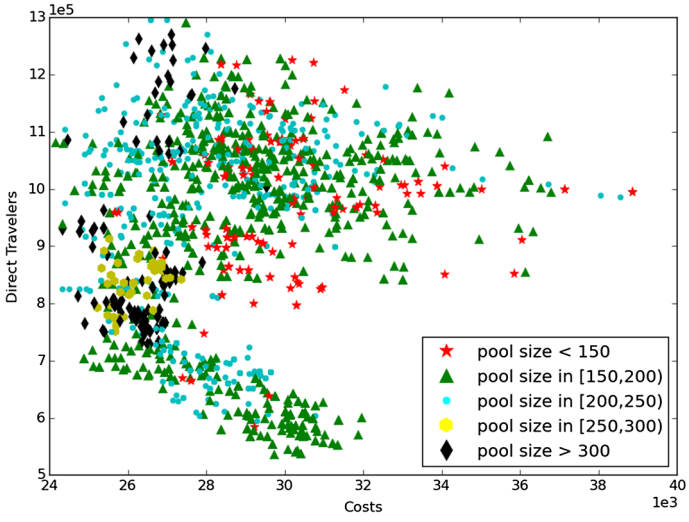


Fig. 7 Different line pools evaluated for costs and direct travelers, color coding by the size of the pools

5 Extension of the algorithm

5.1 Adding shortest paths to the line pool

In the literature, lines are sometimes constructed in a rather naive approach of just taking all pairwise shortest paths for all $u, v \in V$ and removing all lines which are contained in other lines. In our setting this returned a line pool of more than 2000 lines. Such a line pool size is far more than what is desired and what can be properly handled by OD-pair related line planning problems such as LP-Direct.

However, we allow to add shortest paths to our line pool. This is done by using Algorithm 3 after Algorithm 1 terminated.

Algorithm 3 Function for adding shortest paths as lines to the line pool

```
1: function ADD_SHORTEST_PATHS(pool)
2:     if shortest paths are to be added as lines then
3:         for u, v ∈ V do
4:             if C_{u,v} ≥ max_{u',v'} C_{u',v'} · ratio_sp then
5:                 Add shortest path from u to v to pool.
6:             end if
7:         end for
8:     end if
9: end function
```

In order to control which and how many shortest paths are chosen as lines the parameter `sp_ratio` is introduced. If the demand between two stations $u, v \in V$ is at least `sp_ratio` of the maximal demand between any two stations, the shortest path from u to v is added as a line to the pool.

The feasibility of a pool is tested after the shortest paths are added as lines in the same way as the feasibility of a pool constructed by Algorithm 1.

We also evaluated this step experimentally. The results are depicted in Fig. 8. Changing from no additional shortest paths in the line pool to `sp_ratio` $= 0.5, 0.7$ slightly increases the number of lines in the pool from 190.52 to 197.01 and 209.33, respectively. On the other hand, some benefits both in terms of the costs and direct travelers are gained. The costs are decreased from 28,863.92 to 28,329.74 and 27,789.67, respectively. The direct travelers, in turn, are increased from 897,935.13 to 902,885.91 and 917,859.82, respectively. Hence, adding some shortest paths increases the line pool and also has a small, but visible positive effect on both the costs and the number of direct travelers.

5.2 Adding random lines to the line pool

To further investigate the influence of the size of the line pool on its quality, we add randomly generated lines to one of the line pools generated by Algorithm 1. The results are shown in Fig. 9. It can be seen that we have to differentiate between the cost model and the direct travelers model. For the direct travelers model we recognize a slight improvement of 7.7 % together with a large increase in

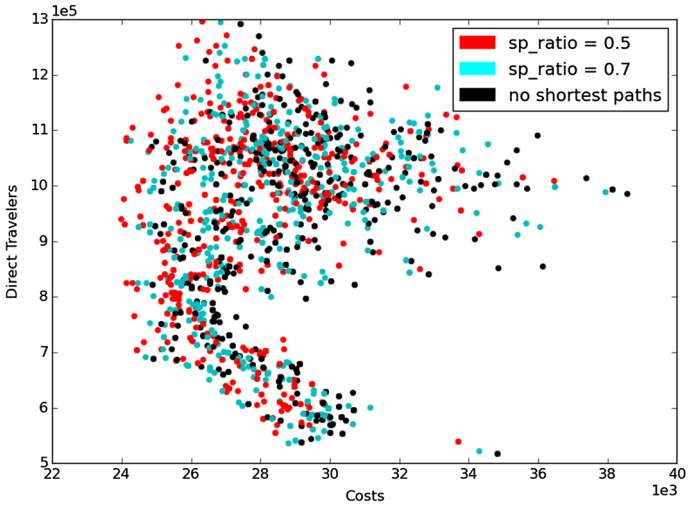


Fig. 8 Different line pools evaluated for costs and direct travelers, color coding by `sp_ratio` parameter

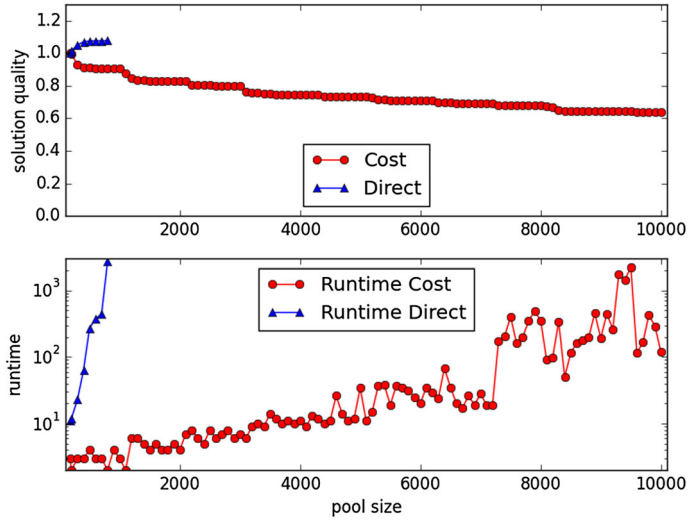


Fig. 9 Different line pools obtained by adding randomly chosen lines evaluated for costs and direct travelers

computation time. The largest pool we can solve optimally consists of 800 lines with a running time of 45 min. For larger pools, rounding errors lead to infeasible instances for the accuracy we used before.

For pools of this size, the cost model can still be solved within seconds. Thus, for the cost model we are able to consider pools of significantly larger size as input. The largest pool considered here consists of 10,000 lines with a maximal running time of

37 min, yielding an improvement of 36 %. The staircase-like behavior of the objective shows that adding more lines can decrease the costs of a line concept a lot if appropriate lines are among the added ones.

6 Outlook

In this work we study the influence of the line pool on line planning. We propose an algorithm taking care of the line pool construction in a reliable manner. In our tests, most of the generated line pools are feasible and guarantee low running times for the subsequent line planning algorithms. This holds not only for the experiments presented here, but also for tests on a network generated from the Athens Metro. For details see Gattermann (2015).

We have seen that choosing different parameters for the algorithm results in very different line pools, and we sketched the resulting correlations. More systematic and detailed numerical tests with an additional statistical analysis could give a clearer picture of the actual relations between the algorithm parameters and the evaluation parameters. We also work on line pool generation on simple graphs such as linear or star-shaped networks which will yield a better understanding of the general problem.

The problem of designing a line pool can also be seen as a line planning problem under uncertainty: At the stage of designing the pool, the network PTN is more or less known, but the cost parameters and the lower and upper frequency requirements are usually not known. While there is a need to design lines, the frequencies may be adapted in a second stage. Such a two-stage approach would mean to fix only the line pool (i.e., the x variables in the IP formulation of LPool) and deal with the frequencies after the real parameters are known. This interpretation as adjustable robust problem is currently under research.

Acknowledgments The authors are partially funded by the Simulationswissenschaftliches Zentrum Clausthal-Göttingen (SWZ) and the DFG funded Research Unit FOR2083. We also thank the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under Grant no. 246647 with the New Zealand Government (Project OptALI) for financial support.

References

Borndörfer R, Grötschel M, Pfetsch ME (2007) A column generation approach to line planning in public transport. Transp Sci 41:123–132

Bussieck MR (1998) Optimal lines in public transport. PhD thesis, Technische Universität Braunschweig

Bussieck MR, Kreuzer P, Zimmermann UT (1996) Optimal lines for railway systems. Eur J Oper Res 96(1):54–63

Claessens MT, van Dijk NM, Zwaneveld PJ (1998) Cost optimal allocation of rail passenger lines. Eur J Oper Res 110:474–489

Curtin KM, Biba S (2011) The transit route arc-node service maximization problem. Eur J Oper Res 208(1):46–56

Dienst H (1978) Linienplanung im spurgeführten Personenverkehr mit Hilfe eines heuristischen Verfahrens. PhD thesis, Technische Universität Braunschweig (**in German**)

Fan L, Mumford CL (2010) A metaheuristic approach to the urban transit routing problem. J Heuristics 16(3):353–372

Farahani RZ, Miandoabchi E, Szeto WY, Rashidi H (2013) A review of urban transportation network design problems. Eur J Oper Res 229(2):281–302

Gattermann P (2015) Generating line-pools. Master's thesis, Universität Göttingen

Goerigk M, Harbering J, Schöbel A (2015) LinTim—integrated optimization in public transportation. Homepage. see http://lintim.math.uni-goettingen.de/

Guan JF, Yang H, Wirasinghe SC (2006) Simultaneous optimization of transit line configuration and passenger line assignment. Transp Res Part B Methodol 40(10):885–902

Kepaptsoglou K, Karlaftis M (2009) Transit route network design problem: review. J Transp Eng 135(8):491–505

Kruskal JB (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. Proc Am Math Soc 7(1):48–50

Mandl C (1979) Applied network optimization

Nachtigall K, Jerosch K (2008) Simultaneous network line planning and traffic assignment. In: Fischetti M, Widmayer P (eds) ATMOS 2008—8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems. Dagstuhl, Germany. http://drops.dagstuhl.de/opus/volltexte/2008/1589

Schöbel A (2012) Line planning in public transportation: models and methods. OR Spectr 34(3):491–510

Schöbel A, Scholl S (2006) Line planning with minimal travel time. In: 5th workshop on algorithmic methods and models for optimization of railways, vol 06901

Silman LA, Barzily Z, Passy U (1974) Planning the route system for urban buses. Comput Oper Res 1(2):201–211

Zwaneveld PJ, Claessens MT, van Dijk NM (1996) A new method to determine the cost optimal allocation of passenger lines. In: Defence or Attack: Proceedings of 2nd TRAIL Phd Congress 1996, Part 2, Delft/Rotterdam. TRAIL Research School