

Exploiting Treewidth to Solve the Pricing Problem in Column Generation for Non-Pool-Based Line Planning

Markus Niebisch ✉ 

Department of Data Analytics and Digitalisation, School of Business and Economics, Maastricht University, The Netherlands

Tom C. van der Zanden ✉ 

Department of Data Analytics and Digitalisation, School of Business and Economics, Maastricht University, The Netherlands

Abstract

The line planning problem (LPP) involves determining a set of lines (routes for vehicles to travel on) for a transportation network that meet travel demands while minimizing both travel time and the line operation costs. Unfortunately, this problem is NP-hard. Recently, Heinrich et al. [17] proposed using treewidth to solve (a variant of) this problem; however, the sizes of networks and values of treewidth for which it is practical are limited.

We propose a different way of exploiting bounded treewidth in solving the LPP. We combine a column generation approach due to Borndörfer et al. [5] with dynamic programming on tree decompositions to solve the pricing problem, which entails solving the NP-hard longest path problem.

We show that this approach is feasible. We are able to apply our approach to graphs with both more vertices and higher treewidth (up to 9) than Heinrich et al. We experimentally evaluate our approach using artificial benchmarks (connected concentric ring graphs) and two practical examples, the Munich train/subway network and the Japanese railroad system.

2012 ACM Subject Classification Mathematics of computing → Network flows; Mathematics of computing → Paths and connectivity problems; Applied computing → Transportation; Theory of computation → Fixed parameter tractability; Theory of computation → Integer programming

Keywords and phrases line planning, public transport, treewidth, integer programming, fixed parameter tractability, column generation

Digital Object Identifier 10.4230/OASICS.ATMOS.2026.13

Supplementary Material *Software*: <https://doi.org/10.5281/zenodo.21797580>

1 Introduction

Public transport optimization is an important research topic in light of sustainable transport, rural exodus, and CO₂ emission goals [6, 12, 19]. One relevant aspect of public transportation optimization is the design of transportation routes and schedules. In this paper we study one such problem, which is called the line planning problem (LPP).

In the LPP it is asked to find a set of lines, i.e., routes for public transportation vehicles connecting origin-destination pairs, together with the frequency of the vehicles on those lines such that passenger transportation demands are met and the total cost is minimized, i.e., a weighted sum of passenger transportation time, fixed cost per line, and operating costs for running vehicles on lines. Unfortunately, the LPP is NP-hard [5]. One approach to solving the LPP is using column generation. We investigate the use of treewidth-based dynamic programming for solving the pricing problem, improving upon the approach using Color Coding due to Borndörfer et al. [5].



© Markus Niebisch and Tom C. van der Zanden;
licensed under Creative Commons License CC-BY 4.0

26th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2026).

Editors: Valentina Cacchiani and Stefan Funke; Article No. 13; pp. 13:1–13:20

OpenAccess Series in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The LPP is well-studied in the literature. Claessens et al. [7] solve a subset of the Dutch network with 28 stations in total to optimality. Fu et al. [13] solve the line planning problem for the full 783-station Chinese high-speed railway network of 2020, using a heuristic approach, while first determining the passenger paths and then the specific lines used, while at the same time first planning connections between major stations and after that connections with minor stations. Schöbel and Scholl [25] propose a method that can account for line transfers and evaluate it on the German rail network. For an extensive survey, see Schöbel [24].

One approach to tackling NP-hard problems is through parameterized complexity. A problem is *fixed-parameter tractable* (FPT) if it can be solved in time $f(k) n^{O(1)}$, where n is the size of the input and k is the *parameter*, see e.g., Cygan et al. [8]. Here, $f(k)$ can be an arbitrary computable function. Instances for which this parameter is small can then be solved efficiently. Both Borndörfer et al. [5] and Heinrich et al. [17] propose using FPT techniques to solve LPP.

Borndörfer et al. [5] formulate the LPP as a mixed integer linear program (MILP). Since the MILP contains variables corresponding to every possible line and every possible passenger path (of which there are exponentially many), it is infeasible to construct the full MILP. They therefore solve the linear relaxation of the MILP using column generation. They provide a greedy algorithm to round the solution of the LP relaxation into a solution of the original problem. They use column generation to generate variables corresponding to potential lines, i.e., origin-destination pairs and paths between them, and variables corresponding to passenger paths. The pricing problem for the lines is the weighted longest path problem. It is well known that this problem is strongly NP-hard [26].

To solve the pricing problem for the lines, Borndörfer et al. [5] employ a randomized dynamic programming algorithm, *color coding*, to find the longest weighted path. Color coding uses as a parameter k the maximum length of the path. They argue that this parameter is usually bounded in real-world transportation networks, which might be planar (in which case they argue “typical” lines have length $O(\sqrt{|V|})$ for a network with $|V|$ nodes), or might even be trees or tree-like, in which case one would get a bound of $O(\log |V|)$ on the maximum length of a path if the graph is preprocessed to remove nodes of degree 2. Therefore, they argue that color coding, an FPT algorithm parameterized by path length, should be a suitable method for solving the pricing problem.

The tree-likeness of networks can be exploited more directly by using the notion of treewidth. An obstacle to using treewidth in practice was the difficulty of finding tree decompositions. PACE 2016 and 2017 [9, 10] called for the development of practical treewidth solvers. Tamaki’s work (see e.g., [27, 28]) has been particularly influential in this area and has resulted in the availability of excellent solvers.

The first use of treewidth in solving the LPP is due to Heinrich et al. [17]. They observe that the treewidth of graphs underlying transportation networks is often small. Instead of using column generation, their algorithm, TW-LINES, constructs an integer linear program (ILP) that exploits the bounded treewidth of the network, allowing it to be solved in FPT time. However, the dependence on the parameter treewidth is bad, and they can only solve instances with a relatively small treewidth of at most 5. Moreover, they consider a more restricted version of the problem named “Line planning on all paths” (LPAL), in which the passenger paths are fixed in advance, and the only question is the routing and the integer frequency of the lines. Another simplification is their exclusion of a frequency-independent cost. This simplification is necessary since, otherwise, the problem is NP-hard even on paths and stars [16], thus excluding (unless $P=NP$) also the possibility of an FPT algorithm parameterized by treewidth. However, they can solve the ILP to optimality, whereas column generation only solves the LP relaxation to optimality and may yield a suboptimal solution after rounding.

We propose an alternative way to exploit treewidth to solve the LPP. We combine the column generation approach of Borndörfer et al. [5] with dynamic programming on tree decompositions to find the longest weighted path. Unlike the color coding approach used by Borndörfer et al. [5] to solve the (line) pricing problem, our approach can find lines of arbitrary length, which usually allows us to use fewer lines. This also allows us to generate columns for larger graphs without applying aggressive preprocessing. Borndörfer et al. [5] preprocess the network by iteratively removing leaves as well as degree-2 vertices, which is necessary to keep the length of the paths required reasonable. However, this simplification can make the model unrealistic. In the case of the Munich S-Bahn and subway network, this preprocessing would collapse the entire graph into only a few core vertices, making it difficult to apply the solution to the original graph.

To convert a solution to the LP relaxation to a solution for the original MILP, it is necessary to round the solution. Since the LP relaxation may contain many low-frequency lines, naively rounding it can result in a bad solution to the original MILP. We consider two complementary approaches: For small enough networks, it may be feasible to solve the original MILP restricted to the line set obtained by column generation for the LP relaxation. For larger networks, it may be necessary to use heuristics, as Borndörfer et al. [5] do.

We compare two different heuristic methods for rounding a solution of the LP relaxation to an MILP solution. Borndörfer et al. [5] used a method which we refer to as “Frequency-Based Greedy Rounding” (FB). We suggest an alternative method, “Generation-Order Greedy Rounding” (GO), which typically yields better results while reducing runtime by about 90%.

We similarly compare two methods to round the LP relaxation by solving an MILP. Following similar logic, the columns which are used in the final solution are usually generated early on during the column generation. Therefore, we compare an “early-stopping” variant of MILP rounding (that uses the same restricted line set as GO rounding) to MILP rounding using the full line set from the LP relaxation.

We experimentally evaluate our algorithm by solving the line-planning problem for concentric ring graphs, the Munich S-Bahn and subway network, and the Japanese national network, using both our method and color coding. We compare runtime, costs, and rounding methods. Furthermore, we adapt our method to the restricted problem of Heinrich et al. [17]. As Heinrich’s method solves LPAL to optimality, our result can only be worse. The main purpose of the comparison is to get insights into the quality of our algorithm with instances that TW-LINES can solve. This allows us to gauge the quality of our algorithm on larger instances that TW-LINES cannot solve.

Heinrich et al.’s [16, 17, 1] ILP-based approach, TW-LINES, is able to (within a time limit of 1 hour) solve instances with up to 90 vertices and treewidth 4, and graphs with up to 40 vertices and treewidth 5 (Intel Xeon Gold 6126 CPU). In some of their experiments, they stop the LP solver when it gets to within 2% of the optimal solution. In this setting, they are able to solve instances with up to 770 vertices and treewidth 3, up to 170 vertices and treewidth 4, and up to 41 vertices on graphs of treewidth 5. Compared to TW-LINES, our approach (with a more restrictive time limit of 30 minutes) can handle instances with up to ~ 700 vertices and treewidth 7, and instances with up to 82 vertices and treewidth 9 (Intel Core i7-11800H). Of course, our results are for solving an LP relaxation (albeit of a more complex problem) and cannot be directly compared to Heinrich et al.’s results, but they extend the set of networks for which *some* advantage of low treewidth can be taken.

This paper is organized as follows: we first discuss the Line Planning problem and Borndörfer et al.’s [5] column generation approach for it. We then discuss both the color coding and treewidth-based approaches for solving the longest path pricing problem. We

then discuss various approaches to rounding the LP relaxation to an integer solution. In our experiments section, we first compare against Heinrich et al.'s TW-LINES [17] (as a benchmark for another treewidth-based method) and against Borndörfer et al.'s [5] original method using color coding. Both comparisons are on artificial networks. We then investigate the performance of the different rounding methods. Finally, we present an evaluation of our method on two real-life networks: the train network of Japan and the public transport network of Munich.

2 Preliminaries

2.1 The Line Planning Problem

In this section, we introduce the variant of the line planning problem (LPP) that we consider, which is largely equivalent to that of Borndörfer et al. [5]. For simplicity, we do not consider multiple modes of transportation, though this can easily be added back into the model.

A public transportation network (PTN) is a graph $G = (V, E)$, where vertices represent stations and edges represent possible direct connections. Let $n := |V|$ denote the number of stations. In addition, we are given a demand matrix $D = (d_{s,t}) \in \mathbb{R}^{n \times n}$, where $d_{s,t}$ gives the number of travelers wanting to travel from station s to station t . Our objective is to find a set of lines (specifying routes and frequencies with which vehicles run on them) together with routes for passengers to travel (using the transportation capacity of the lines) so that the travel demand is met, while minimizing a weighted combination of operational cost and passenger travel time.

Formally, a line ℓ is a simple path $\ell = (v_1, \dots, v_k)$, where $v_i \in V$. The lines are undirected because we assume each transportation line will have vehicles running in both directions at the same frequency. We denote by κ_ℓ the maximum number of passengers that can be transported by a vehicle of a line in a single run. Each edge has a maximum frequency of vehicles that can run on it, which we denote by $f_e^{\max}$. We denote by F the maximum of the maximum edge frequencies of the entire network, i.e., $F = \max_{e \in E} f_e^{\max}$. We denote by $\mathcal{L}$ the set of all lines (i.e., the set of all simple paths in G).

In contrast, the passenger flows may be asymmetric. We define the passenger travel graph as $G_P = (V, A)$, with two antiparallel arcs $a_{s,t}$ and $a_{t,s}$ for every edge $e_{s,t}$ in E . For simplicity, we denote the two arcs of an edge e by $a(e)$ and $\bar{a}(e)$. The associated edge e (in the PTN) for arc a is represented by $e(a)$.

A travel path from v_s to v_t is a simple path $(v_s, \dots, v_t)$ using arcs in A . Each arc has a travel time τ_a , which leads to the travel time of a path p being $\tau_p = \sum_{a \in p} \tau_a$. The set of all travel paths connecting s and t is denoted as $\mathcal{P}_{s,t}$, and the set of all travel paths as $\mathcal{P}$.

The fixed operating cost of a line is c_ℓ^{fix} , and the variable operating cost of a line is the sum of the variable costs of its edges $c_\ell^{\text{var}} = \sum_{e \in \ell} c_e^{\text{var}}$.

Finally, to balance the total line operating cost with the travel cost, we multiply c_ℓ^{fix} and c_ℓ^{var} by $\lambda \in [0, 1]$. Similarly, we use $(1 - \lambda) \times \tau_a$ to adjust the travel cost. For notational convenience, we omit the scaling factor from our expressions and assume it has already been incorporated into the respective variables.

In the mixed-integer linear programming (MILP) formulation, we define a continuous variable y_p for each travel path $p \in \mathcal{P}$, giving the number of travelers using that path.

We define a binary variable x_ℓ , giving for each line ℓ whether that line is used in the solution, and a continuous variable f_ℓ , giving the frequency of vehicle runs on that line.

The total number of passengers using each arc for travel must be able to be transported by the lines that run along that arc. To this end, we define the line frequency of a single edge $f_e = \sum_{\ell \in \mathcal{L}(e)} f_\ell$, where the set of lines which share a single edge $e \in E$ is denoted as $\mathcal{L}(e) = \{\ell \in \mathcal{L} : e \in \ell\}$. Analogously, the passenger flow of a single arc is $y_a = \sum_{p \in \mathcal{P}(a)} y_p$, where $\mathcal{P}(a) = \{p \in \mathcal{P} : a \in p\}$ denotes the paths that use arc a .

Finally, to later be able to guarantee feasibility during the initial phases of column generation, we introduce continuous artificial variables $u_{s,t}$, denoting the unfulfilled travel demand between s, t , and introduce a big- M penalty M for unfulfilled demand. This then leads to the following MILP formulation:

$$\begin{aligned}
 (\text{MILP}) \quad & \min \sum_{p \in \mathcal{P}} \tau_p y_p + \sum_{\ell \in \mathcal{L}} c_\ell^{\text{fix}} x_\ell + \sum_{\ell \in \mathcal{L}} c_\ell^{\text{var}} f_\ell + \sum_{s,t \in D} M u_{s,t} \\
 \text{s.t.} \quad & \sum_{p \in \mathcal{P}_{s,t}} y_p + u_{s,t} = d_{s,t} & \forall (s,t) \in D & \quad (i) \\
 & y_a - \sum_{\ell \in \mathcal{L}(e(a))} \kappa_\ell f_\ell \leq 0 & \forall a \in A & \quad (ii) \\
 & f_e \leq f_e^{\max} & \forall e \in E & \quad (iii) \\
 & f_\ell \leq F x_\ell & \forall \ell \in \mathcal{L} & \quad (iv) \\
 & x_\ell \in \{0, 1\} & \forall \ell \in \mathcal{L} & \quad (v) \\
 & f_\ell \geq 0 & \forall \ell \in \mathcal{L} & \quad (vi) \\
 & y_p \geq 0 & \forall p \in \mathcal{P} & \quad (vii) \\
 & u_{s,t} \geq 0 & \forall (s,t) \in D & \quad (viii)
 \end{aligned}$$

The objective function contains four elements:

1. $\sum_{p \in \mathcal{P}} \tau_p y_p$: the total travel time across all passenger paths,
2. $\sum_{\ell \in \mathcal{L}} c_\ell^{\text{fix}} x_\ell$: the total fixed costs of all active lines,
3. $\sum_{\ell \in \mathcal{L}} c_\ell^{\text{var}} f_\ell$: the variable costs times the line frequencies,
4. $\sum_{s,t \in D} M u_{s,t}$: finally, a big- M penalty for unfulfilled travel demands.

The constraints are:

- (i) assigned passenger flow plus unfulfilled demand equals travel demand,
- (ii) the available capacity on each arc is at least the total travel demand y_a on that arc,
- (iii) the frequency of vehicles running on an edge does not exceed its maximum frequency,
- (iv,v) decide whether a line is part of the line plan or not,
- (vi, vii, viii) non-negativity of passenger flow, line frequencies, and unfulfilled travel demands.

2.2 Column generation for Line Planning

We use column generation to solve the LP relaxation of the MILP formulation introduced in the previous section. Borndörfer et al. [5] observe that the LP relaxation can be simplified because, in the LP relaxation, constraint (iv) will always be fulfilled with equality. We can therefore remove the x_ℓ variables and incorporate the fixed costs of the lines into the variable costs by setting $c_\ell^{\text{total}} = c_\ell^{\text{fix}}/F + c_\ell^{\text{var}}$. This leads to the following LP relaxation:

$$\begin{aligned}
 (\text{LP}) \quad & \min \sum_{p \in \mathcal{P}} \tau_p y_p + \sum_{\ell \in \mathcal{L}} c_\ell^{\text{total}} f_\ell + \sum_{s,t \in D} M u_{s,t} \\
 \text{s.t.} \quad & \sum_{p \in \mathcal{P}_{s,t}} y_p + u_{s,t} = d_{s,t} & \forall (s,t) \in D & (i) \\
 & y_a - \sum_{\ell: e(a) \in \ell} \kappa_\ell f_\ell \leq 0 & \forall a \in A & (ii) \\
 & f_e \leq f_e^{\max} & \forall e \in E & (iii) \\
 & f_\ell \geq 0 & \forall \ell \in \mathcal{L} & (vi) \\
 & y_p \geq 0 & \forall p \in \mathcal{P} & (vii) \\
 & u_{s,t} \geq 0 & \forall (s,t) \in D & (viii)
 \end{aligned}$$

To solve this LP, we use column generation to generate columns corresponding to the lines $\mathcal{L}$ and passenger paths $\mathcal{P}$. For an introduction to column generation, we refer to [11].

The dual of the above LP is:

$$\begin{aligned}
 (\text{dual}) \quad & \max \sum_{s,t \in D} d_{s,t} \pi_{s,t} - \sum_{e \in E} f_e^{\max} \eta_e \\
 \text{s.t.} \quad & \pi_{s,t} - \mu(p) \leq \tau_p & \forall p \in \mathcal{P}_{s,t}, (s,t) \in D & (a) \\
 & \kappa_\ell \mu(\ell) - \eta(\ell) \leq c_\ell^{\text{total}} & \forall \ell \in \mathcal{L} & (b) \\
 & \pi_{s,t} \leq M & \forall (s,t) \in D & (c) \\
 & \mu_a \geq 0 & \forall a \in A & (d) \\
 & \eta_e \geq 0 & \forall e \in E & (e)
 \end{aligned}$$

where

$$\mu(\ell) = \sum_{e \in \ell} (\mu_{a(e)} + \mu_{\bar{a}(e)})$$

For the derivation of the dual formulation, we refer to [5]. The only difference between our LP and that of [5] is that we use a big- M formulation to deal with unfulfilled demand, which leads to an additional constraint $\pi_{s,t} \leq M$ in the dual.

Again, following [5], the reduced cost of the variable relating to a passenger path p is

$$\bar{\tau}_p = \tau_p - \pi_{s,t} + \mu(p) = \tau_p - \pi_{s,t} + \sum_{a \in p} \mu_a = -\pi_{s,t} + \sum_{a \in p} (\mu_a + \tau_a)$$

Thus, to solve the pricing problem, we need to solve a weighted shortest path problem in the passenger travel graph, with arc weights $\mu_a + \tau_a$. This can easily be done using, e.g., Dijkstra's algorithm, since all weights are non-negative.

For the lines, the reduced cost is

$$\begin{aligned}
 \bar{\tau}_\ell &= c_\ell^{\text{total}} - \kappa_\ell \mu(\ell) + \eta(\ell) \\
 &= c_\ell^{\text{fix}}/F + c_\ell^{\text{var}} - \kappa_\ell \mu(\ell) + \eta(\ell) \\
 &= c_\ell^{\text{fix}}/F + \sum_{e \in \ell} (c_e^{\text{var}} - \kappa_\ell (\mu_{a(e)} + \mu_{\bar{a}(e)}) + \eta(e))
 \end{aligned}$$

Thus, to find a line with a negative reduced cost, we need to find a line ℓ with:

$$\sum_{e \in \ell} (\kappa_\ell(\mu_{a(e)} + \mu_{\bar{a}(e)}) - \eta(e) - c_e^{\text{var}}) > c_\ell^{\text{fix}}/F$$

As $\kappa_\ell(\mu_{a(e)} + \mu_{\bar{a}(e)}) - \eta(e) - c_e^{\text{var}}$ is not restricted in sign, solving the pricing problem for the lines is equivalent to solving a maximum weighted path problem, with edge weights $\kappa_\ell(\mu_{a(e)} + \mu_{\bar{a}(e)}) - \eta(e) - c_e^{\text{var}}$.

2.3 Color Coding

Borndörfer et al. [5] solve the longest weighted path problem with a randomized algorithm, color coding. In every iteration, a random color is assigned to every vertex, and, using dynamic programming, the longest weighted path without a color repeating is found for a given start vertex. By repeating this process with different random colorings, we can boost the probability of finding the longest path. A single iteration has a complexity of $m \times 2^{O(k)}$, where m is the number of edges and k is the number of colors. This algorithm would need $a \times e^k$ iterations to achieve a success rate of $1 - e^{-a}$. For example, this would mean that for $a = 2$ and $k = 10$, we would require $\sim 44,000$ iterations to have a success rate of $\sim 86\%$. Finally, this process must then be repeated for every vertex as a starting point.

This algorithm has several downsides; for one, we cannot find a path longer than the number of colors k . Furthermore, if a larger number of colors is selected, the runtime increases exponentially, as we need an exponentially larger number of iterations to achieve the same success rate, and the runtime of a single iteration also grows exponentially. Borndörfer et al. [5] use only 15 iterations in their implementation, suppress stations with degrees 1 or 2, and use a subset of V as terminal vertices for lines to make this approach feasible. In our experiments, we also increase the number of colors rather than the number of iterations, as suggested by [18].

In comparison, our treewidth-based dynamic programming can find paths of arbitrary length, provided the graph has bounded treewidth. Furthermore, we would generally not need to remove a node of degree 1 or 2, since subdividing edges does not increase treewidth; therefore, we can consider the original network.

3 Our approach

The main algorithm first constructs the initial linear-programming (LP) relaxation, as described in Section 2.2. Afterward, it conducts column generation, i.e., alternates between solving the pricing problem for the transport paths and for the transport lines, and modifies the LP using the edge weights as described in the same section. To solve the former, we use Dijkstra's algorithm; for the latter, we use dynamic programming on tree decompositions, as described in Section 3.1. This process of adding paths and lines is repeated until no improvement can be found after adding a path. After that, the solution is rounded to an MILP solution, and low-frequency lines are removed, as described in Section 3.2.

3.1 Treewidth-based dynamic programming for Longest Path

In this section, we briefly describe the algorithm for solving the weighted longest path problem by dynamic programming on tree decompositions and recall the basic definitions. For a thorough introduction to treewidth and dynamic programming, we refer to [8].

A *tree decomposition* of a graph G is a tuple $(T, \mathcal{B})$ where T is a tree and $\mathcal{B} = \{b_i \mid i \in V(T)\}$ is a set of bags. Each bag $b_i \in \mathcal{B}$ is a subset of V . Additionally, the following properties must be satisfied:

1. $\bigcup_{b \in \mathcal{B}} b = V$, i.e., every vertex is contained in some bag,
2. For all $(u, v) \in E$, there exists $b \in \mathcal{B}$ such that $\{u, v\} \subseteq b$, i.e., for each edge in E , there is a bag which contains both its endpoints,
3. For all vertices $u \in V$, the bags $b \in \mathcal{B}$ containing vertex u must form a connected subgraph.

The *width* of a tree decomposition is $\max_{b \in \mathcal{B}} |b| - 1$, and the *treewidth* of a graph is the minimum width of a tree decomposition of it.

An important property of tree decompositions is that the bags are separators of G (see e.g., [8, Lemma 7.3]). This property is what enables dynamic programming. If we view the tree decomposition as a rooted tree, then every bag $b_t \in \mathcal{B}$ separates the vertices in the subgraph of G induced by the vertices contained in (the bags of) the subtree of T rooted at t from the rest of G . We denote this subgraph by $G[t]$. For clarity, $G[t]$ can be viewed as b_t plus a union of connected components of $G \setminus b_t$; which components exactly depends on the choice of root.

We define a *partial solution* for $t \in V(T)$ as a union of vertex-disjoint (including endpoints) paths of $G[t]$, such that at most two endpoints of the union of paths are not contained in b_t and every path has at least one endpoint in $G[t]$, unless the partial solution consists of a single path.

Note that if p is a path in G , then the intersection of p with $G[t]$ is a partial solution for t . Thus, we can solve the longest path problem by enumerating all partial solutions. Dynamic programming on tree decompositions speeds up this enumeration by taking advantage of the fact that some partial solutions are in some sense “equivalent”, and we only need to keep a partial solution of maximum weight per equivalence class.

Two partial solutions for $t \in V(T)$ are equivalent if (1) all vertices in b_t have the same degree in the partial solution (which can be either 1 or 2, or, if the vertex is not contained in the partial solution, 0), (2) both partial solutions connect the same degree-1 vertices of b_t and (3) the number of path endpoints not contained in b_t is equal. For a graph of treewidth tw , the *characteristic* of a partial solution can thus be defined as a $(2|b_t| + 1)$ -tuple, with the first $|b_t|$ elements giving the degree of each vertex of b_t in relation to the partial solution, the second $|b_t|$ giving which vertex each degree-1 vertex is connected with (if any) and the last value giving the number of endpoints not contained in b_t .

Our algorithm is standard dynamic programming on tree decompositions: we process the tree in a bottom-up fashion, and for each node $t \in V(T)$ compute – using the values previously computed for the node’s children – a list of partial solutions, grouped by characteristic, storing only the maximum weight partial solution for each. To simplify our algorithm, we preprocess the tree decomposition into a *nice tree decomposition* [4]. We also implemented the rank-based approach [3] to further reduce the number of equivalence classes by recognizing that certain connectivity partitions of degree-1 vertices are equivalent.

This is very similar to the algorithm for Hamiltonian Cycle evaluated in [29] (to which we refer for more details), though modified slightly since we do not need to visit all vertices or connect the endpoints of a path into a cycle.

3.2 Rounding

As a final step, the LP solution needs to be rounded to a solution for the MILP. Since the binary decision variables x_ℓ for all $\ell \in \mathcal{L}$ were removed during relaxation, they must now be set to obtain a feasible solution to the MILP. Hereby, the solution frequencies for the lines f_ℓ and paths y_p of the LP are substituted into the objective function for the MILP. If $f_\ell > 0$ then x_ℓ is set to 1, otherwise x_ℓ is set to 0. As the cost per line changes from $\sum_{\ell \in \mathcal{L}} c_\ell^{\text{total}} f_\ell$ to $\sum_{\ell \in \mathcal{L}} c_\ell^{\text{fix}} x_\ell + \sum_{\ell \in \mathcal{L}} c_\ell^{\text{var}} f_\ell$, it is likely that the solution can be improved by omitting some low-frequency lines.

We use two approaches for heuristic rounding. First, we follow the greedy process of Borndörfer et al. [5], which starts with the full line pool and removes lines one by one to obtain a new solution. This is achieved by setting the capacity coefficient κ_ℓ of the line with the lowest non-zero frequency to 0, and solving the pricing problem for the passenger paths again. This is repeated as long as the MILP cost is not increasing. The last result is then the final solution. We will refer to this rounding method as “Frequency-Based Greedy Rounding” (FB).

The number of lines generated when solving the LP relaxation is usually far larger than the number of lines that end up in the final solution (generated by FB rounding). However, a lot of time is consumed by re-solving the LP each time a line is removed during FB rounding. We observed that, in practice, the lines which end up being used in the solutions are usually generated fairly early on during the column generation process.

This observation leads to an alternative approach. Here, we start with an empty line pool, i.e., set the capacity κ_ℓ of all lines ℓ to zero, and add them one by one in the order they were created during the column generation, i.e., set κ_ℓ to the original value, as long as the respective MILP cost is decreasing. We will refer to this rounding method as “Generation-Order Greedy Rounding” (GO). This method has the advantage that no additional runtime is needed when executed in parallel to the regular column generation.

Finally, we compare the results of the two heuristic rounding methods with those obtained by solving the MILP restricted to the line pool $\mathcal{L}$ produced by column generation for the LP relaxation. We distinguish between two different line pools: “Converged MILP” (CO MILP) uses all lines of the optimal solution to the LP relaxation. The second method, “Early Stop MILP” (ES MILP) uses a line pool containing only those lines that would have been used by “Generation-Order Greedy Rounding” (GO). For the biggest instances, running the column generation to completion is not possible within a reasonable amount of time. Similarly to the situation with GO vs. FB rounding, lines generated late are usually not used in the solution. Therefore, ES MILP rounding allows us to save significant computation time without a large decrease in solution quality.

4 Experiments

We run experiments on two types of graphs, concentric ring graphs and real-world examples, and investigate three different topics in our experiments. First, we compare the runtime and costs of our approach, which we refer to as TREEWIDTH COLUMN GENERATION (TWCG), against TW-LINES by Heinrich et al. [17, 1]. Second, we compare the runtime and solution quality of TWCG against COLOR CODING (CC) (as used by Borndörfer et al. [5]) for the pricing problem in line planning. Finally, we compare the quality of the four different rounding methods.

4.1 Experiment parameters

If not stated otherwise, we use the following parameter values: fixed costs per line $c_\ell^{\text{fix}} = 10,000$, operational costs for an edge $c_e^{\text{var}} = 100$, transport time $\tau_a = 1$, penalty value for unfulfilled transport demand $M = 10^6$, travel demand between $d_{s,t} = 50$ for all vertex pairs (s, t) . We use a time limit of 30 minutes wall time, after which rounding is started and we allow the MILP solver to run for an additional 15 minutes. The time limit was only reached for instances with treewidth 9 or instances with very many vertices. Since the final line used is typically found within the first few minutes, the time limit primarily affects how many ultimately disused columns were generated, without affecting the final solution quality.

The transport capacity κ_ℓ and the maximum edge frequency F are instance-dependent and are created similarly to [17] to help comparability: We set $\kappa_\ell = (|V| - 1)^2$ for all lines ℓ . To set F , we compute first the shortest paths between each pair of vertices s, t . We then count the number of passengers who would traverse each edge, assuming each passenger travels from their origin to their destination over the shortest path. The maximum edge frequency F is then the maximum resulting value divided by κ_ℓ , i.e., let $sp_{s,t}$ be the shortest path between v_s and v_t , and let $d_a = \sum_{\{(s,t): a \in sp_{s,t}\}} d_{s,t}$, then $F = \max_{a \in A, \ell: e(a) \in \ell} \frac{d_a}{\kappa_\ell}$. The line cost estimation in the LP relaxation will be closer to the MILP cost if the frequency of a line f_ℓ is close to the maximum frequency F , as in the LP relaxation, the line cost is estimated as $c_\ell^{\text{total}} = c_\ell^{\text{fix}}/F + c_\ell^{\text{var}}$. In practice, to solve the LPP using a relaxation-based approach, one would have to come up with reasonable estimates of the maximum frequency per line so that the integrality gap does not become too large. Our choice of F guarantees a demand-fulfilling solution while keeping the gap reasonably small.

To determine the value of λ for different instances, we let the experiment run twice: Once with $\lambda = 0.5$, after which we calculate the λ value that would equalize travel and operating costs of the LP solution and we run the experiment with that λ .

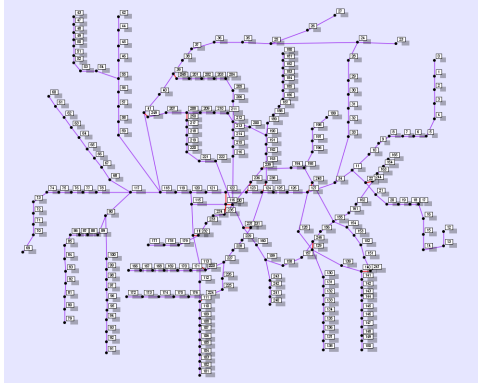
All experiments were conducted using an Intel Core i7-11800H processor at 2.30 GHz, 12 GiB RAM running Windows 11. Our implementation was written in Java and executed using OpenJDK Runtime Environment 22.0.1 [21]. We used OR-Tools version 9.8.3296 [22] together with Gurobi version 10.0.2 [15]. Although Gurobi 12.0.1 was available, OR-Tools did not yet support it when the experiments were conducted. For comparison, we used the authors' original C++ implementation of Heinrich's algorithm TW-LINES [1], which we ran with Gurobi 12.0.1. In order to speed up our experiments, we parallelized the COLOR CODING algorithm over the possible different starting points. Furthermore, we also parallelized the join operator of TWCG over the entries of one of the tables being joined. Every tree decomposition was created using Jdrasil 0.4dev [2], which is the same solver Heinrich et al. [17] use. We do not report the time of computing a tree decomposition separately, as even for the largest instances this only takes a few seconds. For this reason, we did not explore other solvers.

4.2 Instances

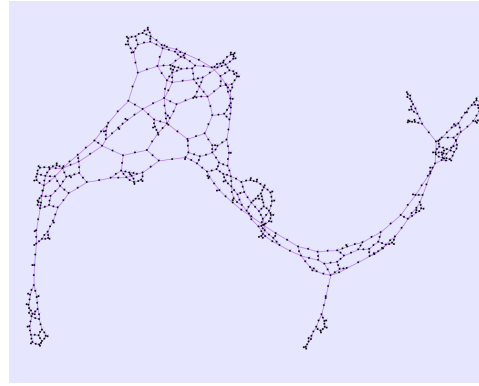
Next, we introduce the two types of graphs on which we conduct our experiments: concentric ring graphs and two real-world examples.

The first type of graph in our experiments is concentric ring graphs, as used in the experiments of Heinrich et al. [17]. The concentric ring graph consists of r concentric cycle graphs C_s , each with s vertices, and one extra "hub" vertex. All vertices of the first cycle graph are connected to the central hub and form a wheel graph with s spokes. Each vertex in each cycle graph is then connected to the corresponding vertex in the previous and next cycle graph. In total, we solve 32 instances with between 3 and 9 spokes and treewidth ranging from 3 to 9.

We also experiment on two graphs modeling real-world examples, namely the Munich S-Bahn and subway network and the Japanese national rail network.



■ **Figure 1** Munich S-Bahn and subway network. $|V| = 251$, $|E| = 249$, $tw = 4$. Transfer edges in red.



■ **Figure 2** Train network of Japan. $|V| = 694$, $|E| = 830$, $tw = 7$.

The Munich network consists of 251 vertices, 249 regular edges, 14 transfer edges, and has treewidth 4, as shown in Figure 1. We obtained the network by tracing publicly available maps [20]. As it utilizes two transportation modes, we employ a disconnected graph for line planning and additionally use 14 special transfer edges for path planning, an approach also used by [5]. These edges have infinite capacity and zero cost. We chose Munich as a medium-sized case to show that several transportation modes are possible. Furthermore, the Munich graph cannot be handled by the original simplification step proposed by Borndörfer et al. [5], which iteratively removes degree-2 vertices and leaves. In the case of the Munich S-Bahn without the subway, the resulting graph would no longer be a realistic model of the network, as the simplification would remove substantial portions of the network.

The final case is the Japanese rail network, which consists of 995 vertices and 1,131 edges [23]. We are using $d_{s,t} \in \{1, 2, 3, 4, 5\}$ chosen uniformly at random for all $(s, t) \in V \times V$. To reduce computation time, we contract the common edge between neighboring vertices of degree 2 and adjust the demand matrix accordingly (this is a slightly less aggressive preprocessing than the one used by Borndörfer et al. [5]). The contracted graph (see Figure 2) has 694 vertices and 830 edges, and treewidth 7.

For the Japanese network, the bottleneck is not solving the pricing problem but rather solving the LP relaxation. Using the full demand matrix results in a large number of constraints, since its size is quadratic in the number of vertices. Therefore, we reduce the size of the demand matrix and zero out all the columns except for 30. This could be interpreted as a rush hour where passengers commute into several key cities from all over the country, which may be a more realistic scenario than one in which we have equal demand between all pairs of stations. Additionally, we set $\kappa_\ell = 1,320$, the approximate transport capacity of a Shinkansen train, and $F = 18$. This happens to equal the actual maximum frequency of the Shinkansen, but the fact that this yields a feasible solution with our demand matrix (which does not include realistic demands) is a coincidence. To account for the larger instance size, we increase the running time before rounding starts to 60 minutes and use “Early Stop MILP”, as explained in Section 3.2, as the rounding method.

4.3 Comparison against TW-LINES

We compare TWCG against Heinrich et al.’s TW-LINES [17] treewidth-based algorithm on concentric ring graphs. The problem (LPAL) solved by TW-LINES is substantially different from the LPP problem: It does not concern itself with passenger paths (and instead, takes as input a travel demand per edge) and it does not incorporate a per-line fixed cost, which makes it a much simpler problem than LPP, but it does constrain line frequencies to be integer rather than continuous (which makes this aspect harder than LPP). This makes it difficult to compare TW-LINES and our TWCG, but since TW-LINES is the only other approach to a line planning problem using treewidth, we decided to include this benchmark.

Due to these differences, we have to adjust our LP formulation. Since passenger paths are no longer part of the problem, we add, for each OD-pair, the shortest path to the LP as a column and prohibit adding additional paths. Furthermore, we set $\lambda = 1$ (to take travel costs out of the equation). Instead of a per-line fixed cost, LPAL uses a frequency-dependent fixed cost. Since in the LP relaxation of LPP, the per-line fixed cost turns into a frequency-dependent cost, we can make these parameters equal.

After the LP relaxation is solved and no further line columns can be added, we solve an ILP version of the LP, i.e., all line variables are set to integer instead of continuous. Note that LPAL also features a maximum frequency per line, which would make solving this ILP infeasible, but our F is large enough to avoid this scenario.

The results of these experiments are shown in Table 1. As expected, their solution (as they solve to optimality instead of dealing with an ILP relaxation), is better in almost every case – with the exception of some cases (the two cases with treewidth 5 and 8 or 10 rings) where their ILP solver runs out of time before finding an optimal solution.

In the worst case, our solution was 23.6% more expensive for 5 spokes and 5 rings, while in the best case, we achieved 12.79% lower cost for 5 spokes and 8 rings. On average, our solutions were 8.3% worse, with a median of 8.0%.

When comparing runtimes, we observe that our algorithm runs faster on every ring graph instance except the smallest. The largest instances took TWCG 40 seconds, with many being below 10 seconds. For example, TWCG solves 3 spokes and 20 rings in 35.1 seconds, 4 spokes and 7 rings in 4.3 seconds, and 5 spokes and 5 rings in 2.6 seconds, while TW-LINES uses the full 30 minutes for each of these instances and finishes with IP gaps of 2%, 0.5%, and 0.6%, respectively.

The most complex instances where TW-LINES managed to provide a solution were for 5 spokes and 8 rings, and for 5 spokes and 10 rings. However, those had larger IP gaps of 12.8% and 16.6%, respectively. TW-LINES did not manage to solve the smallest treewidth 6 instance with 5 rings. In contrast, our approach can handle (within the 30-minute time limit) instances with treewidth up to 9.

Heinrich et al. [17] report average improvements of 36% over heuristic line pool generation in [14]. Considering that our solutions were only on average 8.3% worse than Heinrich et al.’s, TWCG appears to perform decently as a heuristic for LPAL (a problem it was not initially designed for).

4.4 Comparison against COLOR CODING on ring graphs

Next, we compare the solution quality of TWCG and COLOR CODING on concentric ring graphs. We solve instances with treewidth up to 9, up to 9 rings, and spokes ranging from 3 to 9. For COLOR CODING, we consider two parameter settings and report the one that yields better results for each instance: $k = 20$ with 50 iterations and $k = 25$ with 20 iterations.

■ **Table 1** Comparing TW-LINES by Heinrich et al. and TREewidth COLUMN GENERATION (TWCG) on ring graphs. Left: description of the instances where s , r , tw , and n denote the number of spokes, rings, treewidth, and number of stations. Middle: the TWCG solution – number of lines, total cost, average frequency, average line length, and runtime. Right: the same metrics for the TW-LINES solution, and the IP gap.

Input				TWCG					TW-LINES					
s	r	tw	n	no. l	cost	avg. freq	avg. len.	time (s)	no. l	cost	avg. freq	avg. len.	time (s)	IP gap
3	5	3	16	7	17,560	1	12.286	0.849	7	17,060	1	11.571	0.382	0
3	10	3	31	6	23,180	1	25.833	2.818	6	21,680	1	23.333	58.872	0
3	15	3	46	6	29,980	1	37.167	8.680	6	28,280	1	34.333	964.664	0
3	20	3	61	6	37,780	1	50.167	35.109	6	35,180	1	45.833	1,803.808	0.020
4	5	4	21	7	19,760	1	15.429	1.230	6	17,280	1	16	44.413	0
4	7	4	29	6	21,680	1	23.333	4.333	6	19,380	1	19.500	1,801.615	0.005
4	10	4	41	6	27,380	1	32.833	7.619	6	24,280	1	27.667	1,802.223	0.043
4	12	4	49	6	30,180	1	37.500	18.093	5	25,300	1	37.800	1,803.261	0.040
5	3	5	16	8	18,140	1	9.875	0.483	8	17,240	1	8.750	19.606	0
5	4	5	21	8	20,340	1	12.625	1.598	8	18,740	1	10.625	133.323	0
5	5	5	26	6	19,780	1	20.167	2.593	5	16,000	1	19.200	1,816.643	0.006
5	6	5	31	6	21,780	1	23.500	4.742	5	18,000	1	23.200	1,820.557	0.006
5	8	5	41	6	34,060	1	31.167	22.952	6	39,057	1	27	1,828.562	0.128
5	10	5	51	5	34,900	1	44.200	39.919	5	38,831	1	39.200	1,837.041	0.166
6	5	6	31	7	23,560	1	20.857	15.864	-	-	-	-	1,830	-

As shown in Table 2, TWCG outperforms the COLOR CODING pricing method on ring graphs in 17 of the 21 examples. This is primarily because longer lines are possible. For example, on an instance with 65 vertices and treewidth of 8, TWCG has an LP cost of 45,248 with 96 active lines and a MILP cost of 90,747 with 4 active lines, while COLOR CODING has an LP cost of 58,441 with 8 active lines and a MILP cost of 112,762 with 6 active lines. For the most complex instance with 9 rings and 9 spokes and treewidth 9, TWCG has an LP cost of 53,045 with 21 active lines and a MILP cost of 105,208 with 5 active lines, while COLOR CODING has an LP cost of 61,460 with 15 active lines and a MILP cost of 141,501 with 8 active lines.

As expected, the difference increases as the graph grows. Since the maximum line length for COLOR CODING is limited to 20 or 25, respectively, it incurs higher fixed costs because it must cover long paths with many short lines. Furthermore, in larger instances, relatively few lines are generated for COLOR CODING, as the runtime for generating them increases. COLOR CODING works better with more colors k on the smaller instances, but on the largest instances with 73 and 82 stations, the setting with $k = 20$ outperformed $k = 25$ because more lines could be generated.

4.5 Running times

Next, we investigate the runtime of TWCG and COLOR CODING on the same concentric ring graphs, with treewidth up to 9 and at most 82 stations. Table 3 shows the comparison of runtime between the different instances. In all instances, it took less than 1 second to price the transportation paths, which is negligible.

For treewidth up to 6, pricing the lines with TWCG is practically instantaneous: 11 seconds for treewidth 6 and 37 vertices. For instances with treewidth 7, the runtime still remained reasonable at below 2 minutes. For treewidth 8 and 57 stations, TWCG took 19 minutes to price all lines. For all instances with higher treewidth, it was no longer possible to generate all columns and solve the LP relaxation to optimality within our time limit.

■ **Table 2** Ring graph comparison between TWCG and COLOR CODING (CC). The left block describes the instances, where s , r , tw , n are the number of spokes, rings, treewidth, and number of vertices. The middle block compares the number of lines and cost between TWCG and COLOR CODING in the LP relaxation. The right block compares the MILP solution. Additionally, the number of colors k and the number of iterations $iter.$ are stated.

Input				LP						MILP							
<i>s</i>	<i>r</i>	<i>tw</i>	<i>n</i>	TWCG	no. <i>l</i>	cost	CC	no. <i>l</i>	cost	TWCG	no. <i>l</i>	cost	CC	no. <i>l</i>	cost	<i>k</i>	<i>iter.</i>
3	1	3	4		2	1,160		2	1,160		2	1,351		2	1,351	25	20
3	2	3	7		8	4,418		5	4,418		3	6,219		2	6,195	25	20
3	3	3	10		8	9,612		7	9,612		2	15,450		2	15,702	25	20
4	2	4	9		6	7,906		6	7,906		3	12,166		2	12,206	20	50
4	3	4	13		11	13,646		14	13,646		3	22,625		2	22,302	25	20
4	4	4	17		23	19,487		26	19,344		2	31,389		2	31,821	20	50
5	3	5	16		25	16,390		20	16,425		4	29,733		2	28,323	25	20
5	4	5	21		33	22,395		34	22,355		2	37,964		2	38,493	25	20
5	5	5	26		37	27,003		35	27,061		2	46,290		2	49,874	25	20
6	4	6	25		40	23,874		41	24,117		2	44,152		3	48,651	25	20
6	5	6	31		49	27,778		43	28,403		3	53,758		3	59,476	25	20
6	6	6	37		66	34,174		40	34,625		4	65,071		4	70,300	25	20
7	5	7	36		54	30,619		38	31,100		3	55,827		4	66,072	25	20
7	6	7	43		72	34,354		30	35,671		3	63,850		4	78,186	25	20
7	7	7	50		84	39,559		19	43,685		4	75,337		4	88,973	25	20
8	6	8	49		90	36,820		19	38,058		3	75,883		5	89,969	25	20
8	7	8	57		97	42,094		12	46,469		4	85,276		5	101,027	25	20
8	8	8	65		96	45,248		8	58,441		4	90,747		6	112,762	25	20
9	7	9	64		62	44,084		8	53,577		5	90,570		6	111,923	25	20
9	8	9	73		20	46,995		19	51,361		3	92,354		7	124,643	20	50
9	9	9	82		21	53,045		15	61,460		5	105,208		8	141,501	20	50

COLOR CODING already reaches the 30-minute limit with treewidth 6 and 31 stations. All instances larger than that reached the threshold of 30 minutes, due to the time it takes to generate the lines. It would be possible to run the same experiments with fewer colors. We tried $k = 12$ colors as well, but then the solution quality is worse. It is better to have fewer high-quality lines in the pool than many small ones.

However, recall that COLOR CODING is not guaranteed to find the optimal solution for the longest paths, and the quality of the solution depends on the number of colors and number of iterations (relative to the instance size). However, since we keep these parameters fixed, this means that the quality of the solution found by COLOR CODING is decreasing – if we wanted to keep the quality of the solution comparable, we should increase both parameters of COLOR CODING with the size of the graph, but doing so is infeasible. As COLOR CODING generated fewer lines, the time for solving the MILP was less than 2 minutes, even in the worst instance with treewidth 6 and 31 stations.

A treewidth of 9 is the maximum treewidth the algorithm can solve within the 30-minute time limit. The join operator becomes the bottleneck, since it may have to combine every subsolution of the first solution set with many of the second solution set. Depending on the structure of the tree decomposition, the running time can change drastically (affected by, e.g., whether a join node is at the root of the tree decomposition). Unfortunately, the Jdrasil solver is nondeterministic and generates a new decomposition each run. For the treewidth-9 instances, we selected a decomposition for our experiments that gave a fast running time.

We observed that generating many lines is not necessarily advantageous for the MILP rounding due to time constraints. Furthermore, we often noticed that the final lines were generated early in the algorithm. If we stop early and use “Generation-Order Greedy Rounding” or “Early Stop MILP” as explained in Section 3.2, we can still solve the biggest instance within 6 minutes.

■ **Table 3** Comparison of the runtime on ring graphs of the different algorithmic components. The left block describes the instances, where s , r , tw , n are the number of spokes, rings, treewidth, and number of vertices. The middle block gives the results for TWCG, the right block those for CC. Shown are the number of iterations, the cumulative time for pricing the paths and lines, for solving the linear program (LP), and for solving the mixed integer linear program (MILP).

Input				TWCG						CC					
s	r	tw	n	iterations	t lines	t paths	t lp solve	t MILP solve	t total	iterations	t lines	t paths	t lp solve	t MILP solve	t total
3	1	3	4	14	0.037	0.009	0.048	0.005	0	16	0.119	0.003	0.046	0.006	0
3	2	3	7	26	0.022	0.007	0.093	0.024	0	30	0.203	0.004	0.093	0.033	0
3	3	3	10	40	0.071	0.012	0.139	0.162	0	38	0.763	0.005	0.222	0.177	1
4	2	4	9	20	0.016	0.003	0.106	0.032	0	16	0.219	0.002	0.089	0.022	0
4	3	4	13	36	0.129	0.008	0.275	0.338	1	48	2.859	0.007	0.274	0.494	4
4	4	4	17	72	0.147	0.012	0.523	1.492	2	108	16.699	0.009	0.539	1.607	19
5	3	5	16	82	0.345	0.011	0.492	1.102	2	140	20.327	0.007	0.657	1.198	22
5	4	5	21	86	1.134	0.019	0.734	1.756	4	118	86.846	0.020	0.963	2.303	90
5	5	5	26	116	1.334	0.030	1.438	18.922	22	282	676.806	0.090	3.552	10.600	691
6	4	6	25	98	4.914	0.027	1.329	9.308	16	224	463.672	0.064	2.663	76.640	543
6	5	6	31	130	5.827	0.058	2.761	18.910	28	252	1,808.483	0.122	5.158	105.383	1,920
6	6	6	37	180	11.212	0.138	4.556	257.973	274	120	1,805.300	0.100	4.689	10.331	1,821
7	5	7	36	146	44.331	0.093	3.999	64.125	113	124	1,815.915	0.092	4.871	27.963	1,849
7	6	7	43	208	100.625	0.189	7.839	403.585	513	64	1,843.678	0.084	4.449	5.225	1,854
7	7	7	50	252	110.525	0.349	19.354	900.100	1,031	38	1,852.437	0.099	8.412	40.648	1,902
8	6	8	49	294	1,023.991	0.396	11.819	267.337	1,304	38	1,821.465	0.078	4.385	14.908	1,841
8	7	8	57	368	1,142.769	0.719	22.448	900.098	2,067	24	1,883.243	0.094	6.436	4.662	1,895
8	8	8	65	324	1,788.235	0.888	36.869	900.150	2,728	16	1,843.733	0.096	3.007	0.827	1,848
9	7	9	64	184	1,790.911	0.550	27.985	115.887	1,936	16	1,856.004	0.091	5.362	1.607	1,863
9	8	9	73	40	1,824.380	0.227	18.971	2.308	1,849	38	1,841.109	0.223	21.888	7.818	1,872
9	9	9	82	42	1,882.879	0.419	12.164	3.809	1,901	30	1,929.489	0.273	17.502	13.083	1,961

4.6 Rounding analysis

Next, we compare the four rounding methods and investigate the effect of the line pool size on the quality of the MILP solution. We observed that the lines that are used in the final MILP solution are usually generated very early on in the solving process, and that most of the time towards the end is spent generating lines that do not end up being used.

In this comparison, we examine how often it is preferable to use Generation-Order Greedy Rounding rather than Frequency-Based Greedy Rounding. Furthermore, we compare the effect of a limited line pool of “Early Stop MILP” against the fully generated line pool of “Converged MILP”. In total, we run the problem set on the ring graphs three times: once for GO and ES MILP, once for FB, and once for CO MILP.

Table 4 compares the line selection criteria and the total runtime. When using TWCG, Generation-Order Greedy Rounding yields a better solution in 15 out of 21 instances. This provides evidence that, if processing time is a factor, it is possible to stop early. If the objective value of the MILP solution increases after adding a line and path set, we can remove the last line and stop.

Comparing GO against ES MILP, they appear to yield very similar solution values. The biggest difference has been observed with 9 spokes and 7 rings, resulting in a cost of 0.944 using GO, compared to 0.943 with ES MILP. This is surprising, given that GO naively rounds the LP relaxation, whereas ES MILP recalculates the MILP. However, since the runtime barely increases when moving from GO to ES MILP – at most 4 seconds on the instance with 9 spokes and 9 rings – it is always sensible to run ES MILP as well.

Using GO instead of FB, we have an average improvement of 13.7%, with the biggest improvement being 44% for 8 spokes and 7 rings, with a cost reduction from 1.501 to 0.838. The biggest worsening of 9.4% for 6 spokes and 4 rings would lead to a cost increase from 0.449 to 0.491. This, combined with a runtime speed-up by a factor of 10 to 20, makes GO the preferable method over FB.

The cost of the two MILP rounding methods is usually very close to each other. In some cases, the ES MILP actually finds a very slightly better solution. This is (again) due to nondeterminism in the tree decomposition generation. Even though the CO MILP solves the LP relaxation to optimality, it may end up with a different set of lines that behave differently during the rounding phase.

Furthermore, the larger line pool will sometimes cause the CO MILP to run out of time, as is the case with 8 spokes and 7 or 8 rings. If running CO MILP instead of ES MILP, the biggest improvement is 13.3%, with 9 spokes and 9 rings, from 1.021 to 0.886, while the lowest “improvement” is -20%, with an average improvement of 0.5%. For instances with more than 5 rings, we achieve, on average, about 20× faster runtime when using ES MILP. In the largest instances, this is no longer visible because the CO MILP runs out of time during column generation.

Overall, CO MILP gives the best solutions at the cost of a significantly increased runtime. As the average gap is only 0.5%, this experiment justifies the decision to use ES MILP for the Japan network and to stop early.

■ **Table 4** Comparison of heuristic and MILP rounding. The left block describes the instances in which s and r denote the number of spokes and rings of the circular city. The second block shows “Generation-Order Greedy Rounding” (GO) with its runtime in seconds, number of lines, and cost. The following blocks show the respective values for “Early Stop MILP” (ES MILP), “Frequency-Based Greedy Rounding” (FB), and “Converged MILP” (CO MILP).

Input		GO			ES MILP			FB			CO MILP		
s	r	t	l	cost $\times 10^5$	t	l	cost $\times 10^5$	t	l	cost $\times 10^5$	t	l	cost $\times 10^5$
3	1	0.1	2	0.014	0.1	2	0.014	0.8	2	0.014	1.5	2	0.014
3	2	0.1	2	0.065	0.2	2	0.065	0.2	3	0.076	0.2	3	0.075
3	3	0.1	2	0.166	0.2	2	0.166	0.3	3	0.195	0.5	3	0.186
4	2	0.1	2	0.125	0.1	2	0.125	0.2	2	0.123	0.2	3	0.150
4	3	0.1	2	0.250	0.2	2	0.249	0.5	2	0.244	0.5	2	0.231
4	4	0.1	2	0.358	0.3	2	0.358	0.8	2	0.345	0.9	2	0.325
5	3	0.1	2	0.318	0.3	2	0.318	0.9	3	0.339	1.5	2	0.301
5	4	0.4	2	0.373	0.7	2	0.373	2.0	2	0.426	1.5	2	0.378
5	5	0.3	2	0.514	0.7	2	0.514	2.9	3	0.525	2.8	2	0.448
6	4	0.6	2	0.491	0.7	2	0.491	6.4	2	0.449	8.2	2	0.440
6	5	0.4	2	0.502	0.8	2	0.502	8.8	4	0.654	12.5	3	0.550
6	6	1.0	3	0.639	1.7	3	0.639	16.2	4	0.737	8.0	3	0.633
7	5	4.4	3	0.617	5.0	3	0.617	48.6	4	0.713	47.4	3	0.613
7	6	5.8	4	0.747	6.3	4	0.746	109.0	3	0.708	126.7	3	0.708
7	7	8.5	3	0.779	9.2	3	0.779	130.7	7	1.093	1048.1	3	0.817
8	6	46.2	3	0.756	47.2	3	0.756	1037.0	8	1.168	1231.0	3	0.690
8	7	48.4	4	0.838	50.2	4	0.838	1166.9	11	1.501	2701.5	3	0.847
8	8	68.0	4	0.896	69.6	4	0.895	1827.5	8	1.340	2705.9	4	0.905
9	7	84.6	4	0.944	87.1	4	0.943	1820.3	9	1.390	1839.2	4	0.906
9	8	487.1	4	0.938	489.8	4	0.937	1846.3	7	1.426	1850.6	5	0.960
9	9	354.2	4	1.021	358.1	4	1.021	1897.3	4	1.053	1917.1	3	0.886

4.7 Comparison against COLOR CODING on Real-life networks

Next, we discuss the real-world examples: the Munich S-Bahn and subway network and the Japanese railway network. Table 5 shows details on the solution quality for both locations.

■ **Table 5** Comparisons of the solutions of Munich and Japan with CC and TWCG. Top-left: the network size ($|V|$, $|E|$) and the transport demand (Σd_{st}). Top-right: LP cost, MILP cost, travel cost, and operating cost. Bottom left: the number of active lines, the average frequency, and the average length of the lines. Bottom right: the cumulative time to price the lines and paths, the cumulative time to solve the LPs, and the time to solve the MILP.

Instance		Network			Costs			
Place	Method	$ V $	$ E $	Σd_{st}	LP	MILP	$cost_{travel}$	$cost_{var}$
Japan	CC	694	830	74,672	366,327	1,467,452	97,540	1,369,913
Japan	TWCG	694	830	74,672	338,468	776,061	96,705	679,356
Munich	CC	251	249	3,137,500	184,819	361,748	137,564	224,183
Munich	TWCG	251	249	3,137,500	184,840	333,373	139,013	194,360

Instance		Solution Structure			Time (s)			
Place	Method	act. Lines	$\bar{f}$	avg. len.	lines	paths	LP solve	MILP solve
Japan	CC	116	1.643	12.103	1,852	715	1,405	900
Japan	TWCG	46	0.349	121.370	16	206	2,331	18
Munich	CC	16	1.885	20.688	644	77	1,767	900
Munich	TWCG	13	1.543	27.231	1	61	1,310	197

In the Munich case, we use $k = 70$ colors and 20 iterations for COLOR CODING. In theory, using such a large number of colors should be infeasible; however, as our implementation is “positive-instance driven” (i.e., we do not enumerate infeasible entries of the DP table), it turns out to be feasible to use a large number of colors, as the very simple structure of the Munich network keeps the number of feasible solutions bounded.

The algorithm achieved an LP cost of 184,819; the travel cost is 137,564 and the operating cost is 224,183, yielding an MILP cost of 361,748. The solution has 16 active lines with an average frequency of 1.885 and an average length of 20.688. The time spent on finding the lines and paths, and on solving the LP relaxation and the MILP, is 644, 77, 1,767, and 900 seconds, respectively. The MILP timed out after 15 minutes; the MILP gap is 2.3%.

Using TWCG, we get an LP cost of 184,840; the travel cost is 139,013, and the operating cost is 194,360, yielding an MILP cost of 333,373. This results in a 7.8% reduction in MILP cost compared to COLOR CODING. The solution has 13 active lines with an average frequency of 1.543 and an average length of 27.231. The time spent on finding the lines and paths, and on solving the LP relaxation and the MILP, is 1, 61, 1,310, and 197 seconds, respectively.

Next, we compare the solutions for the Japanese network. Using COLOR CODING with $k = 20$ colors and 20 iterations, an LP cost of 366,327 is achieved. The travel cost is 97,540 and the operating cost is 1,369,913, yielding an MILP cost of 1,467,452. The solution has 116 active lines with average frequency 1.643 and average length 12.103. Time spent on pricing the lines, paths, and solving the LP relaxation and MILP is 1,852, 715, 1,405, and 900 seconds, respectively. The MILP timed out after 15 minutes; the MILP gap is 2.0%.

Using TWCG, the LP cost is 338,468; the travel cost is 96,705, and the operating cost is 679,356, yielding an MILP cost of 776,061. This results in a 47.1% reduction in MILP cost compared to COLOR CODING. The solution has 46 active lines with an average frequency of 0.349 and an average length of 121.370. The time spent on finding the lines and paths, and on solving the LP relaxation and the MILP, is 16, 206, 2,331, and 18 seconds, respectively. We observe that the bottleneck is the LP solver, as we spend only 16 seconds solving the pricing problem in the Japan case, but 39 minutes solving the LP relaxation with Gurobi.

The large difference between LP and MILP costs is because the average frequency is smaller than F , and consequently, the fixed costs are underestimated in the LP relaxation. In practice, this could be improved by tightening the estimate of F .

Using TWCG, we obtain average line lengths of 27.2 and 121.4 for Munich and Japan, respectively. Even if one were to use more colors for COLOR CODING, it is (due to its exponential behavior) impractical to use anywhere near that many colors. Munich allowed us to use many colors, but this is due to the very low average degree. We also see the advantage of TWCG in finding the lines. In the Japan case, with 694 stations, COLOR CODING accounts for over a third of the time spent searching for the lines, whereas using TWCG reduces the time to 16 seconds in total. Also, the solution quality showed the largest differences in real-world scenarios, with a 47.1% improvement in Japan and a 7.8% improvement in Munich. It is also notable that the LP costs in Munich are nearly identical; in the Japan case, however, there is a difference of 7.6%. The latter is due to early termination.

5 Conclusion

We have shown that column generation for the Line Planning Problem can be performed effectively by using treewidth-based dynamic programming for the pricing problem. We compared runtime, costs, and rounding methods for concentric ring graphs, the Munich S-Bahn and subway network, and the Japanese railway network. We improve on existing methods by solving bigger instances with higher treewidth and by improving solution quality. Furthermore, we improve on greedy rounding by introducing an alternative selection criterion, which also reduces runtime.

We can solve instances with treewidth up to 9. For treewidth up to 7, the time required to price lines is comparably negligible, as shown by the example of Japan. The computational bottleneck for networks with a large number of vertices is the LP solver. The bottleneck for solving instances with larger treewidth is the join operator. Switching to path decompositions, or using tree decompositions which have been optimized for a measure of DP “cost” rather than width, could be beneficial. However, unlike for treewidth, there is currently a lack of good pathwidth (or treecost) solvers.

Interestingly, Ziobro and Pilipczuk [29] report in their experimental evaluation of treewidth-based dynamic programming for Hamiltonian Cycle that they are able to solve some instances with treewidth up to 29 (even with the “naive” algorithm without the rank-based approach). This may seem like a large gap compared to our results, which are for a very similar algorithm. However, it is likely that these instances have a structure that limits the number of partial solutions considered to far below the worst-case bound. Even though Longest Path and Hamiltonian Cycle may seem like similar problems, in Hamiltonian Cycle, many partial solutions can likely be discarded early, since failing to visit even one vertex renders the solution infeasible immediately. In contrast, a partial solution for Longest Path cannot be excluded just because it skips one vertex.

TREewidth COLUMN GENERATION outperforms COLOR CODING because it selects longer, more effective lines while using fewer lines. This is particularly evident in real-world examples. TWCG is faster for Munich and Japan. Depending on the parameter choices, COLOR CODING can still be faster, but this results in shorter lines and a worse solution. Both heuristic and MILP-based rounding achieve significant time savings by terminating the column generation early. Our experiments show that this generally still results in good-quality solutions.

Compared to Heinrich et al.'s TW-LINES [17], we are able to handle larger networks with higher treewidth. However, it is difficult to fairly compare the two methods: Heinrich et al. solve a different problem without fixed costs per line and without passenger routing, but with integer frequencies, and solve to optimality rather than rounding an LP relaxation. Although directly using treewidth-based FPT algorithms to holistically solve line planning problems as Heinrich et al. do can provide superior (i.e., provably optimal) solutions, this is limited to cases which happen to admit FPT algorithms. Even then, the complexity of the resulting treewidth DP formulation can severely limit the values of treewidth for which such an approach is feasible. In contrast, our approach, which uses treewidth techniques combined with column generation, allows us to exploit bounded treewidth where it is most advantageous and use it to solve more complicated problems.

References

- 1 Oliver Bachtler, Irene Heinrich, Philine Schiewe, and Constantin Seebach. Non-pool-based line planning: a practical analysis of the FPT algorithm tw-lines. *Transportation Research Procedia*, 86:290–297, 2025. doi:10.1016/j.trpro.2025.04.037.
- 2 Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A Modular Library for Computing Tree Decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2017.28.
- 3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information & Computation*, 2015. doi:10.1016/j.ic.2014.12.008.
- 4 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.
- 5 Ralf Borndörfer, Martin Grötschel, and Marc E. Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 2007. doi:10.1287/trsc.1060.0161.
- 6 Oded Cats, Roberto F. Abenzoza, Chengxi Liu, and Yusak O. Susilo. Evolution of satisfaction with public transport and its determinants in Sweden: Identifying priority areas. *Transportation Research Record*, 2538(1):86–95, 2015. doi:10.3141/2538-10.
- 7 M.T. Claessens, Nico M. van Dijk, and Peter J. Zwaneveld. Cost optimal allocation of rail passenger lines. *European Journal of Operational Research*, 110(3):474–489, 1998. doi:10.1016/S0377-2217(97)00271-3.
- 8 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 9 Holger Dell, Thore Husfeldt, Bart Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The first parameterized algorithms and computational experiments challenge. *International Symposium on Parameterized and Exact Computation*, 2017. doi:10.4230/LIPIcs.IPEC.2016.30.
- 10 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. *International Symposium on Parameterized and Exact Computation*, 2018. doi:10.4230/LIPIcs.IPEC.2017.30.
- 11 Jacques Desrosiers and Marco E. Lübbecke. *A primer in column generation*, pages 1–32. Springer US, Boston, MA, 2005. doi:10.1007/0-387-25486-2_1.
- 12 John Farrington and Conor Farrington. Rural accessibility, social inclusion and social justice: towards conceptualisation. *Journal of Transport Geography*, 13(1):1–12, 2005. Sustainability and the Interaction Between External Effects of Transport (Part Special Issue, pp. 23-99). doi:10.1016/j.jtrangeo.2004.10.002.

- 13 Huiling Fu, Lei Nie, Lingyun Meng, Benjamin R. Sperry, and Zhenhuan He. A hierarchical line planning approach for a large-scale high speed rail network: The China case. *Transportation Research Part A: Policy and Practice*, 75:61–83, 2015. doi:10.1016/j.tra.2015.03.013.
- 14 Philine Gattermann, Jonas Harbering, and Anita Schöbel. Line pool generation. *Public Transport*, 9(1):7–32, 2017. doi:10.1007/s12469-016-0127-x.
- 15 Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Gurobi Optimization, LLC, 2024. URL: <https://www.gurobi.com>.
- 16 Irene Heinrich, Philine Schiewe, and Constantin Seebach. Algorithms and hardness for non-pool-based line planning. *Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, 2022. doi:10.4230/OASICS.ATMOS.2022.8.
- 17 Irene Heinrich, Philine Schiewe, and Constantin Seebach. Non-pool-based line planning on graphs of bounded treewidth. *Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, 2023. doi:10.4230/OASICS.ATMOS.2023.4.
- 18 Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008. doi:10.1007/s00453-007-9008-7.
- 19 Jingwen Huo, Jing Meng, Heran Zheng, Priti Parikh, and Dabo Guan. Achieving decent living standards in emerging economies challenges national mitigation goals for CO₂ emissions. *Nature Communications*, 14(1):6342, 2023. doi:10.1038/s41467-023-42079-8.
- 20 Münchner Verkehrs- und Tarifverbund. S-Bahn, U-Bahn Netzplan München, 2024. Map. URL: <https://www.mvv-muenchen.de/>.
- 21 OpenJDK. JDK 22. <https://openjdk.org/projects/jdk/22/>, 2024. Accessed: 2026-08-03.
- 22 Laurent Perron and Vincent Furnon. OR-Tools. URL: <https://developers.google.com/optimization/>.
- 23 Toshiki Saitoh. Personal communication regarding the Japanese rail network. Personal communication, 2016.
- 24 Anita Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 2012. doi:10.1007/s00291-011-0251-6.
- 25 Anita Schöbel and Susanne Scholl. Line planning with minimal traveling time. In Leo G. Kroon and Rolf H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*, volume 2 of *Open Access Series in Informatics (OASICS)*, pages 1–16, Dagstuhl, Germany, 2006. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ATMOS.2005.660.
- 26 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, Berlin, 2003.
- 27 Hisao Tamaki. Heuristic computation of exact treewidth. In *20th International Symposium on Experimental Algorithms (SEA 2022)*, pages 17:1–17:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SEA.2022.17.
- 28 Hisao Tamaki. A contraction-recursive algorithm for treewidth. *International Symposium on Parameterized and Exact Computation*, 2023. doi:10.48550/arXiv.2307.01318.
- 29 Michał Ziobro and Marcin Pilipczuk. Finding Hamiltonian Cycle in graphs of bounded treewidth: Experimental evaluation. *Journal of Experimental Algorithmics (JEA)*, 24:1–18, 2019. doi:10.1145/3368631.