Visualization of Event Graphs for Train Schedules

Johann Hartleb[1], Marie Schmidt[2], Samuel Wolf[2], and Alexander Wolff[2]

1 **DB InfraGO AG, Germany**
johann.hartleb@deutschebahn.com
2 **Universität Würzburg, Germany**
firstname.lastname@uni-wuerzburg.de

Abstract

Software that is used to compute or adjust train schedules is based on so-called *event graphs*. The vertices of such a graph correspond to *events*; each event is associated with a point in time, a location, and a train. A *train line* corresponds to a sequence of events (ordered by time) that are associated with the same train. The event graph has a directed edge from an earlier to a later event if they are consecutive along a train line. Events that occur at the same location do not occur at the same time.

In this paper, we present a way to visualize such graphs, namely *time-space diagrams*. A time-space diagram is a straight-line drawing of the event graph with the additional constraint that all vertices that belong to the same location lie on the same horizontal line and that the x-coordinate of each vertex is given by its point in time. Hence, it remains to determine the y-coordinates of the locations. A good drawing of a time-space diagram supports users (or software developers) when creating (software for computing) train schedules.

To enhance readability, we aim to minimize the number of turns in time-space diagrams. To this end, we establish a connection between this problem and Maximum Betweenness. Then we develop exact reduction rules to reduce the instance size. We also propose a parameterized algorithm and devise a heuristic that we evaluate experimentally on a real-world dataset.

Related Version *Full version:* https://arxiv.org/abs/2503.01808

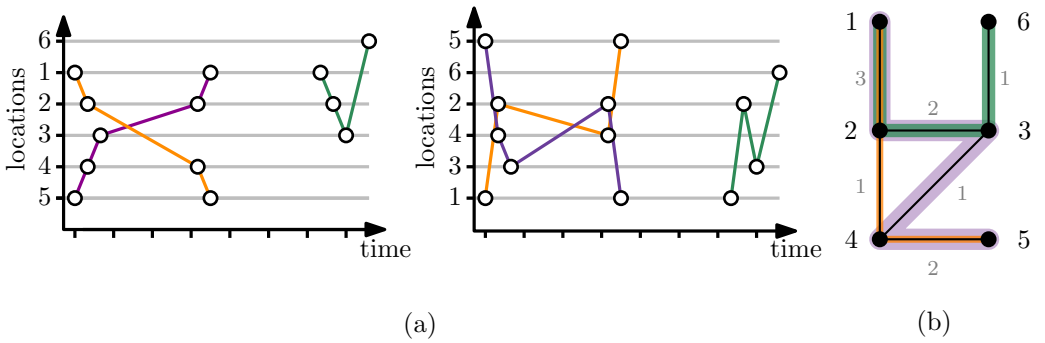
1 Introduction

Train schedules are subject to constant changes due to interferences such as temporary infrastructure malfunctions or congestions due to high traffic volume. As a consequence, train schedules must be adjusted in real-time to remedy the disturbances via rerouting and other means. In recent years, the automation of this process has gained track. DB InfraGO AG, a subsidiary of Deutsche Bahn AG, is developing approaches based on a so-called *event graph* [5] as an underlying structure that encodes the necessary information to (re-)compute a train schedule. An event graph models trains running on specific routes on an infrastructure via events. For the further automation and for real-time human intervention, it is important that the event graph can be easily read by humans. For this purpose, we propose a drawing style and algorithms that aim to produce comprehensible drawings of the event graph.

▶ **Definition 1** (Event Graph)**.** An *event graph* $\mathcal{E}$ is a directed graph. Let $V(\mathcal{E})$ denote the vertex set of $\mathcal{E}$. Each vertex v of $\mathcal{E}$, called *event*, is associated with a location $\ell(v)$, a positive integer $\text{train}(v)$, and a point of time $t(v)$ when the event is scheduled. For two different events u and w, if $t(u) = t(w)$, then $\text{train}(u) \neq \text{train}(w)$ and $\ell(u) \neq \ell(w)$. There is an arc (u, w) in $\mathcal{E}$ if (i) $\text{train}(u) = \text{train}(w)$, (ii) $t(u) < t(w)$, and (iii) there is no event v with $\text{train}(v) = \text{train}(u)$ and $t(u) < t(v) < t(w)$.

For a train z, we call the sequence $v_1, \ldots, v_j$ of all events with $\text{train}(v_1) = \cdots = \text{train}(v_j) = z$ ordered by $t(\cdot)$ the *train line* of train z. We propose to visualize event graphs as follows.

41st European Workshop on Computational Geometry, Liblice, Czech Republic, April 9–11, 2025.
This is an extended abstract of a presentation given at EuroCG'25. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.



(a) (b)

Figure 1 (a) Two different time-space diagrams of the same event graph $\mathcal{E}$ with locations $\{1, \dots, 6\}$. (b) The location graph of $\mathcal{E}$; the colored paths are the train lines, the gray numbers the weights.

▶ **Definition 2** (Time-Space Diagram). Let $\mathcal{E}$ be an event graph, let $Y = |\ell(V(\mathcal{E}))|$, and let $y\colon \ell(V(\mathcal{E})) \to \{1, 2, \dots, Y\}$ be a bijection. The *time-space diagram* induced by y is the straight-line drawing of $\mathcal{E}$ in the plane where event v is mapped to the point $(t(v), y(\ell(v))$.

In a time-space diagram (see Figure 1a for two examples), we call $y(p)$ the *level* of location p. Given a drawing Γ of an event graph $\mathcal{E}$ and three consecutive events of a train line in $\mathcal{E}$ with pairwise distinct locations p, q, r, we say that there is a *turn* in Γ if the level of q is smaller/larger than the levels of p and r.[1] Experiments suggest that minimizing a classical graph drawing objective, the number of crossings, often does not yield comprehensible drawings. Instead, minimizing the number of turns in a drawing seems to be a promising objective, as Figure 2 illustrates. Therefore, we consider the following problem.

▶ **Problem 3.** *Let $\mathcal{E}$ be an event graph. Find a time-space diagram Γ of $\mathcal{E}$ that minimizes the number of turns along the train lines in Γ.*

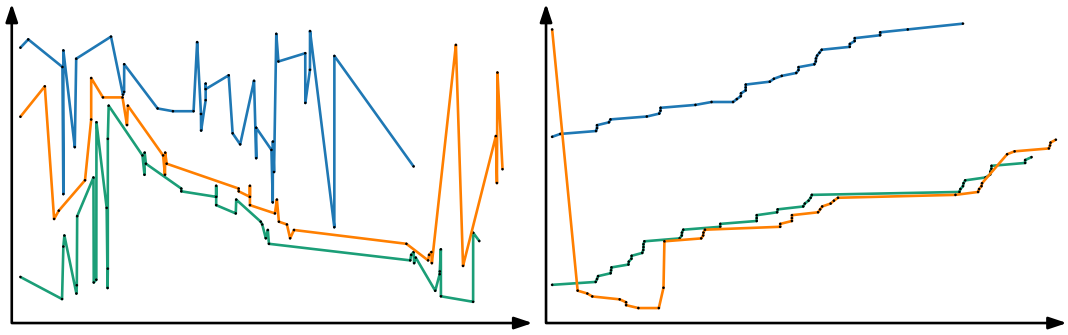


Figure 2 Two time-space diagrams of the same event graph. Left: A crossing-minimal drawing with zero crossings (and 71 turns). Right: A turn-minimal drawing with one turn (and five crossings).

Note that the numbers of turns in a time-space diagram is determined solely by the function y, which represents an ordering of the locations. Therefore, Problem 1.3 is closely related to the following problem.

[1] Note that this definition does not consider the case where consecutive events have the same location. It is easy to see, however, that we can normalize the graph such that consecutive events always have different locations without changing the optimal solution of Problem 1.3.

▶ **Problem 4** (Maximum Betweenness). *Let S be a finite set, and let $R \subseteq S \times S \times S$ be a finite set of ordered triplets called* constraints*. A total order $\prec$* satisfies *a constraint $(a, b, c) \in R$ if either $a \prec b \prec c$ or $c \prec b \prec a$ holds. Find a total order that maximizes the number of satisfied constraints.*

Note that there is a one-to-one correspondance between (optimal) solutions of Problems 1.3 and 1.4. Maximum Betweenness is NP-hard [10], which implies the NP-hardness of Problem 1.3. Maximum Betweenness has been studied extensively [4,11–13]. In particular, it admits 1/2-approximation algorithms [2,9], but for any $\varepsilon > 0$ it is NP-hard to compute a $(1/2 + \varepsilon)$-approximation [1]. Note that, due to the different objectives, these (non-) approximability results do not carry over to Problem 1.3; see [8] for details.

When translating to Maximum Betweenness, we lose information about the train lines. However, this information proves to be beneficial for our purposes. In particular, we want to leverage the natural sparseness of train infrastructures. We define two auxiliary graphs that capture the connections between locations in $\mathcal{E}$, which we use in our algorithms.

▶ **Definition 5** (Location Graph). Let $\mathcal{E}$ be an event graph. The *location graph* $\mathcal{L}$ of $\mathcal{E}$ is an undirected weighted graph whose vertices are the locations of $\mathcal{E}$. For two locations $p \neq q$, the weight $w(\{p, q\})$ of the edge $\{p, q\}$ in $\mathcal{L}$ corresponds to the number of arcs (u, v) or (v, u) in the event graph $\mathcal{E}$ such that $\ell(u) = p$ and $\ell(v) = q$ and $\text{train}(u) = \text{train}(v)$. If $w(\{p, q\}) = 0$, then p and q are not adjacent in $\mathcal{L}$.

See Figure 1b for an example. Note that the train line of a train z in the event graph corresponds to a walk (a not necessarily simple path) in the location graph. Abusing the notation of a train line, we also call this path in the location graph a *train line* of z.

▶ **Definition 6** (Augmented Location Graph). Let $\mathcal{E}$ be an event graph. The *augmented location graph* $\mathcal{L}'$ of $\mathcal{E}$ is a supergraph of the location graph $\mathcal{L}$ of $\mathcal{E}$ containing additional edges $\{\ell(v_{i-1}), \ell(v_{i+1})\}$ for each consecutive triplet (v_{i-1}, v_i, v_{i+1}) of a train line in $\mathcal{E}$.

The augmented graph $\mathcal{L}'$ has the crucial property that three consecutive events v_{i-1}, v_i, v_{i+1} of a train line whose locations can cause a potential turn form a triangle in $\mathcal{L}'$.

2 Exact Algorithms

An exact reduction rule. There are substructures that can be solved easily and independently. Consider the location graph $\mathcal{L}$ of an event graph $\mathcal{E}$. We call a vertex $p \in V(\mathcal{L})$ a *terminal* if a train starts or ends at p, and we say that a path in $\mathcal{L}$ is a *chain* if each of its vertices has degree exactly 2 in $\mathcal{L}$ and the path cannot be extended without violating this property. If a chain contains no terminals, and trains move through the entire chain without turning back, then there is always a turn-minimal drawing of $\mathcal{E}$ that contains no turn along the chain. This is due to the fact that any turn on the chain can be moved to a non-chain vertex adjacent to one of the chain endpoints.

We now sketch a generalization of this intuition. We call two vertices whose removal disconnects a graph a *separating pair*. Let $\{s, t\}$ be a separating pair of $\mathcal{L}$, let C be a connected component of $\mathcal{L} - \{s, t\}$, and let $C' = \mathcal{L}[V(C) \cup \{s, t\}]$. We call C a *transit component* if C does not contain any terminal, and if the trains passing through C' via s also pass through t (before possibly passing through s again).

A transit component C is *contractible* if, for each path associated to a train line in C, we can assign a direction such that the resulting directed graph is acyclic; see in Figure 3.

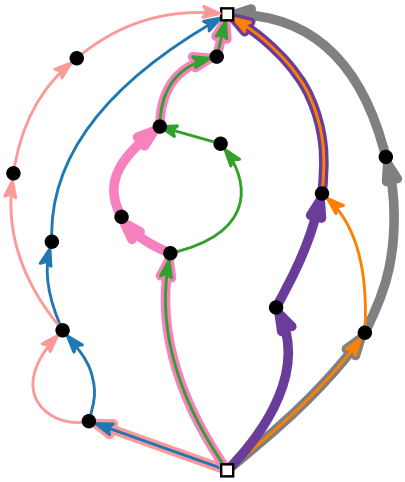


Figure 3 A transit component that is part of a larger location graph. Each train is represented by a sequence of arcs of the same color. Some trains share edges. The component is contractible with respect to the set of trains.

▶ **Reduction Rule 1** (Transit Component Contraction)**.** Let $\mathcal{E}$ be an event graph, let $\mathcal{L}$ be its location graph, and let C be a contractible transit component that is separated by $\{s, t\}$. For each train z that traverses C, replace in $\mathcal{E}$ the part of the train line of z between the events that correspond to s and t by the arc (directed according to time) that connects the two events.

▶ **Theorem 7.** *Let $\mathcal{E}$ be an event graph. If $\mathcal{E}'$ is an event graph that results from applying Reduction Rule 1 to $\mathcal{E}$, then a turn-optimal drawing of $\mathcal{E}$ and a turn-optimal of $\mathcal{E}'$ have the same number of turns.*

Proof sketch. Let Γ be a turn-optimal drawing of $\mathcal{E}$, and let Γ' be a turn-optimal drawing of $\mathcal{E}$ after applying Reduction Rule 1 once; to a contractible transit component C. We now turn Γ' into a drawing of $\mathcal{E}$ without introducing new turns. We expand the mapping y' corresponding to Γ' by placing C between the levels of $y'(s)$ and $y'(t)$ such that the ordering of the locations in C respects a topological ordering. Due to the topological ordering, C contains no turns, whereas turns at s and t are preserved.

Conversely, we can transform Γ into a drawing of $\mathcal{E}'$ without changing the number of turns. Since C contains no terminal, we can move any turn in C to one of the two locations s and t that separate C from the rest of the location graph, and then replace every train line that traverses C by a straight-line segment connecting the two events that correspond to s and t (on that train line). For a complete proof, see [8]. ◀

A parameterized algorithm. We use dynamic programming on a nice tree decomposition of the augmented location graph $\mathcal{L}'$. For a definition of a (nice) tree decomposition, see [3]. Since three consecutive locations of a train line form a triangle in $\mathcal{L}'$, there must be a bag that contains all three locations. Hence, it suffices to check each bag for potential turns in order to find a drawing with the minimum number of turns. For the full proof, see [8].

▶ **Theorem 8.** *Let $\mathcal{E}$ be an event graph, and let $\mathcal{L}'$ be its augmented location graph. Computing a turn-optimal time-space diagram of $\mathcal{E}$ is fixed-parameter tractable with respect to the treewidth of $\mathcal{L}'$.*

3 A Greedy Heuristic Approach

The heuristic tries to minimize the number of turns by iteratively inserting trains into an ordering of locations such that the inserted train does not violate the existing ordering and such that its train line is as monotone as possible. The idea is as follows.

Given an event graph $\mathcal{E}$ and its associated location graph $\mathcal{L}$ with train lines $Z = \{z_1, \ldots, z_k\}$, we sort Z with respect to the total weight of the edges of the train lines in $\mathcal{L}$, in descending order.

For simplicity, we assume that each train line is a simple path in $\mathcal{L}$. If this is not the case, we decompose each non-simple train line into multiple simple train lines. We construct a directed auxiliary graph G that is initially empty and to which we iteratively insert the edges of each train line until we obtain a directed version of $\mathcal{L}$. For a directed graph (or simply a set of arcs) H, let $E(H)$ be the set of undirected edges that correspond to the arcs of H.

Let $i \in \{1, \ldots, k\}$ the index of the current iteration. For each connected component P of $E(z_i) \setminus E(G)$, we do the following. Note that P is adjacent to at most two vertices of G. If P is adjacent to *exactly* two vertices p and q in G, we test whether there is a q–p path in G. If this is the case, we select among the edges of P one that has the smallest weight in $\mathcal{L}$. We reverse the selected edge. Then we insert the vertices and the directed edges of P into G. (The reversal ensures that G remains acyclic.)

After the last iteration, we compute a topological order π of G and draw the time-space diagram of $\mathcal{E}$ induced by π. See [8] for the pseudocode of the heuristic.

4 Experimental Analysis

We test the effectiveness of the reduction rule and of the heuristic on an anonymized and perturbed dataset with 19 instances provided by DB InfraGO AG; see Figure 4 for an overview of the dataset. The result of each experiment is the average value over 25 repetitions of the experiment. We implemented our algorithms in the programming language `Python`. We used Networkx [7] to handle most of the graph operations and Gurobi [6] to solve the integer linear program. All experiments were conducted on a workstation running Fedora 40 with Kernel 6.10.6 using an Intel-7-8850U CPU with 16GB RAM.

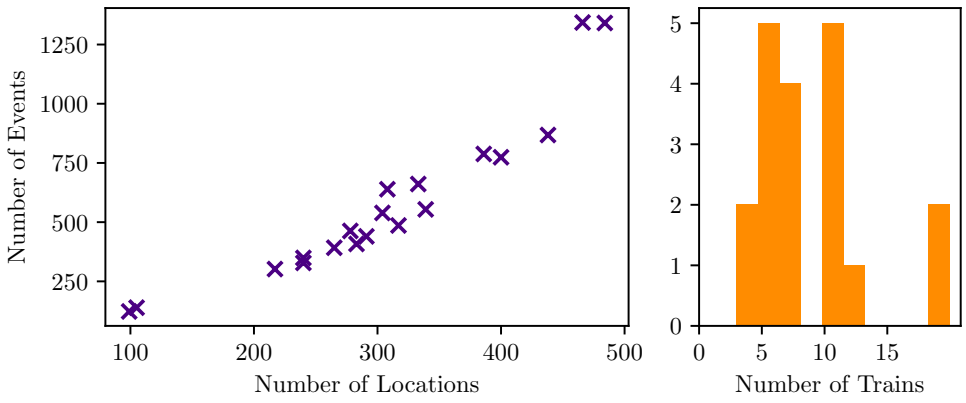


Figure 4 Left: Instances with respect to the number of events and the number of locations. Right: The histogram depicts the frequency (y-axis) of the number of trains (x-axis) in the dataset, i.e., there are 5 instances containing 5 trains.

Effectiveness of the reduction rule. Our implementation of Reduction Rule 1 is restricted to exhaustively contract chains. But even with this restriction, the reduction rule proves to be effective on the provided dataset. On average, the number of locations was reduced by 75%, where the best result was a reduction by 89% (instance 50_4) and the worst result was a reduction by 55% (instance 20_3). A full evaluation is shown in Figure 5.

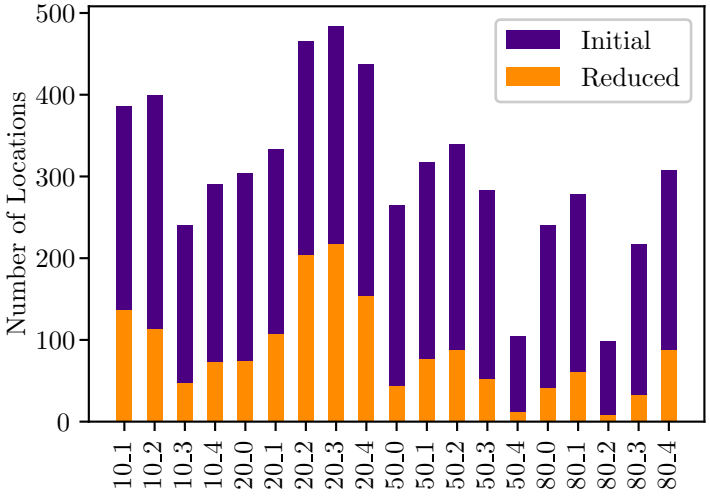


Figure 5 Results of the effectiveness of applying contractions, restricted to contracting chains. The bar diagram shows the number of locations in each instance before and after contraction.

Effectiveness of the heuristic. We compare the results of our heuristic to the corresponding optima, which we obtained using the integer linear program (ILP) described in [8], and to the heuristic EM-algorithm that Filipović, Kartelj, and Matić [4] developed for the Maximum Betweenness problem. We use the implementation of the EM-algorithm of Filipović et al. [4] and their parameter settings. On average, our algorithm yields results that are four times better than those of the EM-algorithm on the original instances. On the reduced instances, our algorithm loses its advantage; there, the two approaches perform comparably on average. However, our approach is significantly faster, as our algorithm takes 0.2 s on the slowest instance while the EM-algorithm takes 26.3 s. See Figures 6 and 7 for a detailed overview of the effectiveness and runtime, respectively. Note that we show the runtime of the ILP approach only on the reduced instances since the ILP did not converge on most of the original instances within a time limit of one hour.

5 Conclusion and Future Work

In this work we have introduced the problem of computing turn-minimal time-space diagrams to visualize event graphs. We have established a connection between this problem and Maximum Betweenness, we have presented an FPT-algorithm parameterized by the treewidth of the location graph, and we have proposed a heuristic for solving large instances.

The heuristic produces drawings that are still far from optimal, thus it would be interesting to improve this heuristic or to find a better one. As part of our ongoing research, we have developed a new exact integer linear programming approach that shows promising preliminary results, solving the largest real-world instance in less than a second.

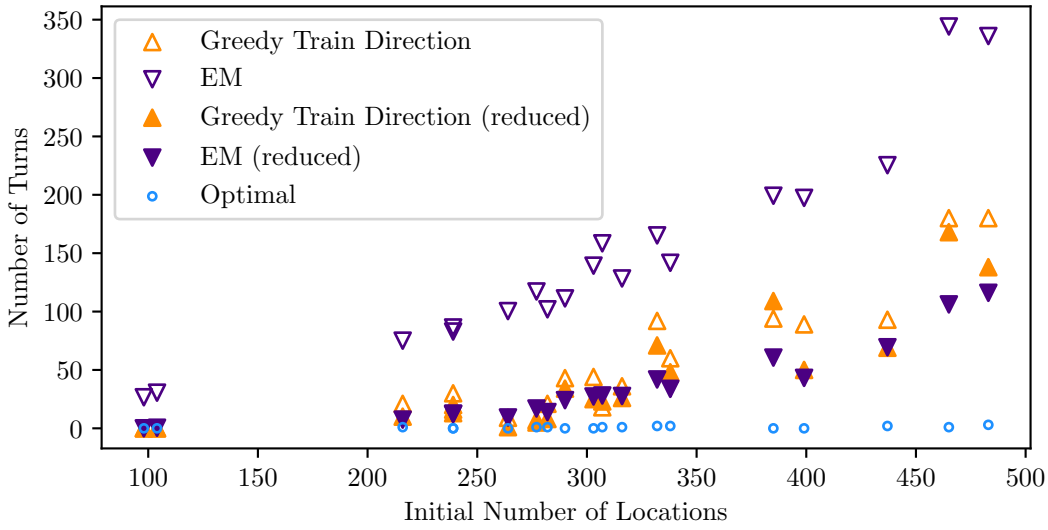


Figure 6 Comparison of the effectiveness between our heuristic (Greedy Train Direction), the EM-algorithm of Filipović et al. [4], and the optimum; on the original and the reduced instances.

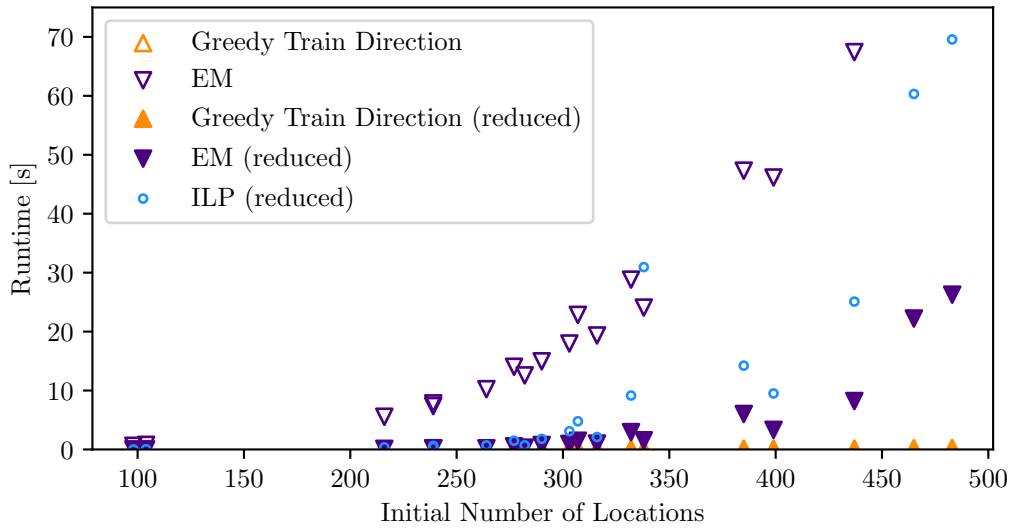


Figure 7 Runtime comparison between our heuristic (Greedy Train Direction) and the EM-algorithm of Filipović et al. [4] on the original and the reduced instances.

References

1 Per Austrin, Rajsekar Manokaran, and Cenny Wenner. On the NP-hardness of approximating ordering-constraint satisfaction problems. *Theory of Computing*, 11(10):257–283, 2015. doi:10.4086/toc.2015.v011a010.

2 Benny Chor and Madhu Sudan. A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, 11(4):511–523, 1998. doi:10.1137/S0895480195296221.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015. doi:10.5555/2815661.

4 Vladimir Filipović, Aleksandar Kartelj, and Dragan Matić. An electromagnetism metaheuristic for solving the maximum betweenness problem. *Applied Soft Computing*, 13(2):1303–1313, 2013. doi:10.1016/j.asoc.2012.10.015.

5 Rihab Gorsane, Khalil Gorsan Mestiri, Daniel Tapia Martinez, Vincent Coyette, Beyrem Makhlouf, Gereon Vienken, Minh Tri Truong, Andreas Söhlke, Johann Hartleb, Amine Kerkeni, Irene Sturm, and Michael Küpper. Reinforcement learning based train rescheduling on event graphs. In *26th Int. Conf. Intell. Transport. Syst. (ITSC)*, pages 874–879. IEEE, 2023. doi:10.1109/ITSC57777.2023.10422531.

6 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

7 Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. In *7th Python Sci. Conf.*, pages 11–15. SciPy, 2008. doi:10.25080/TCWV9851.

8 Johann Hartleb, Marie Schmidt, Samuel Wolf, and Alexander Wolff. Visualization of event graphs for train schedules. arXiv report, 2025. URL: https://arxiv.org/abs/2503.01808.

9 Yury Makarychev. Simple linear time approximation algorithm for betweenness. *Operations Research Letters*, 40(6):450–452, 2012. doi:10.1016/j.orl.2012.08.008.

10 Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979. doi:10.1137/0208008.

11 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

12 Aleksandar Savić. On solving the maximum betweenness problem using genetic algorithms. *Serdica Journal of Computing*, 3(3):299–308, 2009. doi:10.55630/sjc.2009.3.299-308.

13 Aleksandar Savić, Jozef Kratica, Marija Milanović, and Djordje Dugošija. A mixed integer linear programming formulation of the maximum betweenness problem. *European Journal of Operational Research*, 206(3):522–527, 2010. doi:10.1016/j.ejor.2010.02.028.