Heinrich, Irene; Schiewe, Philine; Seebach, Constantin

Non-Pool-Based Line Planning on Graphs of Bounded Treewidth

Published in:
23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2023

DOI:
10.4230/OASICS.ATMOS.2023.4

Published: 01/09/2023

Document Version
Publisher's PDF, also known as Version of record

Non-Pool-Based Line Planning on Graphs of Bounded Treewidth

Irene Heinrich
TU Darmstadt, Germany

Philine Schiewe
Aalto University, Espoo, Finland

Constantin Seebach
RPTU Kaiserslautern-Landau, Kaiserslautern, Germany

Abstract

Line planning, i.e. choosing routes which are to be serviced by vehicles in order to satisfy network demands, is an important aspect of public transport planning. While there exist heuristic procedures for generating lines from scratch, most theoretical investigations consider the problem of choosing lines only from a predefined line pool. We consider the line planning problem when all simple paths can be used as lines and present an algorithm which is fixed-parameter tractable, i.e. it is efficient on instances with small parameter. As a parameter we consider the treewidth of the public transport network, along with its maximum degree as well as the maximum allowed frequency.

2012 ACM Subject Classification Applied computing → Transportation; Theory of computation → Fixed parameter tractability; Mathematics of computing → Integer programming; Theory of computation → Discrete optimization

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

Digital Object Identifier 10.4230/OASIcs.ATMOS.2023.4

Supplementary Material *Software (Source Code)*: `https://github.com/urinstinkt/lptw`

Funding *Irene Heinrich*: The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

Acknowledgements This work was partially developed during a guest stay of the first author at the Aalto University in Espoo, Finland.

1 Introduction

Motivation. Automating public transport planning is a challenging task and traditional approaches split it into multiple stages, as seen in [13] and Figure 1. *Lines* form a foundational building block for all following planning stages. In this context, lines are (simple) paths in the public transport network that have to be covered by vehicles end-to-end. Which lines are chosen highly impacts the subsequent planning steps like timetabling and vehicle scheduling. On the one hand, lines influence the routes and transfers that passengers take, determining the network quality from the passenger's perspective, and on the other hand, they determine the majority of the operating costs.

Line planning refers to selecting a subset of lines and their frequencies, called *line concept*, from a given set of lines, the *line pool*. While there is ample literature on line planning for a given fixed line pool, see [20], the construction of line pools is often neglected.

Instead of designing a line concept from a given line pool, we consider the set of all simple paths as candidates. This greatly extends the solution space and has a high potential to give better results. Thus our approach integrates line pool generation and line planning phases

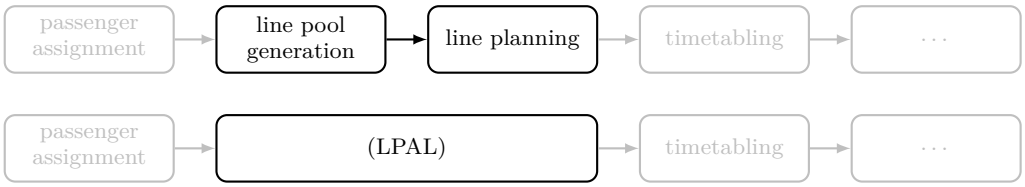

Figure 1 Common sequential approach for public transport planning, adapted from [13] (top). We consider the integration of line pool generation and line planning into a single step (bottom).

into one single problem which we call *line planning on all paths* (LPAL). This integrated problem is not yet thoroughly researched, and although its hardness has been investigated recently [15], not much is known about how to solve it efficiently in practice, if that is even possible, and how much it improves over previous methods.

Throughout the research literature on line planning, different objective functions have been considered. From the passengers' perspective, one wants to maximize the number of direct travelers [7] or to minimize the travel time [21, 6]. Here, it is especially difficult to model passenger behavior realistically, see [12, 19]. In this paper we focus on minimizing the operating costs of a line concept as originally introduced in [8], where assigning passenger routes in a previous step guarantees that passengers can travel on favorable routes, see e.g. [7]. Our cost model includes path-dependent and -independent costs, where the former can be used to model costs for the distance covered on a line and the latter can represent costs of maintaining a vehicle. All costs are frequency-dependent, meaning they scale with the number of vehicles operated per line. We do not model frequency-independent costs, since it was shown [15] that doing so makes (LPAL) NP-hard even on the most simple graph classes.

It was shown previously in [15] that (LPAL) is NP-hard and cannot even be approximated to a reasonable degree in polynomial time, assuming $P \neq NP$. This is supported by the fact that all but the most elementary families of graphs exhibit an exponential number of possible simple paths in terms of the graph order. In fact it is unknown whether (LPAL) is even contained in NP since the line concept could be any subset of an exponentially sized line pool – simply writing it down may not be possible in polynomial time. Given these hardness results, the question arises whether it is at all feasible to solve (LPAL) in practice.

This motivated us to investigate the parameterized complexity of (LPAL). We consider the problem's complexity not only depending on the input size, but also depending on some extra parameter. A parameterized problem is called *fixed parameter tractable (FPT)* if we can solve it in time $f(k)n^{\mathcal{O}(1)}$, where k is the parameter, n is the input size, and f is an arbitrary function. Obtaining such a result furthers our theoretical understanding of the problem, discerning what exactly makes instances hard to solve. Crucially, an FPT-algorithm can also be useful in practice, if the evaluated instances can be expected to have a small parameter.

In particular, we consider the graph parameter *treewidth*, which was introduced by Robertson and Seymour [18] and has become an indispensable graph invariant for studying algorithmic problems which are intractable in their general form. At its core, treewidth captures the notion of how close a graph is to being a tree. One could say that graphs of bounded treewidth are "thickened trees". Trees have a simple and hierarchical structure, making them easier to analyze and work with compared to more complex graphs. Treewidth generalizes this concept by allowing cycles and measuring the extent to which a graph deviates from a tree-like structure.

Numerous problems are linear-time solvable on graphs for which the treewidth is restricted [1, 2]. This is the case, for example for *Maximum independent set*, *Chromatic number* and *Hamiltonian circuit*. The latter is especially relevant, since the problem of finding a

Hamiltonian path is used to obtain various hardness results for line planning [15]. In fact, Courcelle [9] showed in his seminal work that every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth. Furthermore *Integer linear programming (ILP)*, which is ubiquitous in discrete optimization, becomes tractable when the constraint interaction graph has bounded treewidth and the variables have a bounded domain [10].

From a practical view-point, treewidth is a very natural parameter to consider in the context of transportation planning. For example, networks modeling cities that developed along an arterial road or near a river can be intuitively understood as being like "trees, but thicker", i.e. having comparatively small treewidth. More abstract examples are grids and ring graphs (concentric rings connected by spokes). It turns out that when the number of spokes is fixed, the resulting networks have bounded treewidth, no matter how many rings we add. Similarly, when we consider grids where one of the dimensions is fixed and the other one arbitrary, we have bounded treewidth [4]. Many street networks contain such graphs as substructures (striking examples are New York or Paris).

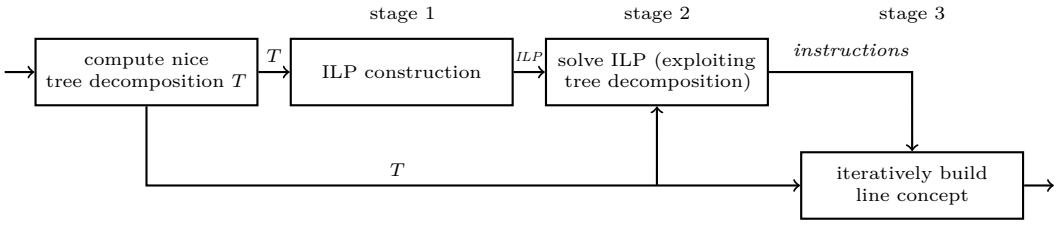

Figure 2 Overview of our line planning algorithm. In the preprocessing stage, a tree decomposition T of the input graph is computed, which is used in all stages of the algorithm. First we construct an ILP (stage 1). Then this ILP is solved, exploiting the given tree-like structure (stage 2). The solution is fed into our assembly algorithm (stage 3) which finally constructs an optimal line concept.

When a predefined line pool is given, line planning can be solved using *integer linear programming (ILP)* in a straight forward way. A variable is introduced for every line in the line pool, representing the frequency of this particular line. Feasibility and costs of the line concept are then easily modeled. It remains to solve the resulting integer program. This remaining task, however, is practical only for small line pools, since no efficient algorithmic solutions are known. When solving (LPAL), all possible simple paths must be considered as line candidates. Indeed, since the number of possible paths of a graph grows exponentially in the number of vertices, using this straight forward approach would result in an integer program with an exponential number of variables, hence we can expect a doubly exponential running-time in the worst case.

Contribution. In this paper we develop an algorithm solving (LPAL) on graphs $G = (V, E)$ of maximum degree Δ with treewidth k and vehicle frequencies bounded by M in time $\mathcal{O}(g_1(k, M\Delta)|V| + g_2(k)|V|^2)$ for some functions g_1 and g_2. In other words, we show (LPAL) is FPT when parameterized by treewidth combined with maximum degree and maximum frequency. Our algorithm can be broken down into multiple stages, as shown in Figure 2. First we need to compute a tree decomposition of the input graph. Using it we construct an ILP having a number of variables which is linear in the number of vertices (stage 1). We prove that if our input graphs additionally have bounded degree and edge frequencies, this ILP can be solved in polynomial time (stage 2). The optimal solution provides the *instructions* for building an optimal line concept. Finally we give an algorithm that carries out these instructions in polynomial time (stage 3), hence solving (LPAL).

We evaluated the practicality of our algorithm by measuring its running time on a set of algorithmically generated instances. Additionally we compare the resulting line concepts to those obtained by a heuristic line pool generation approach [11]. Here our algorithm manages to reduce the costs by 36% on average.

2 Preliminaries

Graph Theory. Let $G = (V, E)$ be a graph and $V' \subseteq V$. The subgraph of G induced by V' is $G[V'] \coloneqq (V', \{e \in E \colon e \subseteq V'\})$. Let V be a set of vertices. The complete graph on V is $K(V) \coloneqq (V, \{\{u, v\} \colon u, v \in V \text{ with } u \neq v\})$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs. Their union is $G_1 \cup G_2 \coloneqq (V_1 \cup V_2, E_1 \cup E_2)$.

Line Planning. A *line planning instance* is a tuple $(G, c_{\text{fix}}, c, f^{\min}, f^{\max})$, where

- $G = (V, E)$ is a graph representing a public transport network,
- $c_{\text{fix}} \in \mathbb{R}_{\geq 0}$ represents *frequency-dependent fixed costs*,
- $c \colon E \to \mathbb{R}_{\geq 0}$, $e \mapsto c_e$ is a map representing the *edge-dependent costs*, and
- $f^{\min}$ and $f^{\max}$ are *integer frequency restrictions* on E, $e \mapsto f_e^{\min}$ (respectively $e \mapsto f_e^{\max}$) such that $f_e^{\min} \leq f_e^{\max}$ for all edges $e \in E$. Note that the lower frequency restrictions $f_e^{\min}$ allow for passengers traveling on favorable routes while the upper frequency restrictions $f_e^{\max}$ represent safety constraints.

Paths and lines. Let $G = (V, E)$ be a graph. We denote the set of all paths which are subgraphs of G by $\mathcal{P}(G)$. We define $\hat{\mathcal{P}}(G) \coloneqq \mathcal{P}(K(V))$, which includes also paths using edges absent in G. Any sequence $v_1, \dots, v_k$ of $k \geq 2$ vertices defines a path $p \in \hat{\mathcal{P}}(G)$. To shorten notation we will simply write $p = v_1, \dots, v_k$, slightly abusing the $=$ sign. Note that the reverse sequence $v_k, \dots, v_1$ defines exactly the same path, hence in our notation we treat the sequences $v_1, \dots, v_k$ and $v_k, \dots, v_1$ as equal. Let $p \in \hat{\mathcal{P}}(G)$ be defined by $v_1, \dots, v_k$, and $W \subseteq V$. The sub-sequence of $v_1, \dots, v_k$ which contains only vertices contained in W defines the path $p|_W$.

A *line concept* $(\mathcal{L}, f)$ is a set of paths $\mathcal{L} \subseteq \mathcal{P}(G)$, also called *lines*, with a *frequency vector* $f = (f_\ell)_{\ell \in \mathcal{L}} \in \mathbb{N}^{\mathcal{L}}$, i.e. f_ℓ is the *frequency* of line ℓ.

Let U be a set. A *multiset over the universe* U is a vector $m \in \mathbb{N}^U$. We can add multisets together just like vectors. Define $\operatorname{supp} m \coloneqq \{x \in U \colon m(x) > 0\}$ and $|m| \coloneqq \sum_{x \in U} m(x)$. To represent a line concept $(\mathcal{L}, f)$ we really just need the multiset $f \in \mathbb{N}^{\mathcal{P}(G)}$, since $\mathcal{L} = \operatorname{supp} f$.

At each edge $e \in E$, the lines sum up to a total frequency

$$F_e^{(\mathcal{L},f)} = \sum_{\ell \in \mathcal{L} \colon e \in E(\ell)} f_\ell,$$

where $E(\ell)$ denotes the edge set of ℓ. A line concept is *feasible* if for each edge $e \in E$ the frequency restrictions are satisfied, i.e. $f_e^{\min} \leq F_e^{(\mathcal{L},f)} \leq f_e^{\max}$.

The *cost of a path* $p \in \mathcal{P}(G)$ is $\operatorname{cost}(p) \coloneqq c_{\text{fix}} + \sum_{e \in E(P)} c_e$. We define $\operatorname{cost}((\mathcal{L}, f)) \coloneqq \sum_{\ell \in \mathcal{L}} f_\ell \cdot \operatorname{cost}(\ell)$. An equivalent representation is $\operatorname{cost}((\mathcal{L}, f)) = \sum_{e \in E} F_e^{(\mathcal{L},f)} c_e + c_{\text{fix}} \cdot \sum_{\ell \in \mathcal{L}} f_\ell$.

With this notation, we can formally define the line planning on all lines problem.

▶ **Definition 1.** *Given a line planning instance, the* line planning on all lines problem (LPAL) *is to find a feasible line concept with minimal costs.*

Tree decompositions. A *tree decomposition* of a graph G is a tuple $(T, \mathcal{B})$ where T is a tree and $\mathcal{B} = \{B_t : t \in V(T)\}$ is a family of subsets of $V(G)$, one for each vertex of T such that

(i) $\bigcup_{t \in V(T)} B_t = V(G)$,
(ii) for every edge $uv \in E(G)$ there exists a $t \in V(T)$ with $\{u, v\} \subseteq B_t$, and
(iii) every triple t_1, t_2, t_3 of vertices in $V(T)$ satisfies: if t_2 is on the unique t_1-t_3-path in T, then $B_{t_1} \cap B_{t_3} \subseteq B_{t_2}$.

The *width* of a tree decomposition $(T, \{B_t : t \in V(T)\})$ is $\max_{t \in V(T)} |B_t| - 1$. The minimum width over all tree decompositions of a graph G is the *treewidth* of G.

Let G be a graph of treewidth k. A tree decomposition $(T, \{B_t : t \in V(t)\})$ of G is *nice* if its width is k and T can be rooted at a vertex r such that

- every vertex of T has at most two children,
- if a vertex $t \in T$ has two children t_1 and t_2, then $B_{t_1} = B_{t_2} = B_t$ (t is a *join node*),
- if a vertex $t \in T$ has exactly one child t', then either $B_t \subsetneq B_{t'}$ and $|B_t| = |B_{t'}| - 1$ (t is a *forget node*) or $B_{t'} \subsetneq B_t$ and $|B_t| = |B_{t'}| + 1$ (t is an *introduce node*),
- if t is a leaf of T, then $|B_t| = 1$ (t is a *leaf node*), and
- $|V(T)| \in \mathcal{O}(k|V|)$.

Kloks [16] proved that every graph G has a nice tree decomposition. Nice tree decompositions are a useful tool that simplify the derivation of algorithms which are parametrized in the treewidth of the input graph.

3 Line planning is FPT

Assume we are given an instance $I = (G, c_{\text{fix}}, c, f^{\min}, f^{\max})$ of (LPAL) where G has treewidth k and maximum degree Δ, together with a nice tree decomposition $(T, \mathcal{B})$ of G of width k. Let r be the root of T. Without loss of generality, we can assume $B_r = \emptyset$ (this can be achieved by adding up to $k+1$ additional forget nodes). We want to work along the tree decomposition, from the bottom up, hence the following definitions are useful: For $t \in V(T)$ we set

$$G_t \coloneqq G\left[\bigcup_{\substack{t' \in V(T):\, t' \text{ is a} \\ \text{descendant of } t \text{ in } T}} B_{t'}\right] \quad \text{and} \quad G_t^+ \coloneqq G_t \cup K(B_t).$$

Note that G_t^+ may contain edges that are not present in the original graph G. These *virtual* edges are only used temporarily by our algorithm, as an intermediate step in constructing lines. We found that when we want to build up a line concept by using a sequence of local modifications, the virtual edges are a crucial ingredient. At the tree decomposition's root, no virtual edges are present and it holds: $G = G_r = G_r^+$.

Our (LPAL)-algorithm consists of three stages (see Figure 2):

1. construct an ILP,
2. solve that ILP,
3. iteratively construct a line concept, guided by the ILP solution.

We first present the procedure of stage 3, as it motivates the ILP construction. This stage can be understood on its own, with the caveat of some values being "to be determined".

3.1 Path operations and path patterns

Our algorithm constructs an optimal solution by starting with an empty line concept and iteratively applying the four following *path operations*. They change the line concept only locally, hence they are especially suited for graphs of bounded treewidth. In the following we view line concepts as multisets of simple paths.

Initialization. We add a single-edge path containing exactly two vertices from $V(G)$ to a line concept. This edge does not necessarily exist in G.

Extension. Let $p = u_1u_2 \dots u_k \in \hat{\mathcal{P}}(G)$ and $v \in V(G) \setminus V(p)$. We say that $p' \coloneqq vu_1u_2 \dots u_k$ is the *extension of* p at u_1 with v. This relation is denoted by $p \xrightarrow{(u_1, vu_1)} p'$.

Subdivision. For $p = u_1u_2 \dots u_k \in \hat{\mathcal{P}}(G)$ and $v \in V(G) \setminus V(p)$ we say that $p' \coloneqq u_1 \dots u_i v u_{i+1} \dots u_k$ is the *subdivision of* p at u_iu_{i+1} with v. This relation is denoted by $p \xrightarrow{(u_iu_{i+1}, u_ivu_{i+1})} p'$.

Join. Let $V_1, V_2 \subseteq V$ such that $B \coloneqq V_1 \cap V_2 \neq \emptyset$. Let $p \in \hat{\mathcal{P}}(G)$, and define $p_1 \coloneqq p|_{V_1}$ and $p_2 \coloneqq p|_{V_2}$. We say p is the *join* of p_1 and p_2 at B. This relation is denoted by $(p_1, p_2) \xrightarrow{B} p$.

Path patterns. We now define a formal notion that allows us to focus on the local behavior of path operations on a subset of vertices B, discarding non-local information.

Let G be a graph and $B \subseteq V(G)$ a vertex subset of G. For a path $p \in \hat{\mathcal{P}}(G)$ we set $\pi_B(p)$ to be the sequence obtained in the following way: Replace every occurrence of vertices $u \notin B$ in p by $\square$. Then replace any runs of multiple $\square$-symbols by a single $\square$. Every output of π_B which is not the singleton $\square$ is a *path pattern on* B. The set of all path patterns on B is denoted by $\mathrm{Pat}(B)$, that is $\mathrm{Pat}(B) = \pi_B(\hat{\mathcal{P}}(G)) \setminus \{\square\}$. Observe that every path pattern contains at least two symbols, no two $\square$'s are consecutive, and every vertex of B appears at most once. Further, the length of a path pattern is at most $2|B| + 1$ (the bound is tight if and only if every symbol of B appears exactly once and every second symbol is a $\square$). We can obtain every pattern of $\mathrm{Pat}(B)$ by choosing a permutation of a subset of B and for each two consecutive elements of the permutation choosing whether they should be separated by a $\square$. Hence

$$|\mathrm{Pat}(B)| = \sum_{m=1}^{|B|} m! \cdot 2^{m+1} \leq (|B|)! \cdot 2^{|B|+2}.$$

Path operations can be extended to also work on path patterns. Then π_B has useful properties in relation to path operations. Let $p, p', p_1, p_2 \in \hat{\mathcal{P}}(G)$ and $u, v, w \in B$. It holds:

- $p \xrightarrow{(u,wu)} p'$ if and only if $\pi_B(p) \xrightarrow{(u,wu)} \pi_B(p')$,
- $p \xrightarrow{(uv,uwv)} p'$ if and only if $\pi_B(p) \xrightarrow{(uv,uwv)} \pi_B(p')$,
- $(p_1, p_2) \xrightarrow{B} p'$ if and only if $(\pi_B(p_1), \pi_B(p_2)) \xrightarrow{B} \pi_B(p')$.

Let $p \in \hat{\mathcal{P}}(G)$ and $B' \subseteq B$. Then $\pi_{B'}(p) = \pi_{B'}(\pi_B(p))$.

3.2 Assembly algorithm

Algorithm 1, when called on node t of the tree decomposition, computes a line concept for the graph G_t^+. We have split the sub-procedures of Algorithm 1 into Algorithm 2, Algorithm 3 and Algorithm 4. The algorithm needs to be supplied with the tree decomposition $(T, \mathcal{B})$ and some integers i_{uv}^t, $e_{p,p'}^t$, $s_{p,p'}^t$ and j_{p_1,p_2}^t for each node $t \in T$. These integers control how many path operations are applied, and the meaning of their subscripts will become apparent from reading Algorithm 1. We call them *instruction variables*; in Subsection 3.3 we discuss how to determine their values.

▶ **Theorem 2.** *There are functions $g_1, g_2 : \mathbb{N} \to \mathbb{N}$ such that Algorithm 1 produces a line concept $\mathcal{L}$ with $|\operatorname{supp} \mathcal{L}| \in \mathcal{O}(ng_1(k))$ and runs in time $\mathcal{O}(n^2 g_2(k))$, where $n \coloneqq |V(G)|$.*

Algorithm 1 Line concept assembly algorithm.

function ASSEMBLE(t)
  if t is a leaf **then**
    return $\{\}$ ▷ empty line concept
  else if t is a forget node **then**
    $t' =$ unique child of t
    return ASSEMBLE(t')
  else if t is an introduce node **then**
    $t' =$ unique child of t
    $w' =$ the vertex introduced by t
    return ASSEMBLE_INTRODUCE(t, t', w)
  else if t is a join node **then**
    $t_1, t_2 =$ children of t
    return ASSEMBLE_JOIN(t, t_1, t_2)
  end if
end function

The proof of Theorem 2 can be found in Appendix A.

Algorithm 2 Line concept assembly sub-procedure: introduce.

function ASSEMBLE_INTRODUCE(t, t', w)
  $L =$ ASSEMBLE(t')
  for $v \in B_{t'}$ **do**
    L += (wv, i^t_{wv})
  end for
  for $p, p' \in \mathrm{Pat}(B_t)$ and $u \in B_{t'}$ with $p \xrightarrow{(u,wu)} p'$ **do**
    $L' =$ SUBTRACT($L, p, e^t_{p,p'}$)
    L += EXTEND(L', u, w)
  end for
  for $p, p' \in \mathrm{Pat}(B_t)$ and $u, v \in B_{t'}$ with $p \xrightarrow{(uv,uwv)} p'$ **do**
    $L' =$ SUBTRACT($L, p, s^t_{p,p'}$)
    L += SUBDIVIDE(L', uv, w)
  end for
  return L
end function

3.3 ILP construction

In the following we construct an integer linear program $\mathcal{P}_{(\mathrm{LPAL})}(I, (T, \mathcal{B}))$ from the (LPAL) instance I and the tree decomposition $(T, \mathcal{B})$. By solving it, we can determine the instruction variable values which lead to an optimal feasible line concept. The ILP constraints must ensure that each path operation the assembly algorithm wants to apply is possible, and that the final line concept is feasible. The ILP objective corresponds to the cost of the resulting line concept, hence minimizing it leads to an optimal solution to (LPAL).

For each node $t \in T$ we construct a set of constraints, depending on the node type (leaf, introduce, forget, join) of t, which become a part of the whole ILP. The constraint variables are shared between neighboring nodes.

Algorithm 3 Line concept assembly sub-procedure: join.

```
1:  function ASSEMBLE_JOIN(t, t_1, t_2)
2:      L = {}
3:      L_1 = ASSEMBLE(t_1)
4:      L_2 = ASSEMBLE(t_2)
5:      for p_1, p_2, p' ∈ Pat(B_t) with (p_1, p_2) --B_t--> p' do
6:          L'_1 = SUBTRACT(L_1, p_1, j^t_{p_1,p_2})
7:          L'_2 = SUBTRACT(L_2, p_2, j^t_{p_1,p_2})
8:          L += JOIN(L'_1, L'_2, B_t)
9:      end for
10:     return L + L_1 + L_2
11: end function
```

Algorithm 4 Line concept subtraction.

```
1:  function SUBTRACT(L, p, c)
2:      L' = {}
3:      while c > 0 do
4:          (path, count) = L.find(p)
5:          if count > c then
6:              L -= (path, c)
7:              L' += (copy(path), c)
8:              c = 0
9:          else
10:             L.erase(path)
11:             L' += (path, count)
12:             c -= count
13:         end if
14:     end while
15:     return L'
16: end function
```

We have 6 different flavors of variables associated with each node t, describing what happens at t during the line concept construction process:

- i^t_{uv}: How many copies of the path uv are *initialized*?
- $e^t_{p,p'}$: How often do we *extend* a path of pattern p into a path of pattern p'?
- $s^t_{p,p'}$: How often do we *subdivide* a path of pattern p into a path of pattern p'?
- $j^t_{p_1,p_2}$: How often do we *join* a path of pattern p_1 with a path of pattern p_2?
- c^t_p: How many paths of pattern p do we have, after the construction finishes node t?
- f^t_e: What is the frequency of the resulting line concept on edge e?

The c^t_p- and f^t_e-variables simply track the current construction state, hence we call them *state variables*. They are firmly constrained using the following equality constraints:

Leaves. Here Algorithm 1 returns an empty line concept, thus

$$\text{for } p \in \text{Pat}(B_t): \quad c^\ell_p = 0$$

Join nodes. Algorithm 1 effectively sums up the children's line concepts, but joins some of the lines, depending on the $j^t_{p_1,p_2}$-variables. Each individual join removes two lines and adds a new one:

$$\text{for each } p \in \mathrm{Pat}(B_t)\colon\ c^t_p = c^{t_1}_p + c^{t_2}_p + \sum_{(p_1,p_2)\xrightarrow{B_t}p} j^t_{p_1,p_2} - \sum_{(p,p_2)\xrightarrow{B_t}p'} j^t_{p,p_2} - \sum_{(p_1,p)\xrightarrow{B_t}p'} j^t_{p_1,p}$$

Forget nodes. Let w be the forgotten vertex. No changes are made to the child's line concept, but we project the lines onto a smaller set of path patterns.

$$\text{for } p \in \mathrm{Pat}(B_t)\colon \qquad c^t_p = \sum_{\substack{p' \in \mathrm{Pat}(B_{t'}) \\ \pi_{B_t}(p') = p}} c^{t'}_{p'}$$

Additionally we track the frequencies of each forgotten edge $e \in E^- \coloneqq \{\{w,u\}\colon u \in B_t\}$.

$$\text{for } e \in E^-\colon \qquad f^t_e = \sum_{\substack{p' \in \mathrm{Pat}(B_{t'}) \\ e \text{ on } p'}} c^{t'}_{p'}$$

Since the forgotten vertex cannot appear again further up in the tree decomposition, these frequencies will remain fixed from this point forward, i.e. f^t_e is also the final line concept's frequency of e. It is possible that $E^- \nsubseteq E(G)$, but the final line concept cannot be allowed to use edges outside of $E(G)$, hence we demand

$$\text{for } e \in E^- \setminus E(G)\colon \qquad f^t_e = 0$$

For actual edges of G we use the following constraint to ensure feasibility of the final line concept:

$$\text{for } e \in E^- \cap E(G)\colon \qquad f^{\min}_e \le f^t_e \le f^{\max}_e$$

Introduce nodes. Let w be the introduced vertex. Algorithm 1 continues with the child's line concept, adds and transforms some of the lines. The newly added lines consist of a single edge; they can be produced only by the introduction operation, hence

$$\text{for } p = wv \in \mathrm{Pat}(B_t) \text{ with } v \in B_{t'}\colon \qquad c^t_p = i^t_{wv}$$

For other path patterns $p \in \mathrm{Pat}(B_t)$ that contain w, we have three cases: Firstly, if w is at the end of p and is next to another vertex $u \in B_t$, then lines having the pattern p are precisely the ones that result from the extension operation:

$$\text{for } p \xrightarrow{(u,wu)} p'\colon \qquad c^t_{p'} = e^t_{p,p'}$$

Secondly, if w has two neighboring vertices $u, v \in B_t$ in p, then lines having the pattern p are precisely the ones that result from the subdivision operation:

$$\text{for } p \xrightarrow{(uv,uwv)} p'\colon \qquad c^t_{p'} = s^t_{p,p'}$$

Thirdly, if w is next to a $\square$ in p, then no line having the pattern p can be created by Algorithm 1:

$$\text{for } p \in \mathrm{Pat}(B_t) \text{ containing } w \text{ next to } \square\colon \qquad c^t_p = 0$$

The case of $p \in \mathrm{Pat}(B_t)$ not containing w remains, i.e. $p \in \mathrm{Pat}(B_{t'})$. Here we find the lines of child's line concept, but we have to subtract lines that were transformed using the extension or subdivision operations:

$$\text{for } p \in \mathrm{Pat}(B_{t'}): \qquad c_p^t = c_p^{t'} - \sum_{p \xrightarrow{(u,wu)} p'} e_{p,p'}^t - \sum_{p \xrightarrow{(uv,uwv)} p'} s_{p,p'}^t$$

Now our variables can track the results of Algorithm 1 at each node, but under the assumption that all operations prescribed by the instruction variables actually succeed. We do not want to assume, but guarantee this, hence we need more constraints.

Operation applicability constraints. Firstly each instruction variable must be non-negative. For introduce nodes we have:

$$\text{for each } p, p' \in \mathrm{Pat}(B_t) \text{ and } u \in B_{t'} \text{ with } p \xrightarrow{(u,wu)} p': \qquad e_{p,p'}^t \geq 0$$

$$\text{for each } p, p' \in \mathrm{Pat}(B_t) \text{ and } \{u, v\} \subseteq B_{t'} \text{ with } p \xrightarrow{(uv,uwv)} p': \qquad s_{p,p'}^t \geq 0$$

$$\text{for each } v \in B_{t'}: \qquad i_{wv}^t \geq 0$$

And for join nodes:

$$\text{for each } p_1, p_2, p' \in \mathrm{Pat}(B_t) \text{ with } (p_1, p_2) \xrightarrow{B_t} p': \; j_{p_1,p_2}^t \geq 0$$

Then we need to ensure that our operations do not subtract more lines than available. For introduce nodes this can be expressed as

$$\text{for } p \in \mathrm{Pat}(B_{t'}): \quad c_p^{t'} \geq \sum_{p \xrightarrow{(u,wu)} p'} e_{p,p'}^t + \sum_{p \xrightarrow{(uv,uwv)} p'} s_{p,p'}^t,$$

$$\text{or equivalently:} \quad c_p^t \geq 0.$$

The join operation subtracts a line from each child line concept, hence we need two constraints for each $p \in \mathrm{Pat}(B_t)$:

$$c_p^{t_1} \geq \sum_{(p,p_2) \xrightarrow{B_t} p'} j_{p,p_2}^t$$

$$c_p^{t_2} \geq \sum_{(p_1,p) \xrightarrow{B_t} p'} j_{p_1,p}^t$$

This concludes the ILP constraints. Now we will define the linear objective of our ILP.

Objective. The final line concept's cost can be determined by counting the effective number of lines and the frequencies at each edge. The f_e^t-variables already keep track of the edge frequencies. For each $e \in E(G)$ we have a unique forget node t where e is forgotten. In our objective, f_e^t gets the weight of c_e.

The number of lines can be counted by counting the number of path operations. The introduce operation increases the number of lines, whereas a join reduces the number of lines. Subdivision and extension operations make no change. Hence in our objective all i_{wv}^t-variables get a weight of c_{fix} and all j_{p_1,p_2}^t-variables get a weight of $-c_{\mathrm{fix}}$.

All other variables have a weight of 0. We arrive at the following linear objective:

$$\sum_{\substack{t \in T,\, e \in E \\ t \text{ is a forget node} \\ t \text{ forgets } e}} c_e \cdot f_e^t + \sum_{\substack{t \in T \\ t \text{ is an introduce node} \\ wv \in \mathrm{Pat}(B_t)}} c_{\mathrm{fix}} \cdot i_{wv}^t - \sum_{\substack{t \in T \\ t \text{ is a join node} \\ (p_1,p_2) \xrightarrow{B_t} p}} c_{\mathrm{fix}} \cdot j_{p_1,p_2}^t$$

▶ **Theorem 3.** *Feeding the solution of $\mathcal{P}_{(\mathrm{LPAL})}(I,(T,\mathcal{B}))$ into Algorithm 1 yields an optimal solution to I.*

Proof. The statement follows from the following two claims (i.e. we produce a feasible solution to I that is at least as good as an optimal one):

1. Let x be a solution to $\mathcal{P}_{(\mathrm{LPAL})}(I,(T,\mathcal{B}))$ of cost c. Then Algorithm 1, when supplied with x, produces a feasible line concept of cost c.
2. There exists a function ψ such that for any feasible line concept $\mathcal{L}$ of cost c it holds: $\psi(\mathcal{L})$ is a feasible solution to $\mathcal{P}_{(\mathrm{LPAL})}(I,(T,\mathcal{B}))$ of cost c.

Claim 1 follows directly from the construction of $\mathcal{P}_{(\mathrm{LPAL})}(I,(T,\mathcal{B}))$, where we already argued the correctness of each constraint.

Claim 2 is proved in Appendix B. ◀

3.4 Solving the ILP

We will now show that the dynamic programming approach of [10, Theorem 6] can be applied to solve our ILP, proving that (LPAL) is FPT when parameterized by treewidth, $f^{\max}$ and Δ. Let M be an upper bound for $f^{\max}$, i.e. for all $e \in E$ we have $f_e^{\max} \leq M$.

First note that the total number of variables and constraints of the ILP can be bounded by $\mathcal{O}\left(|T| \max_{t' \in T} |\mathrm{Pat}(B_{t'})|^2\right) = \mathcal{O}\left(|V| k \left((k+1)! \cdot 2^{k+3}\right)^2\right)$.

Now we have to consider the *incidence graph* $H_{\mathcal{I}}$ of our ILP and provide an upper bound for its treewidth. The vertices of $H_{\mathcal{I}}$ are composed of the variables as well as the constraints of the ILP. A variable v is adjacent to a constraint a in the graph if and only if v occurs in a. The given nice tree decomposition $(T,\mathcal{B})$ can be transformed into a tree decomposition $(T,\mathcal{B}_{\mathcal{I}})$ of $H_{\mathcal{I}}$ by defining the bag B'_t associated with $t \in T$, informally, as follows:

$$B'_t := \{\text{constraints of t}\} \cup \{\text{variables of t}\} \cup \{\text{variables of all children of t}\}.$$

It is possible to bound

$$|B'_t| \in \mathcal{O}\left(\max_{t' \in T} |\mathrm{Pat}(B_{t'})|^2\right) = \mathcal{O}\left(\left((k+1)! \cdot 2^{k+3}\right)^2\right),$$

hence the treewidth of $H_{\mathcal{I}}$ can be bounded by a function in the treewidth of G.

We also need to bound the absolute value of every variable for any feasible assignment. All variables are non-negative, i.e. we only need to provide upper bounds. The variables $f_e^t \leq f_e^{\max}$ are already taken care of. We observe that for any feasible assignment, for all $t \in T$ and $p \in \mathrm{Pat}(B_t)$ it must hold: $c_p^t \leq \Delta \cdot f_e^{\max}$. This is because any path pattern $p \in \mathrm{Pat}(B_t)$ must contain at least one vertex v of G, and any path fitting p must walk over some edge incident to v. The total maximum frequency of these edges cannot exceed $\Delta \cdot f_e^{\max}$. It follows that the j_{p_1,p_2}^t-variables also have to respect this bound. The remaining variables irreversibly increase some c_p^t, hence they are bounded by $\Delta \cdot M$ as well. Therefore we define our bound $\Gamma := \Delta M$ and can apply [10, Theorem 6] to obtain an algorithm solving the ILP in time $\mathcal{O}(g(k, M\Delta)|V|)$ for some function g.

▶ **Corollary 4.** *On any graph $G = (V,E)$ of maximum degree Δ with treewidth k and $f^{\max} \leq M$, the problem* (LPAL) *can be solved in time $\mathcal{O}(g_1(k, M\Delta)|V| + g_2(k)|V|^2)$ for some functions g_1 and g_2.*

Proof. We use Bodlaender's algorithm [3] to compute a tree decomposition of width k for G in linear time (assuming k is fixed). We convert it into a nice tree decomposition [16]. Then we apply our algorithms. Combining Theorem 3 with Theorem 2 and the running time for solving the ILP, the claim follows. ◀

We hypothesize that there is an algorithm solving $\mathcal{P}_{(\mathrm{LPAL})}(I,(T,\mathcal{B}))$ in a time that is not dependent on $M\Delta$, which would imply that (LPAL) is FPT parameterized *only* by treewidth.

4 Experiments

We experimentally evaluated our algorithm on a set of algorithmically generated instances, measuring its running time and comparing the results against a heuristic based approach. Our implementation, including code to reproduce the experiments, is provided as supplementary material. In the ILP solving stage (stage 2) we used the state-of-the-art solver Gurobi [14].

Instance generation. The underlying graphs of our test instances are what we call *ring graphs*. For any $r \geq 1$ and $s \geq 2$, a ring graph is constructed by joining r *rings*, having s vertices each, using s *spokes* that meet in a central vertex. Formally we define $G \coloneqq (V,E)$ with $V \coloneqq \{(0,0)\} \cup \{(i,j) \colon 1 \leq i \leq r,\ 1 \leq j \leq s\}$ and

$$\begin{aligned} E \coloneqq &\{\{(i,j),(i+1,j)\} \colon 1 \leq i \leq r-1,\ 1 \leq j \leq s\} \\ &\cup \{\{(i,j),(i,j+1)\} \colon 1 \leq i \leq r,\ 1 \leq j \leq s-1\} \\ &\cup \{\{(i,s),(i,1)\} \colon 1 \leq i \leq r\} \cup \{\{(0,0),(1,j)\} \colon 1 \leq j \leq s\}. \end{aligned}$$

We evaluated our algorithm on ring graphs for various choices of r and s.

For each test instance we defined $c_{\mathrm{fix}} \coloneqq 50$ and for all edges $e \in E$ we set $c_e \coloneqq 5$ and $f_e^{\max} \coloneqq 20$.

We simulated the following simplified passenger behavior to obtain values for $f^{\min}$: Between each pair of vertices u and v we generate 50 passengers that want to travel between them. Each passenger wants to move on a shortest path between u and v. Hence we choose a random shortest path and count for each edge $e \in E$, how many passengers want to travel over it. This generates a passenger count d_e for each edge $e \in E$. Then we define the vehicle capacity $C \coloneqq (|V|-1)^2$ and finally $f_e^{\min} \coloneqq \lceil d_e/C \rceil$.

Heuristic algorithm. We compare the results of our algorithm, which chooses an optimal line concept from the set of all paths, against an algorithm which chooses an optimal line concept from a given line pool. The line pool is generated using the algorithm from [11]. Then the optimal line concept (restricted to this line pool) is determined by solving an integer program, where each line ℓ from the pool has its own frequency variable f_ℓ.

Results. We evaluated 27 test instances, with r ranging between 2 and 9, and s ranging between 3 and 6. The maximal treewidth of the considered instances was 5. For each instance we obtained $\mathrm{cost}_{\mathrm{o}}$, which is the cost of an optimal line concept resulting from our method, and $\mathrm{cost}_{\mathrm{h}}$, the cost of the line concept computed by the heuristic. We computed the average of the improvement ratio $\mathrm{cost}_{\mathrm{o}}/\mathrm{cost}_{\mathrm{h}}$, which is approximately 0.64. Thus our algorithm managed to reduce the costs by 36% on average.

We measured the running time of our algorithm, each of the three stages separately. It was run on a personal computer with an Intel Core i7-4790K CPU. The time taken by stage 3 was at most 1 second on all instances. The ILP construction (stage 1) took at most 24.1 seconds on instances with treewidth 5. The ILP solving (stage 2) took at most 97 seconds on instances with treewidth 5.

5 Conclusion and outlook

Line planning on all lines (LPAL) means allowing all simple paths as possible lines in a public transport supply. This large search space yields more options and, hence, better solutions for optimal public transport planning. After the mostly discouraging hardness results of [15], we now provided a fixed-parameter tractable algorithm that could be used to solve this problem in practice. This marks just the beginning of the parameterized study of (LPAL), since many questions remain open:

- When our algorithm is combined with algorithms for the later stages of public transport planning and applied to real-world datasets, how much does the quality of the results improve?
- Can the ILP we constructed be solved by an FPT-algorithm that is *not* parameterized by the degree nor $f^{\max}$?
- Can the runtime dependence on the treewidth k be reduced to a single-exponential of the form $\mathcal{O}(a^k)$ for some constant a?

Applying our approach to other formulations of line planning would also be interesting. For example, circular lines instead of paths could be considered, as in [17]. Another example would be replacing the fixed frequency bounds with a flow formulation that models passenger behavior [5].

References

1 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discret. Appl. Math.*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.

2 Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3417.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

4 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

5 Ralf Borndörfer, Martin Grötschel, and Marc E. Pfetsch. A column-generation approach to line planning in public transport. *Transp. Sci.*, 41(1):123–132, 2007. doi:10.1287/trsc.1060.0161.

6 Simon Bull, Jesper Larsen, Richard Martin Lusby, and Natalia J. Rezanova. Optimising the travel time of a line plan. *4OR*, 17(3):225–259, 2019. doi:10.1007/s10288-018-0391-5.

7 Michael R. Bussieck, Peter Kreuzer, and Uwe T. Zimmermann. Optimal lines for railway systems. *European Journal of Operational Research*, 96(1):54–63, 1997. doi:10.1016/0377-2217(95)00367-3.

8 M. T. Claessens, Nico M. van Dijk, and Peter J. Zwaneveld. Cost optimal allocation of rail passenger lines. *Eur. J. Oper. Res.*, 110(3):474–489, 1998. doi:10.1016/S0377-2217(97)00271-3.

9 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

10 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 815–821. AAAI Press, 2017. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14272.

11 Philine Gattermann, Jonas Harbering, and Anita Schöbel. Line pool generation. *Public Transp.*, 9(1-2):7–32, 2017. doi:10.1007/s12469-016-0127-x.

12 Marc Goerigk and Marie Schmidt. Line planning with user-optimal route choice. *Eur. J. Oper. Res.*, 259(2):424–436, 2017. doi:10.1016/j.ejor.2016.10.034.

13 Valérie Guihaire and Jin-Kao Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008. doi:10.1016/j.tra.2008.03.011.

14 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

15 Irene Heinrich, Philine Schiewe, and Constantin Seebach. Algorithms and hardness for non-pool-based line planning. In Mattia D'Emidio and Niels Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2022, September 8-9, 2022, Potsdam, Germany*, volume 106 of *OASIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASIcs.ATMOS.2022.8.

16 Ton Kloks. *Treewidth: computations and approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 1994. doi:10.1007/BFb0045375.

17 Berenike Masing, Niels Lindner, and Ralf Borndörfer. The price of symmetric line plans in the parametric city, 2022. arXiv:2201.09756.

18 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

19 Alexander Schiewe, Philine Schiewe, and Marie Schmidt. The line planning routing game. *Eur. J. Oper. Res.*, 274(2):560–573, 2019. doi:10.1016/j.ejor.2018.10.023.

20 Anita Schöbel. Line planning in public transportation: models and methods. *OR Spectr.*, 34(3):491–510, 2012. doi:10.1007/s00291-011-0251-6.

21 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 2005, September 14, 2005, Palma de Mallorca, Spain*, volume 2 of *OASIcs*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2005. URL: http://drops.dagstuhl.de/opus/volltexte/2006/660.

A Assembly algorithm analysis

Proof of Theorem 2. When Algorithm 1 is run on a nice tree decomposition of width k, at each node t we have $|B_t| \leq k+1$. In this analysis we treat k as a constant. Hence all considered path patterns have at most $k+1$ vertices and for each node t we can compute some encoding $\chi_t : \mathrm{Pat}(B_t) \to \mathbb{N}$ in constant time. A line concept L is implemented as a dictionary, where for any $p \in \mathrm{Pat}(B_t)$ we can look up exactly the sub-multiset of paths $\ell \in L$ having $\pi_{B_t}(\ell) = p$. These multisets are implemented as lists of tuples, each tuple consisting of a path and a number. Paths are implemented as linked lists of vertices.

The functions EXTEND and SUBDIVIDE can each be implemented to run in $\mathcal{O}(|\operatorname{supp} L'|)$, where L' is the input multiset of paths, by having pointers to the relevant vertices and using linked list insertion. Similarly the function JOIN can be implemented to run in $\mathcal{O}((|\operatorname{supp} L_1| + |\operatorname{supp} L_2|))$, where L_1 and L_2 are the input multisets, by changing around a constant number of pointers for each input path.

By treating k as a constant, the total number of nodes in the tree decomposition is in $\mathcal{O}(n)$. We will argue that at each node, the non-recursive part of ASSEMBLE only adds a constant number of new lines to the line concept and requires $\mathcal{O}(n)$ time. From this the claim follows.

It is crucial to first understand the line concept subtraction. When a line is removed completely from the line concept, we can simply detach the linked list representing the line, which takes constant time. When a line is removed partially, we have to copy the linked list, but this happens at most once per subtraction. Hence the running time of the subtraction

is in $\mathcal{O}(|\operatorname{supp} L'| + n)$, where L' is the subtracted multiset. Introduce nodes as well as join nodes make a constant number of subtractions. Since in total no more than the whole of L (respectively L_1 and L_2) can be subtracted, the total time taken by subtractions at one node is in $\mathcal{O}(n)$.

Let t be a forget node. Here no new lines are added, and in the code it looks like nothing at all is happening, but this is deceiving: Since B_t is different from $B_{t'}$, the encodings χ_t and $\chi_{t'}$ are also different – hence the dictionary data structure needs to be rebuilt entirely. This can be achieved in time in $\mathcal{O}(|\operatorname{supp} L|)$, which by induction is $\mathcal{O}(n)$.

Let t be an introduce node. The introduction operation adds at most a constant number of new lines. Extension and Subdivision both work by first subtracting a multiset L' of lines, transforming it into a multiset L'', then adding L'' back. We implement the subtraction such that all lines except one, i.e. $|\operatorname{supp} L'| - 1$ lines are removed completely from L. The transformation does not change the number of lines, i.e. $|\operatorname{supp} L''| = |\operatorname{supp} L'|$, thus a single iteration increases $|\operatorname{supp} L|$ by at most 1. Only a constant number of iterations take place, hence the desired upper bounds follow.

The same arguments show that join nodes only add a constant number of lines, taking linear time to do so. The line concept sums can be computed efficiently by concatenating a constant number of linked lists. ◀

B Correctness of the ILP

Proof of Theorem 3, claim 2. Let $L = (\mathcal{L}, f)$ be a feasible line concept with cost b.

Definition and feasibility of ψ

We define ψ by giving an assignment to every variable of the ILP. At the same time we show that this assignment is feasible.

For any $t \in T$ we use the shorthand $V_t \coloneqq V(G_t)$.

Then ψ assigns for each $t \in T$ and $p \in \operatorname{Pat}(B_t)$:

$$c_p^t \coloneqq \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p}} f(\ell).$$

Let t be a leaf node. Then $|V_t| = 1$, hence for all $\ell \in \mathcal{L}$ the filtered path $\ell|_{V_t}$ contains at most one vertex. Since path patterns need to have a length of at least two, for all $p \in \operatorname{Pat}(B_t)$ we have $\emptyset = \{\ell \in \mathcal{L} \colon \pi_{B_t}(\ell|_{V_t}) = p\}$ and hence $c_p^t = 0$.

Let t be a forget node which forgets $v \in B_{t'} \setminus B_t$. Then ψ assigns the f_e^t variables in the way which is required by the equality constraints:

$$\text{for each } e \in E^- \colon\ f_e^t \coloneqq \sum_{\substack{p' \in \operatorname{Pat}(B_{t'}) \\ e \text{ on } p'}} c_{p'}^{t'}$$

Consider any $\ell \in \mathcal{L}$ which visits v. Since t is forgetting v, all neighbors of v in G must have been introduced already, i.e. are contained in $V_{t'}$. Hence any edge $e \in E^-$ occurs on ℓ if and only if it occurs on $\ell|_{V_{t'}}$, if and only if it occurs on $\pi_{B_{t'}}(\ell|_{V_{t'}})$. Using this we see that f_e^t is the frequency exhibited by L:

$$\text{for each } e \in E^- \colon\ f_e^t = \sum_{\substack{p' \in \operatorname{Pat}(B_{t'}) \\ e \text{ on } p'}} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_{t'}}(\ell|_{V_{t'}}) = p'}} f(\ell)$$

$$= \sum_{\substack{\ell\in\mathcal{L}\\ e \text{ on } \pi_{B_{t'}}(\ell|_{V_{t'}})}} f(\ell)$$
$$= \sum_{\substack{\ell\in\mathcal{L}\\ e \text{ on } \ell}} f(\ell) \;=\; F_e^{(\mathcal{L},f)}.$$

Since L is a feasible line concept, it follows that the constraints on the f_e^t-variables are satisfied. There is one more set of constraints to verify, for each $p \in \mathrm{Pat}(B_t)$:

$$\sum_{\substack{p'\in\mathrm{Pat}(B_{t'})\\ \pi_{B_t}(p')=p}} c_{p'}^{t'} = \sum_{\substack{p'\in\mathrm{Pat}(B_{t'})\\ \pi_{B_t}(p')=p}} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_{t'}}(\ell|_{V_{t'}})=p'}} f(\ell)$$
$$= \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\pi_{B_{t'}}(\ell|_{V_{t'}}))=p}} f(\ell)$$
$$= \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) \quad = c_p^t,$$

using the facts $B_t \subseteq B_{t'}$ and $V_t = V_{t'}$.

Now let $t \in T$ be an introduce node which introduces $w \in B_t \setminus B_{t'}$. Then ψ assigns the instruction variables in the way which is required by the equality constraints. This immediately fulfills the non-negativity constraint. Since T is a tree decomposition, no neighbor of w can yet be forgotten, i.e. each neighbor is currently in the bag or coming later, hence contained in the set $B_t \cup (V \setminus V_t)$. Thus for all $\ell \in \mathcal{L}$ it holds: If w occurs on $\pi_{B_t}(\ell|_{V_t})$, then it is not adjacent to $\square$. Thus for all $p \in \mathrm{Pat}(B_t)$ containing w adjacent to $\square$ it holds:

$$c_p^t = \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) = 0,$$

as is required. Let $p \in \mathrm{Pat}(B_{t'})$ and consider some $\ell \in \mathcal{L}$ with $\pi_{B_{t'}}(\ell|_{V_{t'}}) = p$. If w does not occur on ℓ, then $\pi_{B_t}(\ell|_{V_t}) = p$. Otherwise $\ell|_{V_t}$ can be created from $\ell|_{V_{t'}}$ by applying an extension or subdivision operation. Therefore we have:

$$c_p^{t'} = \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_{t'}}(\ell|_{V_{t'}})=p}} f(\ell)$$
$$= \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) + \sum_{p\xrightarrow{(u,wu)}p'} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p'}} f(\ell) + \sum_{p\xrightarrow{(uv,uwv)}p'} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p'}} f(\ell)$$
$$= c_p^t + \sum_{p\xrightarrow{(u,wu)}p'} c_{p'}^t + \sum_{p\xrightarrow{(uv,uwv)}p'} c_{p'}^t,$$

which was the last set of constraints to check for this type of node.

Finally let $t \in T$ be a join node with children t_1 and t_2. It holds: $B_t = B_{t_1} = B_{t_2}$. For each $p_1 \in \mathrm{Pat}(B_{t_1}), p_2 \in \mathrm{Pat}(B_{t_2})$ with $(p_1, p_2) \xrightarrow{B_t} p$ the function ψ assigns the following non-negative value:

$$j_{p_1,p_2}^t := \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1\\ \pi_{B_t}(\ell|_{V_{t_2}})=p_2}} f(\ell).$$

We first check the operation applicability constraints. Let $p_1 \in \mathrm{Pat}(B_t)$. We have

$$\begin{aligned}\sum_{\substack{p_2,p'\in\mathrm{Pat}(B_t)\\(p_1,p_2)\xrightarrow{B_t}p'}} j^t_{p_1,p_2} &= \sum_{\substack{p_2,p'\in\mathrm{Pat}(B_t)\\(p_1,p_2)\xrightarrow{B_t}p'}} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1\\ \pi_{B_t}(\ell|_{V_{t_2}})=p_2}} f(\ell)\\ &= \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1}} \sum_{\substack{p'\in\mathrm{Pat}(B_t)\\(p_1,\pi_{B_t}(\ell|_{V_{t_2}}))\xrightarrow{B_t}p'}} f(\ell)\\ &\le \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1}} f(\ell) \;=\; c^{t_1}_{p_1},\end{aligned}$$

where we use the fact that when p_1 and p_2 are given, there exists at most one p' with $(p_1,p_2)\xrightarrow{B_t}p'$. Symmetrically we can verify for each $p_2 \in \mathrm{Pat}(B_t)$:

$$\sum_{\substack{p_1,p'\in\mathrm{Pat}(B_t)\\(p_1,p_2)\xrightarrow{B_t}p'}} j^t_{p_1,p} \;\le\; c^{t_2}_{p_2}.$$

Define $\mathcal{L}^\circ := \{\ell \in \mathcal{L}\colon (\pi_{B_t}(\ell|_{V_{t_1}}), \pi_{B_t}(\ell|_{V_{t_2}})) \xrightarrow{B_t} \pi_{B_t}(\ell|_{V_t})\}$, i.e. the subset of lines which could possibly result from a join operation at t. We claim that the following holds for all $\ell \in \mathcal{L}\setminus\mathcal{L}^\circ$ and $p \in \mathrm{Pat}(B_t)$:

$$\pi_{B_t}(\ell|_{V_t}) = p \iff \pi_{B_t}(\ell|_{V_{t_1}}) = p \;\oplus\; \pi_{B_t}(\ell|_{V_{t_2}}) = p,$$

where $\oplus$ denotes *exclusive or*. Let $p \in \mathrm{Pat}(B_t)$. It holds:

$$\begin{aligned}&c^t_p - c^{t_1}_p - c^{t_2}_p\\ &= \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_{t_1}})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell)\\ &= \sum_{\substack{\ell\in\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_{t_1}})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell)\\ &\quad + \sum_{\substack{\ell\in\mathcal{L}\setminus\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}\setminus\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_{t_1}})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}\setminus\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell)\\ &= \sum_{\substack{\ell\in\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_{t_1}})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}^\circ\\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell).\end{aligned}$$

We also obtain:

$$\begin{aligned}&\sum_{(p_1,p_2)\xrightarrow{B_t}p} j^t_{p_1,p_2} - \sum_{(p,p_2)\xrightarrow{B_t}p'} j^t_{p,p_2} - \sum_{(p_1,p)\xrightarrow{B_t}p'} j^t_{p_1,p}\\ &= \sum_{(p_1,p_2)\xrightarrow{B_t}p} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p\\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1\\ \pi_{B_t}(\ell|_{V_{t_2}})=p_2}} f(\ell) - \sum_{(p,p_2)\xrightarrow{B_t}p'} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p'\\ \pi_{B_t}(\ell|_{V_{t_1}})=p\\ \pi_{B_t}(\ell|_{V_{t_2}})=p_2}} f(\ell) - \sum_{(p_1,p)\xrightarrow{B_t}p'} \sum_{\substack{\ell\in\mathcal{L}\\ \pi_{B_t}(\ell|_{V_t})=p'\\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1\\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell)\end{aligned}$$

$$= \sum_{\substack{\ell\in\mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_1}})=p}} f(\ell) - \sum_{\substack{\ell\in\mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell).$$

Hence

$$c_p^t - c_p^{t_1} - c_p^{t_2} = \sum_{(p_1,p_2)\xrightarrow{B_t}p} j_{p_1,p_2}^t - \sum_{(p,p_2)\xrightarrow{B_t}p'} j_{p,p_2}^t - \sum_{(p_1,p)\xrightarrow{B_t}p'} j_{p_1,p}^t,$$

meaning that the constraints for this node are satisfied.

Cost equivalence of ψ

Now we want to show that the cost of this assignment ψ is equal to the cost of $(\mathcal{L}, f)$. It holds:

$$\text{cost}((\mathcal{L}, f)) = c_{\text{fix}} \sum_{\ell\in\mathcal{L}} f(\ell) + \sum_{\ell\in\mathcal{L}} \sum_{e\in E(\ell)} c_e f(\ell).$$

Each $e \in E$ occurs exactly once in the set of forgotten edges E^- for some forget node t. As argued before, we have

$$f_e^t = \sum_{\substack{\ell\in\mathcal{L} \\ e \text{ on } \ell}} f(\ell),$$

and since $\text{cost}(f_e^t) = c_e$, these variables account for the second term of $\text{cost}((\mathcal{L}, f))$.

For each $t \in T$ we define θ^t to be the total cost caused by all i- and j-variables which belong to t or descendants of t. We claim that, under the defined assignment of ψ, it holds:

$$\theta^t = c_{\text{fix}} \sum_{\substack{\ell\in\mathcal{L} \\ |\ell|_{V_t}|\geq 2}} f(\ell),$$

which we will prove by induction:

For leaf nodes t we clearly have

$$\theta^t = 0 = c_{\text{fix}} \sum_{\substack{\ell\in\mathcal{L} \\ |\ell|_{V_t}|\geq 2}} f(\ell).$$

Let t be a forget node with child t'. It holds that $V_t = V_{t'}$ and $\theta^t = \theta^{t'}$ since here no costs for i- or j-variables are added. The equality follows.

Let t be an introduce node that introduces w, with child t'. It holds:

$$\begin{aligned}
\sum_{\substack{\ell\in\mathcal{L} \\ |\ell|_{V_t}|\geq 2}} f(\ell) &= \sum_{\substack{\ell\in\mathcal{L} \\ w \text{ on } \ell \\ |\ell|_{V_t}|=2}} f(\ell) + \sum_{\substack{\ell\in\mathcal{L} \\ w \text{ on } \ell \\ |\ell|_{V_t}|>2}} f(\ell) + \sum_{\substack{\ell\in\mathcal{L} \\ w \text{ not on } \ell \\ |\ell|_{V_t}|\geq 2}} f(\ell) \\
&= \sum_{\substack{\ell\in\mathcal{L},\, v\in B_{t'} \\ \pi_{B_t}(\ell|_{V_t})=wv}} f(\ell) + \sum_{\substack{\ell\in\mathcal{L} \\ w \text{ on } \ell \\ |\ell|_{V_{t'}}|>1}} f(\ell) + \sum_{\substack{\ell\in\mathcal{L} \\ w \text{ not on } \ell \\ |\ell|_{V_{t'}}|\geq 2}} f(\ell) \\
&= \sum_{\substack{\ell\in\mathcal{L},\, v\in B_{t'} \\ \pi_{B_t}(\ell|_{V_t})=wv}} f(\ell) + \sum_{\substack{\ell\in\mathcal{L} \\ |\ell|_{V_{t'}}|\geq 2}} f(\ell).
\end{aligned}$$

Therefore:

$$\begin{aligned}\theta^t &= \theta^{t'} + c_{\text{fix}} \sum_{v \in B_{t'}} c^t_{wv} = \theta^{t'} + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L},\, v \in B_{t'} \\ \pi_{B_t}(\ell|_{V_t}) = wv}} f(\ell) \\ &= \theta^{t'} + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t'}}| \geq 2}} f(\ell) = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell).\end{aligned}$$

Let t be a join node with children t_1 and t_2. Let $\ell \in \mathcal{L}^\circ$. Then $|\ell|_{V_t}| \geq 2$, $\left|\ell|_{V_{t_1}}\right| \geq 2$ and $\left|\ell|_{V_{t_2}}\right| \geq 2$. Consider on the other hand $\ell \in \mathcal{L} \setminus \mathcal{L}^\circ$. If $|\ell|_{V_t}| \geq 2$ holds, then either $\left|\ell|_{V_{t_1}}\right| \geq 2$ or $\left|\ell|_{V_{t_2}}\right| \geq 2$, but not both. It follows:

$$\begin{aligned}\theta^t &= \theta^{t_1} + \theta^{t_2} - c_{\text{fix}} \sum_{(p_1,p_2) \xrightarrow{B_t} p} j^t_{p_1,p_2} \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ \left|\ell|_{V_{t_1}}\right| \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ \left|\ell|_{V_{t_2}}\right| \geq 2}} f(\ell) - c_{\text{fix}} \sum_{(p_1,p_2) \xrightarrow{B_t} p} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1 \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p_2}} f(\ell) \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ \left|\ell|_{V_{t_1}}\right| \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ \left|\ell|_{V_{t_2}}\right| \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \left|\ell|_{V_{t_1}}\right| \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \left|\ell|_{V_{t_2}}\right| \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ \left|\ell|_{V_{t_1}}\right| \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ \left|\ell|_{V_{t_2}}\right| \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) \\ &= 2c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ |\ell|_{V_t}| \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) \\ &= c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ |\ell|_{V_t}| \geq 2}} f(\ell) \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell).\end{aligned}$$

This concludes the proof that for all $t \in T$:

$$\theta^t = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell).$$

For the root r of T we have $V_r = V$, hence:

$$\theta^r = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell| \geq 2}} f(\ell) = c_{\text{fix}} \sum_{\ell \in \mathcal{L}} f(\ell),$$

so the first term of $\text{cost}((\mathcal{L}, f))$ is correctly accounted for as well. ◀