On Recognizing Staircase Compatibility

Andreas Bärmann\(^1\), Patrick Gemander\(^2\), Alexander Martin\(^1\) and Maximilian Merkert\(^3\)

\(^1\)Andreas.Baermann@fau.de
Andreas.Baermann@fau.de
Lehrstuhl für Wirtschaftsmathematik,
Department Mathematik,
Friedrich-Alexander-Universität Erlangen-Nürnberg,
Cauerstraße 11, 91058 Erlangen, Germany

\(^2\)Patrick.Gemander@fau.de
Group Optimization
Fraunhofer-Arbeitsgruppe für Supply Chain Services SCS,
Fraunhofer-Institut für Integrierte Schaltungen IIS,
Nordostpark 93, 90411 Nürnberg, Germany

\(^3\)Maximilian.Merkert@ovgu.de
Institut für Mathematische Optimierung,
Fakultät für Mathematik,
Otto-von-Guericke-Universität Magdeburg,
Universitätsplatz 2, 39106 Magdeburg, Germany

Abstract

For the problem to find an \(m\)-clique in an \(m\)-partite graph, staircase compatibility has recently been introduced as a polynomial-time solvable special case. It is a property of a graph together with an \(m\)-partition of the vertex set and total orders on each subset of the partition. In optimization problems involving \(m\)-cliques in \(m\)-partite graphs as a subproblem, it allows for totally unimodular linear programming formulations which have shown to efficiently solve problems from different applications. In this work, we address questions concerning the recognizability of this property in the case where the \(m\)-partition of the graph is given, but suitable total orders are to be determined. While finding these total orders is NP-hard in general, we give several conditions under which it can be done in polynomial time. For bipartite graphs, we present a polynomial-time algorithm to recognize staircase compatibility, and show that staircase total orders are unique up to a small set of reordering operations. On \(m\)-partite graphs, where the recognition problem is NP-complete in the general case, we identify a polynomially solvable subcase and also provide a corresponding algorithm to compute the total orders. Finally, we evaluate the performance of our ordering algorithm for \(m\)-partite graphs on a set of artificial instances as well as real-world instances from a railway timetabling application. It turns out that applying the ordering algorithm to the real-world instances and subsequently solving the problem via the aforementioned totally unimodular reformulations indeed outperforms a generic formulation which does not exploit staircase compatibility.

Keywords: Staircase Structure, Clique Problem, Multiple-Choice Constraints, Scheduling

Mathematics Subject Classification: 90C27 - 90C35 - 90C57 - 90C90
1 Introduction

The clique problem is among the most prominent combinatorial optimization problems studied in the literature. The interest in this problem is closely related to its importance for the modelling of applied optimization tasks, such as e.g. scheduling problems. In order to solve practical problems involving cliques efficiently, it is usually important to exploit the structure of the underlying graph as far as possible. Being able to recognize special cases of the problem from the incidence structure of the graph can beneficially inform both the choice of the optimization model used and the derivation of fast solution algorithms.

In this work, we focus on recognizing a polynomially solvable subcase of the problem to find an $m$-clique in an $m$-partite graph. This special case is called staircase compatibility and allows for totally unimodular linear programming formulations of polynomial size, see [BGMS18]. In particular, the formulation based on a dual network flow problem introduced there has shown to be very efficient for solving problems arising in different applications, such as railway timetabling ([BMS20]) and flow problems with piecewise-linear arc costs ([LM16]). In [JM18], the authors present a new formulation for the vertex colouring problem based on the same dual-flow formulation and show that it is computationally superior on many instances. Although their instances do not have the staircase property, a reformulation trick allows them to use this formulation anyway. From these successes, there arises a natural interest to analyse the problem of recognizing the staircase property on a given graph $G = (V, E)$ and to generalize the applicability of the aforementioned dual-flow formulation for the clique problem.

At its core, staircase recognition reduces to finding suitable total orders on the subsets of a partition of the vertex set of the given graph. By way of incidence matrices, staircase compatibility gives an alternate approach to so-called staircase matrices. Namely, a bipartite graph has the staircase property if its adjacency matrix can be re-sorted into completely dense staircase form, i.e. the non-zeros in each row form a single block which is not interrupted by zero-elements. This way, staircase compatibility is closely linked to staircase matrices, which are an important structure in applied mathematics, cf. [Fou84]. These matrices are the natural target form to be achieved when performing Gauss-Jordan elimination and allow for efficient algorithms to solve linear programs (see [Fou82a, Fou82b]), to name only two examples.

Constructing conflict graphs from constraint matrices is one possible way to find clique structures in mixed-integer programming problems, see e.g. [BS19, ABG+20]. Apart from staircase compatibility, $m$-cliques in an $m$-partite graph can also be found in polynomial time if the so-called dependency graph is a forest; see [BGM20b], where the problem is studied under the name clique problem with multiple-choice constraints. For polyhedral properties of the general clique problem we refer to [Bor98, BBPP99].

Contribution We study the problem to recognize the staircase property in a graph with a given $m$-partition of the vertex set. For bipartite graphs, we can check the staircase property in time $O(|V| + |E|)$, and the algorithm can easily be extended to the case of $m$-partite graphs. Furthermore, we can give a full characterization of staircase orderings on bipartite graphs by showing that they are unique up to a few trivial operations on the two total orders they consist of. We will see that on bipartite graphs, the computation of staircase orderings is possible in polynomial time as well. For this task, we derive an algorithm which runs in $O(V_i(|V| + |E|))$, where $V_i$ can be chosen to be the smaller of the two bipartite subsets of vertices. This scheme can be extended to a polynomial-time algorithm for computing staircase orderings on $m$-partite graphs where all bipartite subgraphs $G_{ij}$ induced by two subsets $V_i$ and $V_j$ of the partition are connected. For general $m$-partite graphs, however, we show that deciding existence of a staircase ordering is NP-complete. Finally, if a given ordering is not staircase on an $m$-partite graph, it is possible to establish the staircase property by adding new edges to the graph. The problem of finding an ordering with the minimal number of edges to be added to the graph
such that the staircase property is fulfilled is NP-complete even in the case that all bipartite subgraphs $G_{ij}$ are connected.

These results are particularly interesting because in many cases they allow for efficiently checking practical instances of $m$-clique problems on $m$-partite graphs for the staircase property of their underlying graph. Such staircase-sortable instances are then solvable in polynomial time as a linear program (LP), as shown in [BGMS18]. In a computational study, we will demonstrate that our algorithms can efficiently check for the staircase property. Furthermore, for instances from a real-world timetabling problem we will see that first resorting them to be staircase and then solving them via the formulation from [BGMS18] is significantly faster than using a generic model for the problem.

**Structure**  The remainder of the article is organized as follows. Section 2 introduces the required notation and definitions to describe the problem. We then focus on bipartite graphs in Section 3, where we consider the recognition and uniqueness of staircase orderings and present algorithms to solve the problem in a polynomially solvable special case. Finally, in Section 5 we present the results of our computational study.

The main focus of our work is the so-called staircase property of a graph together with an $m$-partition of its vertex set and total orders on each subset. This property establishes local structures related to the clique problem on $m$-partite graphs.

**Definition 2.1.** (Subgraphs $G_{ij}$ and Dependency Graph $G$) Let $G = (V, E)$ be an $m$-partite graph together with a partition $V = \{V_1, \ldots, V_m\}$ of $V$. For two subsets $V_i, V_j \in V$ with $i \neq j$, we write

$$G_{ij} := (V_i \cup V_j, E_{ij})$$

for the subgraph of $G$ induced by $V_i \cup V_j$, where $E_{ij}$ is the corresponding edge set. Note that all subgraphs $G_{ij}$ are bipartite.

Furthermore, we call $G = (V, E)$ with

$$E := \{\{V_i, V_j\} \subseteq V \mid i \neq j \land \exists u \in V_i: N_{G_{ij}}(u) \neq V_j\}$$

the dependency graph of $G$. Note that $\{V_i, V_j\} \in E$ is equivalent to $G_{ij}$ not being a complete bipartite graph.

The main focus of our work is the so-called staircase property of a graph together with an $m$-partition of its vertex set and total orders on each subset. This property establishes local structures on all bipartite subgraphs $G_{ij}$ and can be used to efficiently solve certain clique problems on the graph. It is defined as follows.

**Definition 2.2.** (Ordering, Staircase Ordering) Let $G = (V, E)$ be an $m$-partite graph, together with a partition $V = \{V_1, \ldots, V_m\}$ of $V$ and total orders $>_i$ on the subsets $V_i \in V$. For simplicity, we refer to the set $\{>_1, \ldots, >_m\}$ of total orders as an ordering on $G$ if $V$ is clear from the context. An ordering on $G$ is staircase if for all subgraphs $G_{ij}$ of $G$ the following two conditions hold:

$$u \in V_i \land v_1 >_j v_2 >_j v_3 \in V_j \cup \{u, v_1\}, \{u, v_3\} \in E_{ij} \Rightarrow \{u, v_2\} \in E_{ij} \quad (SC1)$$

$$u_1 >_j u_2 \in V_i \land v_1 >_j v_2 \in V_j \cup \{u_1, v_1\}, \{u_2, v_1\} \in E_{ij} \Rightarrow \{u_1, v_1\}, \{u_2, v_2\} \in E_{ij}. \quad (SC2)$$
Condition (SC1) ensures that the neighbourhoods of all vertices are continuous with respect to the total order, whereas (SC2) yields a kind of monotonicity on the edge set $E_{ij}$. The two conditions are illustrated in Fig. 1.

![Illustration of (SC1)](image1)

![Illustration of (SC2)](image2)

Figure 1: Illustration of the two staircase conditions. If the solid edges are contained in the graph, the dashed ones must be contained as well.

Note that both (SC1) and (SC2) are trivially satisfied for complete bipartite subgraphs $G_{ij}$. Therefore, it is sufficient for the ordering on $G$ to be staircase if they hold for those $G_{ij}$ with $\{V_i, V_j\} \in E$. From the definition it also immediately follows that removing vertices from the graph does not destroy the staircase property. Moreover, it is easy to show that (SC1) is implied by (SC2) if $G_{ij}$ does not contain vertices with degree zero. In the next definition, we canonically extend the notion of a staircase ordering onto a partition $V$ on $V$.

**Definition 2.3.** (Staircase Partition) Let $G = (V, E)$ be an $m$-partite graph with vertex set partition $V = \{V_1, \ldots, V_m\}$. We call $V$ staircase if there exists a staircase ordering on $V$.

We close this section with some notation used to more concisely describe our algorithms.

**Definition 2.4.** (Minimal and Maximal Neighbour Index) Let $G = (V_1 \cup V_2, E)$ be a bipartite graph with total orders $>_1, >_2$ on $V_1$ and $V_2$, respectively, and $u \in V_1$ with $\deg_G(u) \geq 1$. We write

$$\lambda_{V_2}(u) := |\{ v \in V_2 | w >_2 v \forall w \in N_G(u)\}| + 1$$

$$\overline{\lambda}_{V_2}(u) := |V_2| - |\{ v \in V_2 | v >_2 w \forall w \in N_G(u)\}|$$

for the minimal and maximal index of any neighbour of $u$ in $V_2$ in the total order $>_2$.

## 3 Staircase Orderings on Bipartite Graphs

We start by presenting our results for staircase orderings on bipartite graphs. First, we state an algorithm to determine within polynomial time whether a given ordering on a bipartite graph is staircase. In Section 3.2, we then show that a staircase ordering on a bipartite graph is unique up to some easily recognizable situations where the order of vertices may be changed. Finally, an algorithm to compute a staircase ordering on a given bipartite graph within polynomial time is given in Section 3.3.

### 3.1 Recognizing Staircase Orderings

In this section, we will give a polynomial-time algorithm to verify whether a given ordering on a bipartite graph is staircase. Let $G = (V_1 \cup V_2, E)$ with $V_1 = \{u_1, \ldots, u_{|V_1|}\}$ and $V_2 = \{v_1, \ldots, v_{|V_2|}\}$ be a bipartite graph and consider the canonical ordering on $G$, given by the total orders

$$(\forall u_i, u_j \in V_1) u_i >_1 u_j :\iff i > j \text{ and}$$

$$(\forall v_i, v_j \in V_2) v_i >_2 v_j :\iff i > j.$$
We are able to show that it can be checked in polynomial time whether the canonical ordering on $G$ is staircase.

**Lemma 3.1.** It can be checked in time $O(|V| + |E|)$ whether the canonical ordering $\{>1, >2\}$ on a bipartite graph $G = (V_1 \cup V_2, E)$ with $V_1 = \{u_1, \ldots, u_{|V_1|}\}$ and $V_2 = \{v_1, \ldots, v_{|V_2|}\}$ is staircase.

**Proof.** Note that for any triplet $u_i, u_j, u_k \in V_1$, $i > j > k$ where $u_i$ and $u_k$ are in the same connected component $C$ of $G$, Condition (SC1) ensures that $u_j$ is in $C$ as well. In other words, all vertices belonging to $C$ appear consecutively in their respective total orders $>1$ and $>2$, which can be verified in time $O(|V| + |E|)$ using breadth-first search. Apart from this requirement, the staircase conditions can be checked on each connected component of $G$ separately. For the remainder of the proof, we therefore assume w.l.o.g. that $G$ is connected. The following algorithm then solves the problem.

**Algorithm 1** Recognizing a staircase ordering on a bipartite graph

**Input:** A connected bipartite graph $G = (V_1 \cup V_2, E)$ where $V_1 = \{u_1, \ldots, u_{|V_1|}\}$ and $V_2 = \{v_1, \ldots, v_{|V_2|}\}$

**Output:** Returns whether the canonical ordering on $G$ is staircase

1: if $\Delta V_2(u_1) - \Delta V_2(u_1) + 1 > |N_G(u_1)|$ then 
2: \hspace{1em} return "The canonical ordering violates (SC1)"
3: end if 
4: for $i \in \{2, \ldots, |V_1|\}$ do 
5: \hspace{1em} if $\Delta V_2(u_i) < \Delta V_2(u_{i-1})$ or $\Delta V_2(u_i) < \Delta V_2(u_{i-1})$ then 
6: \hspace{2em} return "The canonical ordering violates (SC2)"
7: \hspace{1em} end if 
8: \hspace{1em} if $\Delta V_2(u_i) - \Delta V_2(u_{i-1}) + 1 > |N_G(u_i)|$ then 
9: \hspace{2em} return "The canonical ordering violates (SC1)"
10: \hspace{1em} end if 
11: end for 
12: return "The canonical ordering on $G$ is staircase"

In Line 1, we directly check (SC1) for the neighbourhood of $u_1$. Afterwards, we check (SC1) for all remaining $u_i \in V_1$ and (SC2) for certain key vertices of $G$. The following case distinction shows that these checks indeed suffice to decide whether the canonical ordering on $G$ is staircase:

- **Case 1** – The canonical ordering on $G$ fulfils (SC1) and (SC2):
  It is clear that the checks in Lines 1 and 8 do not fail as they are equivalent to (SC1) with the vertex $u \in V_1$. Suppose there exists some $i \in \{2, \ldots, |V_1|\}$ such that $k := \Delta V_2(u_i) < j := \Delta V_2(u_{i-1})$, i.e. we fulfil the first condition of the check in Line 5. Then we have $u_i > u_{i-1}$ as well as $v_j > v_k$. Moreover, we have $\{u_i, v_k\} \in E$ and $\{u_{i-1}, v_j\} \notin E$ by definition of $\Delta V_2(u_{i-1})$. However, this is a contradiction to (SC2). Therefore, $\Delta V_2(u_i) \geq \Delta V_2(u_{i-1})$ must hold for all $i \in \{2, \ldots, |V_1|\}$. Analogously, we get that $\Delta V_2(u_i) \geq \Delta V_2(u_{i-1})$ holds for all $i \in \{2, \ldots, |V_1|\}$. Hence, Algorithm 1 works correctly if the canonical ordering is staircase.

- **Case 2a** – The canonical ordering violates (SC2):
  In this case, we can find $u_i, u_j \in V_1, v_j, v_j \in V_2$ with $i_1 < i_2 < j_1 < j_2$ and edges $\{u_i, v_j\}, \{u_j, v_j\} \in E$ such that $\{u_i, v_j\} \notin E$ or $\{u_j, v_j\} \notin E$ holds. Assume we have $\{u_i, v_j\} \notin E$. If $N_G(u_j)$ contains a vertex $v_j$ with $j_2 < j_3$, the check in Line 8 will fail for $i = i_2$. Therefore, we can assume w.l.o.g. $j_1 \leq \Delta V_2(u_{i_2}) < j_2$. However, this immediately
yields \( j_1 \leq \overline{\lambda}_2(u_i) < j_2 \leq \underline{\lambda}_2(u_i) \). Hence, for some \( i \in \{i_1 + 1, \ldots, i_2\} \) the check in Line 5 fails. Analogously, either of the checks in Line 5 or 8 must fail if \( \{u_i, v_j\} \notin E \).

- **Case 2b** – The canonical ordering violates \((SC1)\):
  If we find \( u \in V_1, v_1, v_2, v_3 \in V_2 \) such that \((SC1)\) is violated, the check in Line 1 or the one in Line 8 fails, depending on whether \( u = u_1 \) or \( u = u_i, i \in \{2, \ldots, |V_2|\} \). Otherwise, we can find \( u_1, u_2, u_3 \in V_1, v \in V_2 \) such that \((SC2)\) is violated, i.e. we have \( \{u_1, v\}, \{u_3, v\} \in E \) while \( \{u_2, v\} \notin E \). However, the vertex \( u_2 \) is by assumption not isolated and we can therefore find some vertex \( v' \in V_2 \) with \( \{u_2, v'\} \in E \). As Fig. 2 illustrates, the quadruple \( u_1, u_2, v', v \) violates \((SC2)\) if \( v > v' \) and the quadruple \( u_2, u_3, v, v' \) violates \((SC2)\) if \( v' > v \).
  In both cases, we can apply the arguments used in Case 2b.

To summarize, Algorithm 1 correctly recognizes a canonical ordering which is staircase. For non-staircase canonical orderings, at least one of the checks performed in Lines 1, 5 and 8 fails. The running time of Algorithm 1 is \( O(|V| + |E|) \), as the computation of all \( \lambda_2(u) \) and \( \overline{\lambda}_2(u) \) as well as the preprocessing step to identify the connected components of \( G \) can be done within this amount of time.

3.2 Uniqueness of Staircase Orderings

We have just established that recognizing whether an ordering on a bipartite graph is staircase can be done in polynomial time. Now we will further analyze the structure of staircase orderings on bipartite graphs. More precisely, we will show that it is possible to explicitly state all operations under which a given ordering remains staircase. This gives a full characterization of staircase orderings on bipartite graphs as they are indeed unique up to the following three operations.

First, it is easy to see that w.r.t. an ordering being staircase we can consider each connected component separately. Second, if we have \( N_G(u) = N_G(v) \) for a pair of vertices in a given subset \( V_i \), we can remove one of the two, consider the remaining graph and reinsert the removed vertex at the end. We will therefore assume, for a given bipartite graph \( G = (V_1 \cup V_2, E) \), the following two conditions w.l.o.g.:

\[
G \text{ is connected} \quad (1) \\
(\forall u, v \in V, u \neq v) N_G(u) \neq N_G(v). \quad (2)
\]

Third, any given staircase ordering remains staircase if all total orders are reversed at once, to which we refer as reversing the ordering. Note that this also holds for staircase orderings on \( m \)-partite graphs with \( m > 2 \). Theorem 3.3 states that a given staircase ordering on a bipartite graph is in fact unique up to the operations described above. Before proving Theorem 3.3 which is the main result of this section, the following auxiliary lemma gives additional insight on the lengths of shortest paths in bipartite graphs with a given staircase ordering.

Figure 2: Illustration of reduction of Case 2b to Case 2a

\[\square\]
Lemma 3.2. Let $G$ be a connected bipartite graph together with a staircase ordering $\{>_1, >_2\}$. Further, let $u, v, w \in V_1$ such that $u >_1 v >_1 w$, and let $P_{uv}$ and $P_{uw}$ be shortest paths between $u$ and $v$ or $w$, respectively. Then $|P_{uv}| \leq |P_{uw}|$.

Proof. We prove the claim by contradiction. Suppose there exist vertices $u, v, w \in V_1$ with $u >_1 v >_1 w$ and shortest paths $P_{uv} = (u, \ldots, v), P_{uw} = (u, \ldots, w', x, w)$ such that $|P_{uv}| > |P_{uw}|$. Note that $P_{uw}$ is of even length greater or equal to two, as $G$ is bipartite. We distinguish two cases:

- Case 1 – $w' >_1 v$:
  If we have $v \in N(x)$, then $P_{uv}$ is no longer a shortest path since we can construct a shorter one by simply choosing $v$ instead of $w$ as the last vertex in $P_{uw}$. Hence, $v \notin N(x)$ must hold. However, Fig. 3 illustrates that this already violates (SC1).

- Case 2 – $v >_1 w'$:
  In this case, we have $u >_1 v >_1 w'$ and $|P_{uv}| > |P_{uw'}| = |P_{uw}| - 2$. Therefore, we can start from the beginning with $w'$ assuming the role of $w$. This must eventually yield a contradiction because the path $P_{uw}$ is shortened by two, and once its length is equal to two this second case can no longer occur due to $u = w'$ and $u >_1 v >_1 w$.

![Figure 3: Contradiction for $w' >_1 v$](image)

The condition that $G$ be connected in Lemma 3.2 is not strictly necessary and could be replaced by assuming the paths $P_{uv}$ and $P_{uw}$ to actually exist. Then it would suffice to consider only the induced (connected) subgraph. With this observation in mind, it may seem that the lemma above yields an alternate characterization of the staircase property. However, as the example in Fig. 4 shows, this is not generally the case, even if $G$ is connected. Nonetheless, Lemma 3.2 allows us to prove the main result of this section.

![Figure 4: Although the paths $P_{uv}$ and $P_{uw}$ are of equal length, the ordering shown is not staircase since the quadruple $u, v, x, y$ violates (SC2).](image)

Theorem 3.3. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph which satisfies Conditions (1) and (2). Then a staircase ordering $\{>_1, >_2\}$ on $G$ is unique up to reversal.

Proof. Suppose there are two different staircase orderings $\{>_1, >_2\}$ and $\{>_1, >_2\}$ that cannot be transformed into each other by reversing either ordering.

Since the graph $G$ is connected but not complete bipartite, we can find vertices $u, v \in V_1$ and $x, y \in V_2$ that induce the subgraph shown in Fig. 5.
In this subgraph, we have
\[ u >_1 v \iff x >_2 y \quad \text{and} \quad u >_1 v \iff x >_2 y \]
due to Condition (SC2). Consequently, it is not possible to retain the staircase conditions by reversing only one total order of a given ordering. In other words, we can assume w.l.o.g. that there exist vertices \( u, v \in V_1 \) such that \( u >_1 v \land v >_1 u \) holds.

After possibly reversing all four total orders at once as well as possibly switching \( V_1 \) and \( V_2 \), we can, by the arguments above, find vertices \( u, v, w \in V_1 \) with \( u >_1 v >_1 w \) and \( u >_1 w >_1 v \). Note that in this case, Lemma 3.2 directly implies \( |P_{uv}| = |P_{uw}| \).

We now show that we can additionally assume the paths \( P_{uv} \) and \( P_{uw} \) to be of length two. If \( P_{uw} \) is of length greater or equal to four, let \( P_{uw} = (u, \ldots, w', x, w) \). Then the path \( P_{uw'} \) is shorter than \( P_{uw} \), and with the contraposition of Lemma 3.2, we have \( u >_1 w' >_1 v >_1 w \) as well as \( \{w', x\}, \{w, x\} \in E \). Hence, the edge \( \{v, x\} \) must be contained in \( E \) to not violate (SC1). Renaming \( w' \) to \( u \) now yields the desired assumption.

As the shortest paths \( P_{uv} \) and \( P_{uw} \) are of length two, we have that \( N(u) \cap N(v) \) and \( N(u) \cap N(w) \) are both non-empty. From (SC1) and \( u >_1 v >_1 w \) it follows that \( N(u) \cap N(v) \subseteq N(u) \cap N(v) \). Analogously, from (SC1) and \( u >_1 w >_1 v \) we can also conclude \( N(u) \cap N(v) \subseteq N(u) \cap N(w) \) and hence, \( N(u) \cap N(v) = N(u) \cap N(w) \neq \emptyset \).

Finally, Figures 6a and 6b illustrate that \( N(v) \setminus N(w) \) and \( N(w) \setminus N(v) \) must both be empty, as otherwise (SC2) is violated. However, we then have \( N(v) = N(w) \), which is a contradiction to Condition 2. This completes the proof.

This theorem fully characterizes the operations which can be applied to a staircase ordering to retain the staircase property.

### 3.3 Computing Staircase Orderings

We next introduce a polynomial-time algorithm that computes a staircase ordering on a given bipartite graph if possible, and otherwise states that no such ordering exists. As in the previous section, we may consider each connected component of a bipartite graph separately. The following algorithm then solves the problem.
Algorithm 2 Finding a staircase ordering on a bipartite graph

Input: A connected bipartite graph $G = (V_1 \cup V_2, E)$
Output: Lists $L_1, L_2$ of the vertices in $V_1, V_2$ representing a staircase ordering on $G$ or the result that no staircase ordering exists

1: for $r \in V_1$ do
2: Apply breadth-first search to $G$ starting in $r$ to sort vertices in $V_1 \cup V_2$ into sets $N_i = \{v \in V_1 \cup V_2 |$ shortest path from $r$ to $v$ is of length $i\}$, and let $i_{\max}$ be the length of the longest shortest path in $G$ starting in $r$.
3: $L_1 = [r], L_2 = []$
4: for $N_i = N_1, N_2, \ldots, N_{i_{\max}}$ do
5: Let $H$ be the subgraph of $G$ induced by $N_i \cup N_{i+1}$.
6: Sort vertices $u \in N_i$ in descending order by their degree in $H$.
7: if $i < |N|$ then
8: Let $H'$ the subgraph of $G$ induced by $N_i \cup N_{i+1}$.
9: Sort vertices $u \in N_i$ with equal degree in $H$ in ascending order by their degree in $H'$.
10: end if
11: if $i$ even then
12: Append the (now ordered) elements in $N_i$ to the list $L_2$
13: else
14: Append the (now ordered) elements in $N_i$ to the list $L_1$
15: end if
16: end for
17: if isStaircase($L_1 \cup L_2, E$) then
18: return $L_1, L_2$
19: end if
20: end for
21: return “A staircase ordering on $G$ does not exist”

There are no immediately obvious candidates for the uppermost vertex in a staircase ordering. Hence, the outer for-loop iterates over all vertices of $V_1$. For each choice $r \in V_1$, the inner for-loop constructs an ordering with $r$ as the uppermost vertex. The constructed ordering is then tested, e.g. by Algorithm 1, whether it is staircase. We prove the correctness of Algorithm 2 in Lemma 3.4 and show in Lemma 3.5 that its runtime is $O(|V|(|V| + |E|))$.

Lemma 3.4. Algorithm 2 is correct.

Proof. It is clear that the algorithm terminates in all cases.

If no staircase ordering exists on $G$, the staircase test in Line 16 will always fail and the algorithm correctly returns that no staircase ordering exists on $G$.

Otherwise, the outer for-loop will eventually choose a root $r \in V_1$ as the uppermost vertex of the total order $>_1$ such that a staircase ordering can be constructed. It remains to show that the operations within the inner for-loop are either strictly necessary to create a staircase ordering or simply choose one of multiple valid options.

Due to the contraposition of Lemma 3.2, vertices must be ordered by the length of the shortest path to the root vertex $r$. This is done by the combination of sorting the vertices into subsets $N_i$ during the breadth-first search and then appending them to the lists $L_1$ and $L_2$ in the correct order. Second, the exact order of vertices within each $N_i$ is determined by the operations in Lines 6 and 9. Depending on which operation determines the order between two vertices $u, v \in N_i$, we distinguish three cases. For ease of notation, we write $>_1$ to refer to either $>_1$ or $>_2$ depending on whether the vertices are contained in $V_1$ or $V_2$.

* $\deg_H(u) < \deg_H(v)$:
  This case can only occur if $|N_i| > 1$ as well as $|N_{i-1}| > 1$ hold. Due to $|N_0| = 1$, we have
general case. However, we will identify a special case which is solvable in polynomial time.

As stated by Lemma 3.1, each computed ordering can be checked in time $O(|V| + |E|)$. Each set $N_i$ can be sorted in time $O(|N_i| + \max_{v \in N_i} \deg_G(v))$ via counting sort. The time complexity of the combined time required for sorting can be bounded by the estimate

$$\sum_{i=1}^{k} (|N_i| + \max_{v \in N_i} \deg_G(v)) \leq \sum_{i=1}^{k} |N_i| + \sum_{i=1}^{k} \max_{v \in N_i} \deg_G(v) \leq |V| + |E|.$$ 

As stated by Lemma 3.1, each computed ordering can be checked in time $O(|V| + |E|)$ by Algorithm 1. In total, an iteration of the outer for-loop takes $O(|V| + |E|)$ time. Since there are at most $|V_1|$ iterations, this completes the proof.

In Sections 3.1 to 3.3, we have studied staircase orderings on bipartite graphs and concluded that we can determine an ordering on a bipartite graph to be staircase withing polynomial time and also compute one in polynomial time. Further, a staircase ordering on a bipartite graph is unique up to changing the order of connected components, changing the order of vertices with equal neighbourhood and reversing the entire ordering. We will see in the next section that the recognition problem on $m$-partite graphs with given partition is NP-complete in the general case. However, we will identify a special case which is solvable in polynomial time.
4 Staircase Orderings on \( m \)-partite Graphs

We now consider the staircase recognition problem on \( m \)-partite graphs. More precisely, the partition \( V = \{V_1, \ldots, V_m\} \) on the vertex set of the graph is assumed to be given and it is to be determined whether \( V \) is a staircase partition, i.e. if there exist corresponding staircase orderings \( \{>_1, \ldots, >_m\} \) on its subsets.

It can be shown that recognizing staircase partitions in the general case is NP-complete. In Section 4.1 we will give a proof that uses a reduction from the NP-complete Betweenness Problem, which has been presented in the dissertation [Mer17]. In the special case where all bipartite subgraphs \( G_{ij} \) of an \( m \)-partite graph are connected, we will see in Section 4.2 that Algorithm 2 can be extended to compute staircase orderings within polynomial time.

4.1 Complexity of Recognizing Staircase Partitions

We start by showing that recognizing a staircase partition is NP-complete on general \( m \)-partite graphs.

**Theorem 4.1.** Let \( G = (V, E) \) be an \( m \)-partite graph with vertex set partition \( V = \{V_1, \ldots, V_m\} \). The problem to decide whether a staircase ordering on \( G \) exists is NP-complete, even if the dependency graph \( G \) is cycle-free.

**Proof.** We prove the theorem by giving a polynomial-time reduction from the Betweenness Problem. In an instance of Betweenness, a ground set \( M \) is given together with a set \( \mathcal{M} \) of ordered triples of elements from \( M \). One has to determine whether there exists a total order on \( M \) such that the middle item of each given triple is placed somewhere between the other two items. This problem is known to be NP-complete [Opa79].

From an instance of Betweenness, we construct an equivalent instance of the recognition problem for staircase partitions as follows: We have a special subset \( V_0 = M \), where each vertex is identified with an item from Betweenness. For the triples \( t = (u, v, w) \in \mathcal{M} \), we construct subsets consisting of 3 vertices each that are identified with the items from the respective triple (but of course different from the vertices in \( V_0 \)). Each vertex in such a subset is solely compatible with the vertex in \( V_0 \) corresponding to the same item. Moreover, each subset \( V_t \) representing a triple \( t \) from \( \mathcal{M} \) comes together with an auxiliary subset \( V_t' \) consisting of two vertices. This subset \( V_t' \) has dependencies only with \( V_t \) and these are designed to force \( v_t \), the designated middle element of the triple, to be sorted in between the other two elements in \( V_t \) (see Fig. 8). Note that there are exactly two possibilities for choosing orderings on \( V_t \) and \( V_t' \), which correspond to the two possible orderings of the elements \( u, v, w \) from \( t \) that put \( v \) in the middle. The bipartite subgraphs between subsets or auxiliary subsets corresponding to different triples are complete bipartite graphs such that the dependency graph \( G \) is a tree of depth 2 with root vertex \( V_0 \) and leaves \( V_t', t \in \mathcal{M} \).

On the one hand, any solution of Betweenness gives us a total order on \( M \) that we can use for \( V_0 \). This ordering will ensure that the middle element of each \( V_t \) for \( t \in \mathcal{S} \) is between the other elements of \( t \). Therefore, there will be no crossing of edges between \( V_t \) and \( V_0 \), and hence no violation of (SC2) if the ordering on \( V_t \) and \( V_t' \) is oriented correctly. On the other hand, every choice of total orders on the subsets that yield a staircase ordering on \( G \) immediately gives us an ordering of \( V_0 \) that is a solution to Betweenness. Finally, the transformation of instances is clearly polynomial in the encoding size of the input. \( \square \)

We would like to remark that for an instance of the recognition problem constructed as in the above proof, no ordering can possibly violate (SC1) without also violating (SC2), which means that finding orderings that just satisfy (SC2) is also NP-complete.

Generally, instances in the proof of Theorem 4.1 feature subgraphs \( G_{ij} \) with isolated vertices. Although the general staircase ordering problem is NP-complete, we can show that the problem becomes solvable in polynomial time under an additional connectedness assumption.
4.2 A Polynomial Special Case

In this subsection, we will show that the problem becomes polynomially solvable if all $G_{ij}$ are connected. Before proving this statement, we want to introduce a way to represent all staircase orderings on a given connected subgraph $G_{ij}$ as well as a way to intersect these representations.

As mentioned in Subsection 3.2, there are two ways to modify an ordering on a connected bipartite graph $G_{ij}$ while maintaining the staircase property of the ordering. First, we can simultaneously reverse the ordering and second, we can switch places of vertices $u, v$ with $N_{G_{ij}}(u) = N_{G_{ij}}(v)$.

The first operation can only yield a staircase ordering on $G$ if the total orders of all $V_i$ are reversed at once. Therefore, we can “ground” the ordering by fixing the total order on some $V_i$ and aligning all other total orders with the fixed one.

In contrast, the second operation can change the total order on a given $V_i$ in a way that cannot be uniformly extended to all other subsets $V_j$. To represent this operation, we use preorders, i.e. reflexive, transitive binary relations. Let $G_{ij}$ be some bipartite subgraph of $G$ and $>_i,>_j$ the total orders on $V_i, V_j$ found by Algorithm 2. Then all valid partial reorderings on $V_i$ and $V_j$ are represented by the total preorders

$$(\forall k \in \{i, j\}) u \geq_k v \Leftrightarrow u >_k v \lor (v >_k u \land N_{G_{ij}}(u) = N_{G_{ij}}(v)).$$

In other words, if $\{>_i,>_j\}$ is another staircase ordering on $G_{ij}$, applying this definition to the ordering itself or its reverse will yield $\geq_i$ and $\geq_j$. The next two lemmas show under which circumstances we can merge these preorders.

**Lemma 4.2.** Let $G_{ij}, G_{jk}$ be bipartite connected graphs and $\geq_i, \geq_j, \geq_k$ the total preorders representing all staircase orderings on $G_{ij}$ and $G_{jk}$ respectively. If, even after possibly reversing $\geq_j$ and $\geq_k$, there exists no vertex $u \in V_i$ with $u \geq_j v, u \geq_j v$ for all $v \in V_j$, there cannot be total orders $>_i,>_j,>_k$ such that the orderings $\{>_i,>_j\}$ and $\{>_j,>_k\}$ are staircase on $G_{ij}$ and $G_{jk}$ respectively.

**Proof.** Suppose there are total orders $>_i,>_j,>_k$ such that the orderings $\{>_i,>_j\}$ and $\{>_j,>_k\}$ are staircase on $G_{ij}$ and $G_{jk}$ respectively. Then there exists a vertex $u \in V_j$ with $u >_j v \forall v \in V_j$. However, $>_j$ must be represented by $\geq_j$ as well as $\leq_j$ and therefore we have $u \geq_j v, u \geq_j v \forall v \in V_j$, which is a contradiction. \qed

**Lemma 4.3.** Let $G_{ij}, G_{jk}$ be bipartite connected graphs and $\geq_i, \geq_j, \geq_k$ the total preorders representing all staircase orderings on $G_{ij}$ and $G_{jk}$ respectively. Further, assume there exists some vertex $u \in V_j$...
with \( u \geq_j v, u \geq_j v \forall v \in V_j \). Then there exist staircase orderings \( \{ >_i, >_j \} \) and \( \{ >_j, >_k \} \) on \( G_{ij} \) and \( G_{jk} \) respectively if and only if the composed preorder

\[
\forall j \in V_j : u \geq_j v \Leftrightarrow u \geq_j v \land u \geq_j v
\]

is total. Further, the total preorders \( \geq_i, \geq_j, \geq_k \) represent exactly the set of total orders \( >_i, >_j, >_k \) such that the orderings \( \{ >_i, >_j \} \) and \( \{ >_j, >_k \} \) are staircase on \( G_{ij} \) and \( G_{jk} \) respectively.

**Proof.** Let the composed preorder \( \succeq_j \) be total and \( >_j \) some total order represented by \( \succeq_j \). Then \( >_j \) is also represented by \( \geq_j \) and \( \geq_j \) is the composition of these two preorders. Hence, we can find total orders \( >_i, >_j, >_k \) such that the orderings \( \{ >_i, >_j \} \) and \( \{ >_j, >_k \} \) are staircase on \( G_{ij} \) and \( G_{jk} \) respectively. In fact, any triple \( >_i, >_j, >_k \) of total orders represented by \( \geq_i, \geq_j, \geq_k \) gives us staircase orderings on \( G_{ij} \) and \( G_{jk} \). In summary, we can construct staircase orderings on \( G_{ij} \) and \( G_{jk} \) from the total preorder \( \succeq_j \) and each combination of total orders \( >_i, >_j, >_k \) represented by \( \geq_i, \geq_j, \geq_k \) yields staircase orderings on \( G_{ij} \) and \( G_{jk} \).

Conversely, if total orders \( >_i, >_j, >_k \) exist such that the orderings \( \{ >_i, >_j \} \) and \( \{ >_j, >_k \} \) are staircase on \( G_{ij} \) and \( G_{jk} \) respectively, we have \( u >_j v \lor v >_j u \) for all pairs \( u, v \in V_j \). If \( u >_j v \), both \( u \geq_j v, u \geq_j v \) must hold, and by definition we have \( u \geq_j v \). Analogously, we get \( v \succeq_j u \) if \( v >_j u \). As \( u, v \) were chosen arbitrarily, \( \succeq_j \) is a total preorder. Further, \( >_i, >_j, >_k \) are represented by \( \geq_i, \geq_j, \geq_k \). Altogether, if total orders \( >_i, >_j, >_k \) exist such that the orderings \( \{ >_i, >_j \} \) and \( \{ >_j, >_k \} \) are staircase on \( G_{ij} \) and \( G_{jk} \) respectively, then \( \succeq_j \) must be total and the total orders are represented by \( \geq_i, \geq_j, \geq_k \). This completes the proof. \( \square \)

**Example 4.4.** Consider the graph \( G \) shown in Fig. 9a which can be ordered to be staircase. In the subgraph \( G_{12} \), we can arbitrarily arrange \( v_3 \) and \( v_2 \), and in the subgraph \( G_{23} \) we can arbitrarily arrange \( v_2 \) and \( v_1 \). We write this as \( v_3 =_2 v_2 \) and \( v_2 \approx_2 v_1 \), respectively. As a result, we have \( v_3 =_2 v_2 \geq_2 v_1 \) and \( v_3 \geq_2 v_2 \approx_2 v_1 \). The composed preorder, which is indeed total, is then given by \( v_3 \geq_2 v_2 \geq_2 v_1 \).

In contrast, the bipartite subgraphs \( H_{12} \) and \( H_{23} \) of \( H \) shown in Fig. 9b can be ordered staircase with the preorders \( v_3 =_2 v_2 \geq_2 v_1 \) and \( v_3 \approx_2 v_1 \geq_2 v_2 \), respectively. However, in the composed preorder \( \succeq_2 \) we have neither \( v_2 \geq_2 v_1 \) nor \( v_1 \geq_2 v_2 \), and there indeed exists no staircase ordering on \( H \).

![Figure 9: An example where the composed preorder is not total and no staircase ordering on G exists](image)

We proceed to prove the main theorem of this section, which states that, if all subgraphs \( G_{ij} \) are connected, the problem to decide whether a staircase ordering exists on a given graph with given partition is solvable in polynomial time. Our proof will be constructive, yielding an algorithm to even compute a staircase ordering if it exists, and we will investigate its performance in Section 5.
Theorem 4.5. Let $G$ be an $m$-partite graph with a partition $\mathcal{V} = \{V_1, \ldots, V_m\}$ of the vertex set such that all $G_{ij}$ are connected. Then it can be decided within polynomial time whether a staircase ordering on $G$ exists.

Proof. We prove this theorem by describing an algorithm that solves the problem in polynomial time. The idea is to perform a breadth-first search on the dependency graph and to use Algorithm 2 on $G_{ij}$ whenever an edge $\{V_i, V_j\}$ is traversed. If necessary, the computed total orders on $V_i$ and $V_j$ will be merged with already existing orders on $V_i$ and $V_j$. In the case where a staircase ordering exists, the breadth-first search traverses all edges and the last merging step yields a staircase ordering on $G$. Otherwise, the process stops as one of the merging step fails or Algorithm 2 already recognizes that no staircase ordering exists for some subgraph $G_{ij}$. If the dependency graph $G$ consists of more than one connected component, we can simply apply the above procedure to each connected component. Therefore, we can assume w.l.o.g. that $G$ is connected as well.

It is clear that a staircase ordering on $G$ cannot exist if Algorithm 2 determines that for some $G_{ij}$ no staircase ordering can be found. On the other hand, if a staircase ordering on $G$ does exist, Algorithm 2 does find a staircase ordering on $G_{ij}$, which yields new total preorders $\geq_i, \geq_j$ on $V_i$ and $V_j$ respectively. Suppose there already exists some total preorder $\geq_i$ on $V_i$. If, even after possibly reversing $\geq_i$ and $\geq_j$, we cannot find a vertex $u \in V_i$ with $u \geq_i v, u \geq_j v \forall v \in V_i$, which can be checked in polynomial time, then by Lemma 4.2 there exists no staircase ordering on $G$. Otherwise consider the composed preorder

$u \geq_j v \iff u \geq_i v \land u \geq_j v$.

If the composed preorder is not total, then by Lemma 4.3 there exists no staircase ordering on $G$. We want to emphasize that applying Algorithm 2 to some $G_{ij}$ yields two total preorders $\geq_i, \geq_j$ that need to be merged and we must either reverse both new total preorders or none as reversing only one of them would yield an ordering that is not staircase on $G_{ij}$. This includes the case where, for example, $\geq_j$ is the first total preorder on $V_j$. If the breadth-first search finishes and the last merging step succeeds, we have total preorders on all $V_i \in \mathcal{V}$. Deriving any combination of total orders then yields a staircase ordering on $G$. The resulting algorithm can then be summarized as follows.

Algorithm 3 Finding a staircase ordering on $m$-partite graphs with connected subgraphs $G_{ij}$

Input: A graph $G = (V, E)$ together with a partition $\mathcal{V}$ of its vertex set $V$ and the property that all subgraphs $G_{ij}$ are connected

Output: Returns a staircase ordering or the message that no staircase ordering exists

1: Initialize total preorders on all $V_i \in \mathcal{V}$ as $u \geq v \forall u, v \in V_i$ Select root vertex $R \in \mathcal{V}$
2: for edge $\{i, j\}$ being traversed by the breadth-first search starting in $R$ do
3: Sort $G_{ij}$ via Algorithm 2
4: if No staircase ordering on $G_{ij}$ exists then
5: return “A staircase ordering on $G$ does not exist”
6: end if
7: Compute corresponding preorders on $V_i$ and $V_j$
8: Unify the new preorders with existing preorders on $V_i$ and $V_j$
9: if Unification failed then
10: return “A staircase ordering on $G$ does not exist”
11: end if
12: end for
13: return The computed staircase ordering on $G$
In short, the breadth-first-search approach can indeed decide whether a staircase ordering exists on the graph \( G \). Further, its running time is polynomial in the input size, as we traverse each edge in the dependency graph \( \mathcal{G} \) at most once, Algorithm 2 runs in polynomial time, and finally, each merging step can be done in polynomial time as well.

It is worth mentioning that, since the instances in the proof of Theorem 4.1 may feature subgraphs \( G_{ij} \) with isolated vertices, it does not state that deciding whether staircase sorting is hard under the assumption of nonempty neighbourhoods in each \( G_{ij} \), which is a slightly weaker assumption than the one made in Theorem 4.5.

Before presenting the results of our computational experiments, we want to state an interesting observation. Any given ordering on any given graph can be made staircase by adding additional edges to the graph until the staircase conditions are fulfilled. In fact, only adding edges of violated staircase conditions until the ordering becomes staircase, yields, in terms of number of additional edges, the optimal staircase relaxation. If the partition \( V \) is not staircase, it may then be relevant to find an ordering with minimal staircase relaxation. The following theorem, which is a slight modification of \cite[Theorem 5.20]{Mer17}, extending it to graphs satisfying our connectedness assumption from Theorem 4.5, yields a surprising result. Namely, it shows that finding such an ordering remains NP-complete, even if all bipartite subgraphs \( G_{ij} \) are connected and each \( V_i \in V \) contains at most two elements.

**Theorem 4.6.** Let \( G \) be an \( m \)-partite graph with a partition \( V = \{V_1, \ldots, V_m\} \) of the vertex set such that all \( G_{ij} \) are connected. Then the optimization problem to find an ordering on \( G \) such that its staircase relaxation is minimal is NP-complete, even when the problem is restricted to at most two elements per partition.

**Proof.** We show the theorem by a polynomial-time reduction from the famous MaxCut problem. For a given graph \( H \), it asks for partitioning the vertices into two sets such that the number of arcs between both partitions is maximized. The weighted version of the corresponding decision problem was one of Karp’s 21 NP-complete problems \cite{Kar72}, and also the unweighted case is well known to be NP-hard \cite{GJ79}.

From an instance of MaxCut on a graph \( H \), we construct an equivalent instance of the optimization problem from Theorem 4.6 with only two elements per partition: For each \( v \in V(H) \), construct a partition \( V_v = \{\overline{v}, \overline{v}\} \) consisting of the two elements \( \overline{v}, \overline{v} \). For \( \{u, v\} \in E(H) \) we then add the edges \( \{u, \overline{v}\}, \{\overline{u}, \overline{v}\}, \{\overline{u}, v\} \) (see Fig. 10). For \( u, v \in V(H), \{u, v\} \notin E(H) \) we add edges such that \( G_{uv} \) is complete bipartite, i.e. \( \{V_u, V_v\} \notin \mathcal{G} \). Hence, all resulting bipartite subgraphs \( G_{ij} \) are connected.

Therefore, sorting \( u < v \) and \( \overline{v} < \overline{v} \) would contribute exactly one missing edge to the objective, while flipping one of the orderings (but not both) satisfies (SC1) and (SC2) for the two partitions \( V_u \) and \( V_v \). As a consequence, any cut in \( H \) containing \( l \) edges with vertex subsets \( S, T \) corresponds to a choice for total orderings on the subsets \( V_v, v \in V(H) \) with \( |E(H)| - l \) violations, namely \( \overline{v} < \overline{v} \overline{v} \forall v \in S \) together with \( \overline{v} > \overline{v} \overline{v} \forall v \in T \), and vice versa. \( \square \)
5 Computational Experiments

In the following, we will empirically evaluate Algorithm 3 to decide whether a staircase ordering exists on a given $m$-partite graph exists, and to compute such an ordering if it does. First, we consider two sets of randomly generated instances. On a set of small instances, we benchmark the performance of Algorithm 3 against the mixed-integer program (MIP) for finding staircase orderings suggested in [Mer17]. Several sets of larger instances are then used to test the limits of Algorithm 3. Second, we demonstrate that our algorithm can staircase-sort a set of real-world railway timetabling instances. It will turn out that using the sorted version, which allows for using the totally unimodular dual-flow formulation from [BGMS18], is much more efficient than using a generic model for the problem.

We have implemented Algorithm 3 in Python 3.7.9, using the Python library NetworkX 2.5 ([HSSC08]) for our graph operations. For the MIP approach, we used the Python API of Gurobi 9.1.0 ([GO20]). All experiments have been run on machines with Xeon E3-1240 v6 CPUs (“Kaby Lake”, 4 cores, HT disabled, 3.7 GHz base frequency) and 32 GB RAM.

5.1 Computational Experiments on Random Instances

The artificial instances in our first benchmark set were generated as follows. First, we created a dependency graph $G = (V, E)$ with $m$ vertices and $E = \emptyset$. For each pair of vertices $V_i, V_j \in V$, $i \neq j$ in $G$, we then added the edge $\{V_i, V_j\}$ with probability $p \in [0, 1]$. To ensure that the generated instances do not decompose into multiple smaller instances, we randomly chose two vertices $V_i$ and $V_j$ from two different randomly chosen connected components of $G$ and iteratively added the corresponding edge $\{V_i, V_j\}$ until the entire dependency graph was connected. If the so-obtained $G$ was cycle-free, we randomly added one more edge such that it contained at least one cycle. This later allowed us to create a non-staircase version of the instance without simply adding a non-staircase subgraph $G_{ij}$ (see below). From this dependency graph $G$, we then generated $G$ as follows. For each $V_i \in V$, we randomly drew a subset size $n_i \in [n_i, \overline{n}_i], n_i, \overline{n}_i \in N^+$ and added the vertices $v_{i,1}, \ldots, v_{i,n_i}$ to subset $V_i$. Finally, for each edge $\{V_i, V_j\} \in \overline{E}$, we use Algorithm 4 as a generator for the (connected) subgraph $G_{ij}$.

After initializing the vertex sets $V_i$ and $V_k$, the edge set $E$ as well as the auxiliary variables $\text{minidx}, \text{maxidx}$ in Lines 1 to 5, Algorithm 4 iterates over the vertex indices in $V_i$ in Line 6. For each $v_{i,k}$, its neighbourhood is generated as follows. First, we draw an integer which will represent the smallest index of any neighbour of $v_{i,k}$ in Lines 9 and 10. The exponential function is chosen in order to prevent the generated graphs from being sparse at one end of the ordering while being very dense at the other end of the ordering. We then check if the drawn value indeed represents the index of a neighbour of $v_{i,k}$ (Line 11), and, if necessary, adjust it in Lines 12 to 16 such that the generated graph remains connected. A similar procedure follows for finding a new maximal neighbour index. When valid indices are found, the new edges are added in Line 29. Note, that the minimal and maximal neighbourhood indices increase monotonously in each iteration, and therefore the canonical ordering is indeed staircase for $G_{ij}$.

To test the performance of Algorithm 3 on non-staircase instances, we added a non-staircase duplicate of each instance to our test set. Since Algorithm 2 already detects if no staircase ordering exists on a given $G_{ij}$, which therewith holds for the entire graph $G$, we broke the staircase property in a way that forces Algorithm 3 to also take interdependencies between subgraphs $G_{ij}$ into account. In particular, we created a set of all edges in $\overline{E}$ that are contained in at least one cycle in $G$ and chose one of these edges uniformly at random. In the corresponding subgraph $G_{ij} = (V_i \cup V_j, E_{ij})$, we then flipped $V_j$, i.e. we changed the edge set to

$$\tilde{E}_{ij} = \{\{v_{i,n}, v_{j,m}\} \mid \{v_{i,n}, v_{j,m}\} \in E_{ij}, m' = |V_j| - m + 1\}. $$
Algorithm 4 Generator for bipartite graphs with staircase canonical ordering

**Input:** Subset sizes \( n_i, n_j \in \mathbb{N}^+ \)

**Output:** A bipartite graph \( G_{ij} \) such that the canonical ordering is staircase

1: \( V_i = \{v_{i,1}, \ldots, v_{i,n_i}\} \)
2: \( V_j = \{v_{j,1}, \ldots, v_{j,n_j}\} \)
3: \( E_{ij} = \emptyset \)
4: \( \text{minIdx} = 0 \)
5: \( \text{maxIdx} = 0 \)
6: for \( k = 1, 2, \ldots, n_i \) do
7: \( \text{foundNewMin} = \text{False} \)
8: while not \( \text{foundNewMin} \) do
9: Choose \( x \geq 0 \) with probability density function \( f(x) = \lambda e^{-\lambda x} \) where \( \lambda = \frac{n_i}{n_j} \)
10: \( \text{newMinIdx} = \text{minIdx} + \lfloor (x + 0.5) \rfloor \) \{round to nearest integer\}
11: if \( \text{newMinIdx} < n_j \) then
12: if \( \text{newMinIdx} > \text{maxIdx} \) then
13: \( \text{minIdx} = \text{maxIdx} \)
14: else
15: \( \text{minIdx} = \text{newMinIdx} \)
16: end if
17: \( \text{foundNewMin} = \text{True} \)
18: end if
19: end while
20: \( \text{foundNewMax} = \text{False} \)
21: while not \( \text{foundNewMax} \) do
22: Choose \( x \geq 0 \) with probability density function \( f(x) = \lambda e^{-\lambda x} \) where \( \lambda = \frac{n_i}{n_j} \)
23: \( \text{newMaxIdx} = \text{maxIdx} + \lfloor (x + 0.5) \rfloor \) \{round to nearest integer\}
24: if \( \text{newMinIdx} \leq \text{newMaxIdx} < n_j \) then
25: \( \text{maxIdx} = \text{newMaxIdx} \)
26: \( \text{foundNewMax} = \text{True} \)
27: end if
28: end while
29: \( E_{ij} = E_{ij} \cup \{\{v_{i,k}, v_{j,l}\} | l \in \{\text{minIdx}, \text{minIdx} + 1, \ldots, \text{maxIdx}\}\} \)
30: end for
31: return \( G_{ij} = (V_i \cup V_j, E_{ij}) \)

To further improve the accuracy of our computations, we created five random instances for each set of parameters \( m, p, \pi, \varpi \) used. Our result tables use the parameter naming scheme

\[
m-p-n-\pi, \varpi \tag{3}
\]

with the postfix “-nonSC” to indicate the non-staircase duplicates.

Finally, we prepend each vertex in \( V \) and \( V \) with four random lowercase letters and rebuild the graphs \( G \) and \( G \) by adding vertices and edges in the resulting random lexicographic order to destroy any orders that NetworkX might derive from the vertex names or the order in which they have originally been added to the graphs.

An MIP formulation based on the linear ordering problem which is able to decide whether
a staircase ordering exists, can be stated as follows:

\[
\begin{align*}
\text{find } & \quad z \\
\text{s.t. } & \quad z_{uv} + z_{vu} = 1 \quad \forall v_i \in V \forall u \neq v \in V_i \\
& \quad z_{uv} + z_{vu} + z_{wu} \leq 2 \quad \forall v_i \in V \forall u \neq v \neq w \\
& \quad z_{uv} - z_{v'\nu'} = 0 \quad \forall u_i >_{ij} v, u' >_{ij} v' : \text{(SC2) is violated} \\
& \quad z_{uv} \in \{0, 1\} \quad \forall (u, v) \in \bigcup_{i=1}^{m} (V_i \times V_i).
\end{align*}
\]

Variables \(z_{uv} \in \mathcal{F}\) stand for the decision to order vertex \(u \in V\) before vertex \(v \in V\) for each \(u\) and \(v\) in the same subset of \(V\). Constraint (4a) ensures that exactly one of two possible orders between two vertices is chosen, while Constraint (4b) requires the order within a subset to be acyclic. Finally, Constraint (4d) demands that the orders fulfill Condition (SC2). Note that we do not need to explicitly model (SC1), as it is implied by (SC2) if \(G_{ij}\) is connected. Cf. [Mer17] for this MIP approach.

Table 1 shows our results for the comparison between building and solving the MIP and using Algorithm 3. In the columns, we state the parameter set (see (3)), the dimensions of the graphs \(G\) and \(G\) as well as the time required to build the MIP model, solve the MIP model, or solve the problem with Algorithm 3. For increased readability, each row represents the averages of the five instances with the given parameter set, where the resulting graph dimensions are rounded to integer values.

In terms of performance, we make two important observations. On the one hand, when comparing the pure solution times, the MIP model outperforms our Python-based implementation of Algorithm on the given benchmark set. Second, however, the MIP model takes a very long time to build, as its number of constraints is double-quadratic in the number of vertices per subset of the partition. For example, the MIP model for the first instance with parameters 10-0.5-10-10 consists of only 9,182 rows, 900 columns and 20,764 nonzeros, whereas the MIP model of the first instance with parameters 10-0.5-50-50 already consists of 1,391,617 rows, 24,500 columns and 3,175,234 nonzeros. As a result, increasing the subset size from 10 to 50 already led to a 1000-fold increase in MIP build time in our test.

Although the picture remains largely the same between staircase and non-staircase instances, we can observe that solution times tend to be lower for the non-staircase instances, as either method can stop once it has found the first conflict. The largest variance has been found for the five instances of 150-0.5-10-10, ranging from 57.8 seconds to 65.0 seconds, whereas solution times for the five non-staircase instances show much higher variance and range from 6.6 seconds to 56.6 seconds.

Finally, we conclude that although the model construction time of the MIP scales roughly linearly with \(|\mathcal{E}|\), it quickly becomes prohibitively long for practical application as instance size increases. We therefore neglect this approach for the larger instances we present in the following.

The results on the impact of varying the subset size are given in Table 2. The experiments with the first five parameter sets show that computation time quickly increases with subset size, as Algorithm 2 which is used to sort the bipartite subgraphs \(G_{ij}\), is of quadratic complexity in the number of vertices per subset. The experiments with the next five parameter sets show that increasing the variance in subset size while leaving the expected number of vertices per subset at 250 vertices also slightly increases computation time. As before, we find that even with just a single conflict prohibiting a staircase ordering from existing, computation times are significantly lower for non-staircase instances.

The next set of random instances features a gradually increasing number of vertices in the dependency graph. We consider it to study the impact of an increasing number of bipartite subgraphs \(G_{ij}\). The results can be seen in Table 3. Here we see that the computation time of
Table 1 Performance of sorting algorithm and linear-ordering MIP on small random instances

| Instance  | |V| | |E| | |V| | |E| | MIP Building | MIP Solution | Algorithm |
|----------|-------------------|---|---|---|---|---|---|---|---|---|---|---|
| 10-0.5-10-10 | 10 | 21 | 100 | 643 | 4.0 | 0.0 | 0.1 |
| 10-0.5-20-20 | 10 | 21 | 200 | 846 | 81.0 | 0.1 | 0.2 |
| 10-0.5-30-30 | 10 | 19 | 300 | 2675 | 390.4 | 0.3 | 0.6 |
| 10-0.5-40-40 | 10 | 22 | 400 | 4968 | 1604.0 | 1.2 | 1.5 |
| 10-0.5-50-50 | 10 | 21 | 500 | 7053 | 4335.6 | 2.4 | 2.5 |
| 50-0.5-10-10 | 50 | 619 | 500 | 19894 | 329.4 | 0.3 | 3.1 |
| 75-0.5-10-10 | 75 | 1385 | 750 | 44454 | 972.5 | 0.9 | 8.5 |
| 100-0.5-10-10 | 100 | 2467 | 1000 | 79517 | 2236.0 | 2.4 | 18.8 |
| 125-0.5-10-10 | 125 | 3897 | 1250 | 125680 | 4305.4 | 3.5 | 35.3 |
| 150-0.5-10-10 | 150 | 5599 | 1500 | 179797 | 7300.8 | 6.3 | 60.6 |
| 10-0.5-10-10-nonSC | 10 | 21 | 100 | 643 | 4.0 | 0.0 | 0.0 |
| 10-0.5-20-20-nonSC | 10 | 21 | 200 | 846 | 81.5 | 0.1 | 0.2 |
| 10-0.5-30-30-nonSC | 10 | 19 | 300 | 2675 | 381.2 | 0.3 | 0.3 |
| 10-0.5-40-40-nonSC | 10 | 22 | 400 | 4968 | 1626.2 | 1.0 | 1.1 |
| 10-0.5-50-50-nonSC | 10 | 21 | 500 | 7053 | 4267.8 | 2.5 | 1.3 |
| 50-0.5-10-10-nonSC | 50 | 619 | 500 | 19894 | 316.6 | 0.3 | 1.2 |
| 75-0.5-10-10-nonSC | 75 | 1385 | 750 | 44454 | 964.5 | 0.7 | 2.1 |
| 100-0.5-10-10-nonSC | 100 | 2467 | 1000 | 79517 | 2193.1 | 1.2 | 11.8 |
| 125-0.5-10-10-nonSC | 125 | 3897 | 1250 | 125680 | 4283.4 | 1.8 | 26.4 |
| 150-0.5-10-10-nonSC | 150 | 5599 | 1500 | 179797 | 7230.8 | 4.1 | 31.3 |

Algorithm 3 scales linearly with |E|. Again, computation times for non-staircase instances are significantly lower, but show higher variance.

Table 3 Performance of sorting algorithm on random instances with varying dependency graph size

| Instance  | |V| | |E| | |V| | |E| | Algorithm Time (s) |
|----------|-------------------|---|---|---|---|---|---|---|---|---|---|---|
| 100-0.1-50-50 | 100 | 511 | 5000 | 165292 | 72.0 |
| 200-0.1-50-50 | 200 | 2034 | 10000 | 655546 | 374.1 |
| 300-0.1-50-50 | 300 | 4476 | 15000 | 1440756 | 994.2 |
| 400-0.1-50-50 | 400 | 7958 | 20000 | 2579813 | 2093.1 |
| 500-0.1-50-50 | 500 | 12518 | 25000 | 403818 | 3694.5 |
| 100-0.1-50-50-nonSC | 100 | 511 | 5000 | 165292 | 25.9 |
| 200-0.1-50-50-nonSC | 200 | 2034 | 10000 | 655546 | 217.2 |
| 300-0.1-50-50-nonSC | 300 | 4476 | 15000 | 1440756 | 293.3 |
| 400-0.1-50-50-nonSC | 400 | 7958 | 20000 | 2579813 | 988.0 |
| 500-0.1-50-50-nonSC | 500 | 12518 | 25000 | 403818 | 1893.2 |

Finally, we created a testset that, in terms of dependency graph size and density as well as subset size, approximates the structure of the real-world railway timetabling application studied in the next section. Table 4 states the results. First, we want to emphasize that the smaller subset size allows us to solve the problem for much larger graphs. Further, the results confirm that the computation time grows linearly with the number of bipartite subgraphs and is generally lower for non-staircase instances. On the non-staircase instances of this testset we could once again observe significant variance of solution times. The greatest outliers arose for 250000-0.00002-10-10-nonSC, where solution time ranged from 463.5 seconds to 1573.7 seconds, an almost four-fold increase between the fastest and slowest recognition.
Table 2 Performance of sorting algorithm on random instances with varying subset size

| Instance                  | $|V|$ | $|E|$ | $|V|$ | $|E|$ | Algorithm Time (s) |
|---------------------------|-----|-----|-----|-----|------------------|
| 100-0.1-50-50             | 100 | 511 | 5000| 165292| 72.0             |
| 100-0.1-100-100           | 100 | 507 | 10000| 443142| 486.8            |
| 100-0.1-150-150           | 100 | 506 | 15000| 808059| 1442.6           |
| 100-0.1-200-200           | 100 | 481 | 20000| 1192116| 3111.3          |
| 100-0.1-250-250           | 100 | 490 | 25000| 1670222| 5991.0          |
| 100-0.1-200-300           | 100 | 472 | 24961| 1588812| 5521.0          |
| 100-0.1-150-350           | 100 | 494 | 25027| 1632905| 6025.6          |
| 100-0.1-100-400           | 100 | 486 | 25794| 1783508| 6816.1          |
| 100-0.1-50-450            | 100 | 499 | 25669| 1729277| 6843.8          |
| 100-0.1-1-500             | 100 | 494 | 25788| 1655008| 6770.1          |
| 100-0.1-50-50-nonSC       | 100 | 511 | 5000| 165292| 25.9            |
| 100-0.1-100-100-nonSC     | 100 | 507 | 10000| 443142| 206.1           |
| 100-0.1-150-150-nonSC     | 100 | 506 | 15000| 808059| 548.0           |
| 100-0.1-200-200-nonSC     | 100 | 481 | 20000| 1192116| 1543.5          |
| 100-0.1-250-250-nonSC     | 100 | 490 | 25000| 1670222| 3721.1          |
| 100-0.1-200-300-nonSC     | 100 | 472 | 24961| 1588812| 3213.3          |
| 100-0.1-150-350-nonSC     | 100 | 494 | 25027| 1632905| 1953.6          |
| 100-0.1-100-400-nonSC     | 100 | 486 | 25794| 1783508| 4817.9          |
| 100-0.1-50-450-nonSC      | 100 | 499 | 25669| 1729277| 1791.3          |
| 100-0.1-1-500-nonSC       | 100 | 494 | 25788| 1655008| 3069.2          |

To summarize, the results show that Algorithm 3 clearly outperforms the MIP if model build time is considered. If model build time is disregarded then, at least on our test set consisting of very small instances, the MIP outperforms our Python-based implementation of Algorithm 3. Further, computation times scale, in accordance with our theoretical results, linearly with number of bipartite subgraphs $G_{ij}$ and quadratically with subset size. Finally, computation times tend to be shorter on non-staircase instances as the algorithm can terminate on finding the first conflict.

5.2 Computational Experiments on Timetabling Instances

Now, we evaluate the performance of Algorithm 3 on a set of real-world railway timetabling instances. In the application, we are given a timetable draft, and the task is to slightly adjust the departure times to reduce peak power consumption, which typically accounts for 20–25% of the train operator company’s electricity bill. For full details of the application and different solution approaches, we refer to [BMS17], [BGMS18] and [BMS20], as well as [BGM+20a] and [BGM20b] for an extension of the approach onto underground timetabling. In [BGMS18], it has been shown that explicit knowledge of a staircase ordering allows the usage of a compact formulation to efficiently solve the timetabling problem and drastically reduce solution times. In the case of the largest instance, containing all nationwide passenger traffic of Germany’s 

Table 4 Performance of sorting algorithm on random instances with large sparse dependency graph

| Instance                  | $|V|$ | $|E|$ | $|V|$ | $|E|$ | Algorithm Time (s) |
|---------------------------|-----|-----|-----|-----|------------------|
| 1000000-0.00002-10-10     | 1000000| 116292| 1000000| 3745465| 273.5            |
| 1500000-0.00002-10-10     | 1500000| 329991| 1500000| 7499847| 580.6            |
| 2000000-0.00002-10-10     | 2000000| 436666| 2000000| 12994694| 1042.3          |
| 2500000-0.00002-10-10     | 2500000| 626913| 2500000| 20180041| 1696.4          |
| 3000000-0.00002-10-10     | 3000000| 901312| 3000000| 29000083| 2564.7          |
| 1000000-0.00002-10-10-nonSC | 1000000| 116292| 1000000| 3745465| 107.3            |
| 1500000-0.00002-10-10-nonSC| 1500000| 329991| 1500000| 7499847| 404.6            |
| 2000000-0.00002-10-10-nonSC| 2000000| 436666| 2000000| 12994694| 561.2            |
| 2500000-0.00002-10-10-nonSC| 2500000| 626913| 2500000| 20180041| 939.5            |
| 3000000-0.00002-10-10-nonSC| 3000000| 901312| 3000000| 29000083| 1420.5           |
largest railway company over a whole day, the computation time has been reduced from about three hours to about three minutes.

In our test, we simulated the lack of explicit knowledge of the staircase ordering by randomizing the vertex names as described for the random instances in the previous section and also tested five randomizations for each instance. As not all bipartite subgraphs \( G_{ij} \) were connected in these instances, some additional presolve was required. In particular, a vertex with degree zero in any \( G_{ij} \) can be removed entirely as it cannot be part of a feasible solution, i.e. a valid timetable. After removing all isolated vertices, the condition that all \( G_{ij} \) are connected is fulfilled and we can directly apply Algorithm \[3\].

We have repeated the computations in [BGMS18, Table 1] with our up-to-date version of the instances and the soft- and hardware specified at the beginning of this section. The column \( \Delta(s) \) in Table \[5\] states the time saved if the problem is solved by first applying the aforementioned presolve method (Presol.), then sorting the graphs using Algorithm \[3\] (Alg.) and finally using the dual-flow formulation (DF) to optimize the schedule, instead of directly solving the problem via the naive formulation (NA). In this test, we even used an improved version of the naive formulation, see [BGM20b, Formulation (1)], which performs slightly better than the version used in [BGMS18]. On most instances in the test set, our three-step approach clearly outperforms solving the problem via the (improved) naive formulation. Most notably, we are able to solve the instance \( \text{Würzburg Hbf} \) in less than 30 minutes, whereas the naive formulation runs into the time limit of 10 hours.

**Table 5 Comparison of optimization performance on railway instances**

| Instance            | \(|V|\)  | \(|E|\) | \(|V|\)  | \(|E|\) | NA  | DF  | Presol. | Alg. | \(\Delta(s)\) |
|---------------------|--------|--------|--------|--------|-----|-----|---------|-----|-------------|
| Bayreuth Hbf        | 327    | 306    | 2287   | 8668   | 0.8 | 0.2 | 0.1     | 0.5 | 0.0         |
| Zell                | 762    | 822    | 5329   | 23882  | 9.7 | 0.4 | 0.3     | 1.2 | 7.8         |
| Passau              | 1040   | 1114   | 7273   | 31673  | 162.4 | 8.6 | 0.4     | 1.6 | 151.8       |
| Jena Paradies       | 1102   | 1229   | 7712   | 35372  | 89.0 | 4.2 | 0.5     | 1.8 | 82.5        |
| Lichtenfels         | 1650   | 1944   | 11546  | 56012  | 1914.3 | 14.6 | 0.8     | 3.0 | 1895.9      |
| Kiel Hbf            | 2130   | 2582   | 14885  | 75783  | 60.9 | 2.6 | 1.0     | 4.0 | 53.3        |
| Erlangen            | 2969   | 4056   | 20769  | 0.12M  | 1769.9 | 48.9 | 2.1     | 5.8 | 1713.1      |
| Aschaffenburg       | 3463   | 4467   | 24227  | 0.13M  | 198.2 | 2.4  | 1.8     | 7.0 | 180.7       |
| Bamberg             | 3644   | 4481   | 25495  | 0.13M  | 843.1 | 15.8 | 1.8     | 6.9 | 818.6       |
| Würzburg Hbf        | 4456   | 6041   | 31165  | 0.13M  | 36003.6 | 1741.5 | 2.7   | 100 | 34249.4     |
| Ulm Hbf             | 5729   | 7608   | 40077  | 0.22M  | 122.9 | 6.6  | 3.3     | 12.5 | 100.5       |
| Leipzig Hbf (tief)  | 6810   | 18592  | 47631  | 0.56M  | 5.6  | 0.7  | 20.9   | 1.7  | 19.7        |
| Dresden             | 6936   | 11853  | 48522  | 0.35M  | 2610.0 | 70.3 | 9.2     | 12.4 | 2518.3      |
| Frankfurt (Main)    | 8626   | 13343  | 60332  | 0.40M  | 361.0 | 15.3 | 9.5     | 16.0 | 320.2       |
| Stuttgart Hbf       | 11594  | 46184  | 81112  | 1.40M  | 331.3 | 8.0  | 51.2    | 23.7 | 248.4       |
| Nürnberg             | 12189  | 20995  | 85258  | 0.63M  | 179.8 | 5.7  | 20.1    | 19.9 | 134.1       |
| Hamm-Altona(S)      | 12373  | 60939  | 65556  | 1.93M  | 528.5 | 6.3  | 95.5    | 16.7 | 410.0       |
| Berlin Hbf(S)       | 16114  | 0.10M  | 0.11M  | 3.15M  | 33.4 | 5.2  | 126.6   | 0.9  | -99.3       |
| Regio Nord          | 13379  | 20670  | 93582  | 0.61M  | 204.6 | 8.1  | 15.0    | 24.4 | 157.1       |
| Regio Nordost       | 16496  | 24760  | 90859  | 0.74M  | 846.8 | 10.2 | 17.4    | 33.6 | 785.6       |
| S-Bahn Hamburg      | 17533  | 89859  | 0.12M  | 2.85M  | 749.8 | 12.5 | 154.3   | 24.7 | 558.3       |
| Regio Südwest        | 24191  | 36125  | 0.17M  | 1.07M  | 244.5 | 6.5  | 28.0    | 34.3 | 175.7       |
| Regio Hessen         | 25092  | 80859  | 0.18M  | 2.48M  | 42.2  | 12.2 | 116.4   | 9.5  | -95.9       |
| Regio BW             | 30172  | 74513  | 0.21M  | 2.24M  | 2225.5 | 16.7 | 82.5    | 57.8 | 2068.5      |
| Regio Südost         | 31917  | 55003  | 0.22M  | 1.64M  | 508.4 | 14.8 | 48.1    | 37.4 | 408.1       |
| Regio NRW            | 47026  | 0.11M  | 0.33M  | 3.48M  | 608.2 | 24.3 | 169.7   | 36.9 | 377.3       |
| Regio Bayern         | 49262  | 0.15M  | 0.34M  | 4.66M  | 379.6 | 21.9 | 236.0   | 38.3 | 83.4        |
| S-Bahn Berlin        | 53353  | 0.35M  | 0.37M  | 10.99M | 541.1 | 47.0 | 474.4   | 4.0  | 15.7        |
| Fernverkehr          | 7053   | 11455  | 49233  | 0.35M  | 58.0  | 17.7 | 6.6     | 17.0 | 16.7        |
| Regionalverkehr      | 0.31M  | 1.01M  | 21.85M | 31.25M | 5824.6 | 1965.3 | 1467.9 | 200.7 | 2190.7      |
| Deutschland          | 0.32M  | 1.04M  | 22.07M | 32.43M | 2350.4 | 1558.6 | 1634.1 | 173.7 | -1016.0     |

We would like to note that in [BGM20b] each scheduled departure could also choose from a discrete set of travel times, which changes the structure of the problem such that a large portion of the subgraphs \( G_{ij} \) no longer fulfil the staircase property. We ran preliminary tests on these
instances and our algorithm determined all instances to be non-staircase within less than one second.

6 Conclusions

In this work, we have discussed theoretical aspects of the recognizability of staircase compatibility and derived polynomial-time algorithms to solve this recognition problem. Furthermore, we demonstrated the practical applicability of the latter. For the special case of bipartite graphs we first showed that a given ordering can be verified to be staircase in time $O(|V| + |E|)$. Moreover, staircase orderings on bipartite graphs are unique up to three obvious operations on the ordering and can be computed in time $O(|V_1|(|V| + |E|))$. We then generalized the recognition problem onto $m$-partite graphs, where it becomes NP-hard in the general case. Further, we presented the polynomial subcase, where all bipartite subgraphs $G_{ij}$ are connected and stated an algorithm that solves the decision problem in polynomial time by constructing a staircase ordering if one exists. Finally, we benchmarked the algorithm against solving the sorting problem via the MIP formulation presented in [Mer17] on small randomly generated instances, tested the behaviour of our algorithm on a set of larger randomly generated instances and eventually used it to find a staircase ordering on a set of real-world timetabling instances. From our tests on small artificial instances, the MIP approach turned out to be impractical due to the long time the models take to construct, while Algorithm 5 does not suffer from such weaknesses. On the real-world timetabling instances, our results show that applying Algorithm 5 and then solving the problem with a reformulation exploiting the staircase ordering, is significantly faster than solving the problem with the formulations available when the staircase ordering is not known.

In the future, it would be interesting to study whether the dual-flow formulations perform well on instances where no staircase ordering exists, but adding a low number of edges fulfils the staircase property. As Theorem 4.6 shows that finding an ordering with minimal staircase relaxation remains NP-complete even if all subgraphs $G_{ij}$ are connected, it is especially interesting to study whether there exist polynomially solvable subcases of this problem.

References

[ABG+20] Janniele A. S. Araujo, Haroldo G. Santos Bernard, Gendron, Sanjay Dominik Jena, Samuel S. Brito, and Danilo S. Souza. Strong bounds for resource constrained project scheduling: Preprocessing and cutting planes. Computers & Operations Research, 113, 2020.

[BBPP99] Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. Handbook of Combinatorial Optimization: Supplement Volume A, chapter The Maximum Clique Problem, pages 1–74. Springer, 1999.

[BGM+20a] Andreas Bärmann, Patrick Gemander, Alexander Martin, Maximilian Merkert, and Frederik Nöth. Success Stories of Industrial Mathematics in Germany, chapter Energy-Efficient Timetabling in a German Underground System. Springer, 2020. To appear, available under: http://www.optimization-online.org/DB_FILE/2020/04/7728.pdf

[BGM20b] Andreas Bärmann, Patrick Gemander, and Maximilian Merkert. The clique problem with multiple-choice constraints under a cycle-free dependency graph. Discrete Applied Mathematics, 283:59–77, 2020.
Andreas Bärmann, Thorsten Gellermann, Maximilian Merkert, and Oskar Schneider. Staircase compatibility and its applications in scheduling and piecewise linearization. *Discrete Optimization*, 29:111–132, 2018.

Andreas Bärmann, Alexander Martin, and Oskar Schneider. A comparison of performance metrics for balancing the power consumption of trains in a railway network by slight timetable adaptation. *Public Transport*, 9(1-2):95–113, 2017.

Andreas Bärmann, Alexander Martin, and Oskar Schneider. Efficient formulations and decomposition approaches for power peak reduction in railway traffic via timetabling. *Transportation Science*, 2020. To appear, available under www.optimization-online.org/DB_HTML/2020/09/7996.html.

Ralf Borndörfer. *Aspects of Set Packing, Partitioning, and Covering*. PhD thesis, TU Berlin, 1998.

Samuel Souza Brito and Haroldo Gambini Santos. Preprocessing and cutting planes with conflict graphs. https://arxiv.org/pdf/1909.07780.pdf, 2019.

Robert Fourer. Solving staircase linear programs by the simplex method, 1: Inversion. *Mathematical Programming*, 1982.

Robert Fourer. Solving staircase linear programs by the simplex method, 2: Pricing. *Mathematical Programming*, 1982.

Robert Fourer. Staircase matrices and systems. *SIAM Review*, 26(1):1–70, 1984.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

Adalat Jabrayilov and Petra Mutzel. New integer linear programming models for the vertex coloring problem. In *Latin American Symposium on Theoretical Informatics*, pages 640–652. Springer, 2018.

Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*, pages 85–103. Springer US, Boston, MA, 1972.

Frauke Liers and Maximilian Merkert. Structural investigation of piecewise linearized network flow problems. *SIAM Journal on Optimization*, 26(4):2863–2886, 2016.

Maximilian Merkert. *Solving Mixed-Integer Linear and Nonlinear Network Optimization Problems by Local Reformulations and Relaxations*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2017.

Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979.
Acknowledgements

We acknowledge financial support by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics – Data – Applications (ADA-Center) within the framework of BAYERN DIGITAL II. Moreover, we thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for their support within GRK 2297 Math-CoRe as well as Project Z01 in CRC TRR 154.