Mixed-Integer Reformulations of Resource-Constrained Two-Stage Assignment Problems

Lena Hupp[c], Manu Kapolke[a], Frauke Liers[a], Alexander Martin[a,d], Robert Weismantel[b]

[a] *Friedrich-Alexander-Universität Erlangen-Nürnberg, Discrete Optimization, Cauerstraße 11, 91058 Erlangen*

[b] *ETH Zürich, Institut für Operations Research, Rämistrasse 101, 8092 Zürich*

[c] *Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 München*

[d] *Fraunhofer Institute for Integrated Circuits IIS, Nordostpark 84, 90411 Nürnberg, Germany*

Abstract

The running time for solving a mixed-integer linear optimization problem (MIP) strongly depends on the number of its integral variables. Bader et al. [Math. Progr. 169 (2018) 565–584] equivalently reformulate an integer program into an MIP that contains a reduced number of integrality constraints, when compared to the original model. Generalizing the concept of totally unimodular (TU) matrices, the reformulation is determined via a so-called affine TU decomposition of the underlying constraint matrix. In this work, we develop affine TU decompositions for two-stage resource-constrained b-matching problems that have challenging applications in runway utilization of aircraft. Mathematically, the task consists in determining an optimum two-stage bipartite b-matching in a graph where a node v can be assigned up to $b_v \in \mathbb{N}$ many edges. The two stages are linked by resource restrictions modeled by specific knapsack constraints. Determining an affine TU decomposition for this problem is reduced to the question of how the incidence matrix of a bipartite graph can be enlarged by appending a submatrix such that the resulting matrix again is TU. Sufficient conditions are derived for the structure of such a submatrix. We apply our findings to two-stage b-matching problem with one resource constraint. For several resource constraints, the TU requirements are not met in general. Nevertheless, in case $b_v = 1$ for all nodes $v \in V$, we reformulate the problem with one resource constraint per node such that the underlying polytope is integral. This again leads to a reduced number of integrality constraints, when compared to the original formulation. The computational study focuses on the aircraft application. Using state-of-the-art MIP solvers, it is shown that, especially for the case with few resource constraints, the reformulated models usually lead to considerably shorter running times, when compared to solving the original formulation.

Keywords:
Mixed-integer programming reformulation, affine TU decomposition, total unimodularity, assignment problem, b-matching, knapsack constraints, air traffic management, runway utilization

November 3, 2020

1. Introduction

Real-world optimization problems can often be modeled as mixed-integer linear programs (MIPs). Although MIPs are $\mathcal{NP}$-hard in theory, recent years have witnessed enormous progress in solving them efficiently. Nowadays, large MIPs can be solved by state-of-the-art software. However, the performance strongly scales with the number of integer variables, and it is advantageous to keep their number small.

Recently, theoretical work by Paat et al. [1] has derived upper bounds for the number of integrality constraints needed to solve an integer program (IP) via an MIP relaxation. In the publication by Bader et al. [2], it was investigated how general IPs can be reformulated equivalently as MIPs, with a reduced number of integrality constraints. The findings presented here are largely based on [2]. We develop strategies for achieving a low number of integrality constraints, where the focus is on problems with two-stage assignment substructures. In this context, the concept of totally unimodular matrices is important. A matrix is *totally unimodular (TU)* if each quadratic submatrix has determinant with a value in $\{-1, 0, 1\}$. Hoffman and Kruskal [3] showed that a polytope $\{x \in \mathbb{R}^n \colon Ax \leq b\}$ has integral vertices if $b \in \mathbb{Z}^m$ and $A \in \mathbb{R}^{m\times n}$ is TU. Thus, if the constraint matrix of an IP is TU and the right-hand side is integral, solving the linear programming (LP) relaxation leads to an integral optimum solution.

Bader et al. [2] generalized total unimodularity to so-called affine TU decompositions. A matrix $A \in \mathbb{Z}^{m\times n}$ admits a *k-row affine TU decomposition* if there exist matrices $U \in \mathbb{Z}^{m\times k}$, $W \in \mathbb{Z}^{k\times n}$, and $\tilde{A} \in \mathbb{Z}^{m\times n}$ such that

$$A = \tilde{A} + UW, \quad \text{where } \tilde{A}_W := \begin{pmatrix} \tilde{A} \\ W \end{pmatrix} \text{ is totally unimodular.}$$

It is shown for an affine TU decomposition with $W \in \{0, \pm 1\}^{k\times n}$ and $b \in \mathbb{Z}^m$ that $\operatorname{conv}(\{x \in \mathbb{R}^n \colon Ax \leq b,\, Wx \in \mathbb{Z}^k\})$ is an integral polyhedron, and

$$\operatorname{conv}\left(\{x \in \mathbb{Z}^n \colon Ax \leq b\}\right) = \operatorname{conv}\left(\left\{x \in \mathbb{R}^n \colon Ax \leq b,\, Wx \in \mathbb{Z}^k\right\}\right). \tag{1}$$

This allows to reformulate the integer program

$$\max\left\{c^\top x \colon Ax \leq b,\, x \in \mathbb{Z}^n\right\}$$

as a mixed-integer program with k integrality constraints

$$\max\left\{c^\top x \colon Ax \leq b,\, x \in \mathbb{R}^n,\, Wx \in \mathbb{Z}^k\right\}. \tag{2}$$

The goal is to determine a matrix W such that k is considerably smaller than n.

One way to solve (2) is to write the restriction $Wx \in \mathbb{Z}^k$ in an extended variable space by setting $Wx = z$, $z \in \mathbb{Z}^k$, and to solve the resulting MIP expressed in $n + k$ variables. For unimodular W, the Hermite normal form transformation can be applied, and model (2)

Email addresses: lena.hupp@siemens.com (Lena Hupp), manu.kapolke@fau.de (Manu Kapolke), frauke.liers@fau.de (Frauke Liers), alexander.martin@fau.de (Alexander Martin), robert.weismantel@ifor.math.ethz.ch (Robert Weismantel)

can be converted into an MIP with n variables in total, from which k must be integral, see [2].

The contribution of this paper is to concretize the general approach [2] to one of the main challenges in Air Traffic Management (ATM), namely an efficient planning of runway utilization. In the so-called pre-tactical planning phase, on the first stage each aircraft needs to get assigned a time window of, say, several minutes, where it is expected to land. A time window can get assigned to at most a certain pre-defined number of aircraft. In case of disturbance (given as a finite number of scenarios), on the second stage some aircraft may have to be rescheduled to different time windows. For generating fair schedules, reschedulings need to be restricted appropriately.

Mathematically, this task can be described as a two-stage b-matching problem on a bipartite graph with additional constraints. To this end, let $G = (I \cup J, E)$ be a bipartite graph with nodes $V = I \cup J$, edges $e = (i, j) \in E \subseteq I \times J$ and edge weights $c_{ij} \in \mathbb{R}$. A *b-matching* is a subset of edges $M \subseteq E$ such that each node $v \in V$ is incident to at most $b_v \in \mathbb{Z}_+$ many edges from M. The constraint matrix $A = (a_{ve}) \in \mathbb{R}^{|V|\times|E|}$ of a bipartite b-matching IP is the node-edge *incidence matrix* of G, where $a_{ve} = 1$ if node $v \in V$ is incident to edge $e \in E$ and $a_{ve} = 0$, otherwise. As such a matrix is TU [4], the bipartite b-matching problem can be solved in polynomial time.

Let $F_1, \dots, F_L \subseteq E$ be $L \in \mathbb{N}$ subsets of edges. The *resource-constrained bipartite b-matching problem* (RCMP) with L knapsack or resource constraints is given by the IP formulation

$$\max \left\{ c^\top x \colon x \in \mathcal{P}_{\mathrm{RCMP}} \cap \mathbb{Z}^{|E|} \right\} \tag{RCMP}$$

with

$$\mathcal{P}_{\mathrm{RCMP}} = \Bigg\{ x \in \mathbb{R}^{|E|} \colon \sum_{\substack{j \in J,\\ (i,j)\in E}} x_{ij} \le b_i, \qquad i \in I, \tag{3}$$

$$\sum_{\substack{i \in I,\\ (i,j)\in E}} x_{ij} \le b_j, \qquad j \in J, \tag{4}$$

$$\sum_{(i,j)\in F_l} r_{ij} x_{ij} \le d_l, \qquad l = 1, \dots, L \tag{5}$$

$$0 \le x_e \le 1, \qquad e \in E \Bigg\}, \tag{6}$$

and b_i, b_j, $d_l \in \mathbb{N}$ and $r_{ij} \in \mathbb{Z}$. The *degree inequalities* (3) and (4) for node sets I and J, respectively, imply that each node $v \in I \cup J$ is incident to at most b_v matching edges.

Together with the knapsack inequalities (5), the constraint matrix is no longer TU in general. Indeed, the RCMP is $\mathcal{NP}$-hard in general, since the 0-1 knapsack problem is contained as a special case [5]. However, [2] successfully reformulated special knapsack problems that share a similar structure with the resource constraints from the ATM application. It is thus natural to generalize the findings from [2] by investigating knapsack problems enlarged by a TU matrix and to analyze appropriate reformulations.

In this paper, we derive suitable affine TU decompositions for RCMP. To this end, it is expedient to study which submatrices can be appended to the incidence matrix of a bipartite graph so that total unimodularity is kept. It turns out that solving the resulting MIP reformulations for the ATM models often yield drastically improved performance. Even in cases where the underlying decomposition is not an affine TU decomposition, it is still advantageous to apply the MIP reformulation.

The remainder of this paper is organized as follows. In Section 2, the derivation of an affine TU decomposition for RCMP is reduced to the question of how the incidence matrix of a bipartite graph can be extended by appending a submatrix so that the resulting matrix remains TU. Sufficient conditions for the structure of such a submatrix are given. In Section 3, the resource-constrained two-stage bipartite b-matching problem from runway utilization at an airport is introduced. For a simplified variant of this problem containing only one resource constraint, an affine TU decomposition is derived in Section 3.1. In Section 3.2, the model is generalized to several resource constraints. For the most realistic setting with one resource constraint for each aircraft, an MIP reformulation with a reduced number of integrality constraints is given. Computational results for the ATM application are presented in Section 4.

2. Affine TU Decompositions of Resource-Constrained Bipartite b-Matching Problems

Let $A \in \mathbb{R}^{(|I|+|J|+L)\times|E|}$ be the constraint matrix that consists of the submatrices $B^{(|I|+|J|)\times|E|}$ and $R \in \mathbb{R}^{L\times|E|}$, where B describes the incidence matrix of a bipartite graph and R denotes the coefficient matrix that belongs to the resource constraints (5). The submatrix B can be further decomposed into the submatrices $A^I \in \mathbb{R}^{|I|\times|E|}$ and $A^J \in \mathbb{R}^{|J|\times|E|}$ that represents the degree inequalities (3) and (4). Thus, we have

$$A = \begin{pmatrix} B \\ R \end{pmatrix} = \begin{pmatrix} A^I \\ A^J \\ R \end{pmatrix}.$$

Since B is totally unimodular, a straightforward approach to derive an affine TU decomposition for A is to consider B as part of $\tilde{A}$ and determine the matrices $\tilde{R}$, $\tilde{U}$ and W, so that

$$A = \tilde{A} + UW = \begin{pmatrix} B \\ \tilde{R} \end{pmatrix} + \begin{pmatrix} 0 \\ \tilde{U} \end{pmatrix} W, \quad \text{with totally unimodular } \tilde{A}_W = \begin{pmatrix} B \\ \tilde{R} \\ W \end{pmatrix}.$$

As a consequence, the basic question is: Which submatrices S can be appended to the incidence matrix B of a bipartite graph such that the matrix

$$T = \begin{pmatrix} B \\ S \end{pmatrix} = \begin{pmatrix} A^I \\ A^J \\ S \end{pmatrix} \tag{7}$$

is totally unimodular?

A similar issue is addressed by Truemper [6] for flow problems, where B in (7) is a matrix with entries in $\{0, \pm 1\}$ and two non-zero entries per column. Matrix S has entries in $\{0, 1\}$ with at most one non-zero entry per column. Here, we will use more general conditions for S that allow more than one non-zero entry in each column. Dell'Amico and Martello [7] introduced the *k-cardinality assignment problem* that asks for determining an assignment of a given size $k \in \mathbb{N}$. In this context, they showed that the constraint matrix in (7) remains TU, if S consists of just one row associated with a cardinality constraint summing up over all edges, i.e., $\sum_{(i,j)\in E} x_{ij} \leq d$. The assignment problem with one cardinality constraint has been intensely studied in the literature [7, 8, 9, 10].

It is easy to see that T in (7) is no longer TU in general if in the cardinality constraint the sum over all edges is replaced by an arbitrary subset of edges $F \subsetneq E$. In the following, we will introduce some sufficient conditions that will be summarized in Theorem 2.1. It turns out that if B is the constraint matrix of a bipartite graph, then a row of S can be derived by summing up rows of either A^I or A^J to obtain again a totally unimodular matrix T. Similarly, rows of either A^I or A^J associated with different connected components of the bipartite graph may be subtracted from each other to generate a new row of S. (Clearly, for general TU matrices such operations do not always maintain total unimodularity.)

More formally, let a bipartite graph $G = (I \cup J, E)$ consist of N connected components $G = G_1 \cup \ldots \cup G_N$ with $G_s = (I_s \cup J_s, E_s)$ for $s = 1, \ldots, N$. In addition, for all $s \neq t$, let $I_s \cap I_t = \emptyset$, $J_s \cap J_t = \emptyset$ and $E_s \cap E_t = \emptyset$. This choice is motivated by two-stage assignment problems where each of a finite set of second-stage scenarios corresponds to a connected component of G, see Section 3. For $s = 1, \ldots, N$, let $I_{s,1} \dot\cup \ldots \dot\cup I_{s,n_I^s} \subseteq I_s$ be a union of n_I^s disjoint subsets of I_s. As $I_{s,j} = \emptyset$ is allowed, we assume without loss of generality that all disjoint subsets have the same cardinality, i.e., $n_I^s =: n_I \in \mathbb{N}$ for all $s = 1, \ldots, N$. For each component s, we choose a sign $\operatorname{sgn}(s)$, where $\operatorname{sgn}(s) = 0$ is signed positively, and $\operatorname{sgn}(s) = 1$ is signed negatively. We now study constraints formed from row sums or row differences of A^I, i.e.,

$$\sum_{s=1}^{N} (-1)^{\operatorname{sgn}(s)} \sum_{i \in I_{s,l}} \sum_{\substack{j \in J_s \\ (i,j)\in E_s}} x_{ij} \leq d_l, \qquad l = 1, \ldots, n_I,$$

with $d_l \in \mathbb{Z}_+$. Disjointness of the subsets $I_{s,l}$, $l = 1, \ldots, n_I$, means that each row of A^I can occur at most once in a row sum or in a row difference. The corresponding coefficient matrix derived from these constraints is denoted as $P^I \in \{-1, 0, 1\}^{n_I \times |E|}$.

Analogously, let $J_{s,1} \dot\cup \ldots \dot\cup J_{s,n_J} \subseteq J_s$, $s = 1, \ldots, N$, and let $P^J \in \{-1, 0, 1\}^{n_J \times |E|}$ be the coefficient matrix that belongs to the constraints that are row sums or row differences of A^J. Note that the component signs remain fixed concerning both matrices P^I and P^J. In Theorem 2.1, we show that the matrix S can be generated from rows of P^I and P^J.

Before proceeding to the proof, we first introduce another operation for creating new rows of S that preserve total unimodularity of T. We choose a new row as one of the rows of A^I or A^J and arbitrarily replace one-entries by zero-entries. The same row of A^I or A^J may be used more than once if the edge sets corresponding to the remaining one-entries are disjoint. Formally, for each node $i \in I$ let $J_1^i \dot\cup \ldots \dot\cup J_{n_i}^i \subseteq J$ be a union of n_i disjoint

subsets of node set J. Then, we define $Q^I \in \{0,1\}^{\left(\sum_{i\in I} n_i\right)\times|E|}$ as the constraint matrix associated with the cardinality constraints

$$\sum_{\substack{j\in J_l^i:\\(i,j)\in E}} x_{ij} \leq d_l, \qquad l = 1,\dots,n_i,\ i \in I.$$

Analogously, for each node $j \in J$, we consider a disjoint union $I_1^j \dot\cup \dots \dot\cup I_{n_j}^j \subseteq I$, where $Q^J \in \{0,1\}^{\left(\sum_{j\in J} n_j\right)\times|E|}$ denotes the coefficient matrix of the cardinality constraints.

The following theorem shows that together with the incidence matrix of a bipartite graph, the matrices P^I, P^J, Q^I, and Q^J form a totally unimodular matrix.

Theorem 2.1. *Let $G = (I \cup J, E)$ be a bipartite graph with N connected components. Further, let B describe its incidence matrix that consists of submatrices A^I and A^J. Then, with P^I, P^J, Q^I and Q^J defined as above, the matrix*

$$\mathcal{A} = \begin{pmatrix} A^I \\ A^J \\ P^I \\ P^J \\ Q^I \\ Q^J \end{pmatrix}$$

is totally unimodular.

Proof. The proof uses the TU characterization of Ghouila-Houri [11]. To this end, we show that each (arbitrary) collection of rows of $\mathcal{A}$ can be split into two parts, such that the sum of the rows in one part, minus the sum of the rows in the second part, is a vector with entries in $\{-1,0,+1\}$. We assign $(+1)$ and (-1) to the rows to indicate to which part they belong. Let $\overline{A}^I, \overline{A}^J, \overline{P}^I, \overline{P}^J, \overline{Q}^I, \overline{Q}^J$ be the submatrices consisting of an arbitrary collection of rows of $\mathcal{A}$, with corresponding submatrices $\overline{A}_s^I, \overline{A}_s^J, \overline{P}_s^I, \overline{P}_s^J, \overline{Q}_s^I, \overline{Q}_s^J$ for the components $s = 1,\dots,N$, i.e.,

$$\begin{pmatrix} \overline{A}^I \\ \overline{A}^J \\ \overline{P}^I \\ \overline{P}^J \\ \overline{Q}^I \\ \overline{Q}^J \end{pmatrix} = \begin{pmatrix} \overline{A}_1^I & \overline{A}_2^I & \cdots & \overline{A}_N^I \\ \overline{A}_1^J & \overline{A}_2^J & \cdots & \overline{A}_N^J \\ \overline{P}_1^I & \overline{P}_2^I & \cdots & \overline{P}_N^I \\ \overline{P}_1^J & \overline{P}_2^J & \cdots & \overline{P}_N^J \\ \overline{Q}_1^I & \overline{Q}_2^I & \cdots & \overline{Q}_N^I \\ \overline{Q}_1^J & \overline{Q}_2^J & \cdots & \overline{Q}_N^J \end{pmatrix}.$$

For each $s = 1,\dots,N$, we start by considering $\overline{P}_s^I$ and $\overline{P}_s^J$. If either $\overline{P}_s^I = \overline{P}_s^J = \emptyset$, or if $\overline{P}_s^I$ and $\overline{P}_s^J$ only have entries in $\{0,1\}$, The component s is called *positively signed*. Otherwise, $\overline{P}_s^I$ and $\overline{P}_s^J$ only have entries in $\{-1,0\}$, and the component s is *negatively signed*.

We always assign $(+1)$ and (-1) to $\overline{A}^I, \overline{P}^I$ and $\overline{Q}^I$ in the following way and order: We assign $(+1)$ to all rows from $\overline{P}^I$. For each row vector $a^I = (a^I_1, \ldots, a^I_N)$ chosen from $\overline{A}^I$, we consider the submatrix

$$C = \begin{pmatrix} \overline{P}^I \\ a^I \end{pmatrix}.$$

Assume a^I has non-zero entries in a^I_s. If the component s is positively signed and C contains columns with exactly two $+1$ entries, we assign (-1) to a^I, otherwise $(+1)$. If component s is negatively signed and C contains columns with exactly two non-zero entries (one $+1$ and one -1), we assign $(+1)$ to a^I, otherwise (-1). For each row vector $q^I = (q^I_1, \ldots, q^I_N)$ chosen from $\overline{Q}^I$, we consider the submatrix

$$C = \begin{pmatrix} \overline{P}^I \\ \overline{A}^I \\ q^I \end{pmatrix}.$$

Assume q^I has non-zero entries in q^I_s. If the component s is positively signed, we proceed as follows: If C contains only columns with at most one $+1$ entry or it contains columns with exactly three $+1$ entries, we assign $(+1)$ to q^I. If C contains columns with exactly two $+1$ entries and no columns with three $+1$ entries, we assign (-1) to q^I. If the component s is negatively signed, we proceed as follows: If C contains only columns with at most one non-zero entry or it contains columns with exactly three non-zero entries, we assign (-1) to q^I. If C contains columns with exactly two non-zero entries and no columns with three non-zero entries, we assign $(+1)$ to q^I.

Thus, summing up the rows of the matrices $\overline{A}^I, \overline{P}^I$, and $\overline{Q}^I$ results in a row vector $g^\top = (g^\top_1, g^\top_2, \ldots, g^\top_N)$ with $g \in \{-1, 0, 1\}^{|E|}$. As a consequence, $g^\top_s$ solely has 0 and 1 entries if it corresponds to a positively signed component s. A negatively signed component $g^\top_s$ has 0 and -1 entries. We use an analogous strategy for the collection of rows in $\overline{A}^J, \overline{P}^J$, and $\overline{Q}^J$, which results in a row vector $h^\top = (h^\top_1, h^\top_2, \ldots, h^\top_N)$ with $h \in \{-1, 0, 1\}^{|E|}$. Moreover, $h^\top_s$ has -1 and 0 entries if it corresponds to a positively signed component s. A negatively signed component $h^\top_s$ has 0 and 1 entries. If one or several submatrices $\overline{A}^I, \overline{A}^J, \overline{P}^I, \overline{P}^J, \overline{Q}^I$, or $\overline{Q}^J$ are the empty set, we use the same strategy. Obviously, $g^\top + h^\top$ yields a vector with entries in $\{-1, 0, 1\}$. □

3. Two-Stage *b*-Matching with an Application to Optimized Runway Utilization at an Airport

In this section, we study MIP reformulations for a current challenge in the ATM area, namely the optimization of runway utilization under uncertainty. Each aircraft needs to be assigned one arrival or departure time slot such that the overall delay is minimized (e.g., see [12, 13, 14, 15]). Uncertainties due to weather or technical constraints often lead to deviations from a previously optimal schedule, which then may turn out to be infeasible. In practice, plan adjustments are usually made in hindsight, and efficient algorithms that generate stable and cost-efficient schedules are yet largely missing.

Some matching-based approaches under uncertainty have been studied in the literature (e.g., [16, 17, 18, 19]). A matching-based MIP model for optimized runway utilization that addresses uncertainties in the initial calculation of the plans is introduced by Kapolke et al. [14]. The focus is on the so-called pre-tactical planning phase until around 30 minutes before landing, where aircraft are assigned to time windows of several minutes. Thus, time is discretized into time windows of fixed size. Every aircraft has a feasible time interval from its earliest possible landing time to its latest possible landing time and can be assigned to all discrete time windows within this interval. Each aircraft is assigned to exactly one time window, whereas several aircraft can be scheduled to the same time window. The scheduling is implemented in two decision stages. At the first stage, an assignment of each aircraft to a time window is determined here-and-now. Afterwards, the uncertainty from a finite set of disturbed scenarios $S_1, \dots, S_N$, $N \in \mathbb{N}$, is revealed, where each scenario defines feasible aircraft time intervals. Hence, reassignments are performed at the second stage if necessary.

This optimization task can be modeled as follows. Let $G = (\mathcal{I} \cup \mathcal{J}, E)$ be the associated bipartite graph with $N + 1$ components $G_s = (I_s \cup J_s, E_s)$, $s = 0, \dots, N$, where each component corresponds to a scenario. G_0 is the nominal first-stage scenario. The nodes $I_0 = \{1, \dots, |I_0|\}$ represent the aircraft. Node set J_0 corresponds to time windows, where nodes are labeled chronologically as $1, \dots, |J_0|$ (e.g., they represent a time horizon of 24 hours discretized into time windows of 10 minutes). An edge $(i, j) \in E_0$ indicates that aircraft $i \in I_0$ may be assigned to a time window $j \in J_0$ in the first-stage scenario. Since we assume to have the same set of aircraft and time windows in each scenario, the node sets of G_s for $s = 1, \dots, N$ are copies of G_0. For ease of presentation, subscripts will be omitted in case no confusion is possible, i.e., we write $I := I_s = \{1, \dots, |I_0|\}$ and $J := J_s = \{1, \dots, |J_0|\}$ for $s = 0, \dots, N$.
The edge sets E_s, $s = 0, \dots, N$, however, are different in each scenario as they correspond to the feasible time intervals of the aircraft. Since a feasible time interval is not interrupted, the edge sets have the following property for all $i \in I$ and $j, k, l \in J$, $j \leq l \leq k$:

$$(i, j),\ (i, k) \in E_s \quad \Longrightarrow \quad (i, l) \in E_s. \tag{8}$$

This means that if an aircraft can be scheduled to two different time windows, it can also be assigned to all time windows in between.

In each of the $|N| + 1$ scenarios, each aircraft i has to be assigned to exactly one time window. At most b_j^s aircraft may be scheduled to time window j in scenario S_s, for some $b_j^s \in \mathbb{N}$. With $c_{ij}^0 \in \mathbb{R}$ we denote the cost or deviation from the originally scheduled time that is caused when an aircraft i is assigned to the time window j in the first stage. Analogously, $c_{ij}^s \in \mathbb{R}$ indicates the second-stage costs for each disturbed scenario $s = 1, \dots, N$.

A model as outlined here often arises in a two-stage stochastic context, where we assume that each disturbed scenario occurs with some probability p_s for $s = 1, \dots, N$. As a natural objective function, we then minimize the sum of delays of the first-stage assignment plus the expected sum of delays of the second-stage assignment.

Then, the task consists in computing a b-matching on the first stage in G_0 before the uncertainty is revealed, together with b-matchings for each of the disturbed scenarios

in G_s, $s = 1, \ldots, N$. The latter might differ from the first-stage matching in G_0, for example if an aircraft is delayed in a disturbed scenario s such that the a priori (first-stage) assignment to a certain time window is not feasible in s.

A highly desirable goal consists in the determination of fair schedules. We therefore limit for each aircraft i the distance of a rescheduling action from a time window j_0 of the first-stage to a time window j_s of the second-stage by $|j_0 - j_s| \leq d_i^s \in \mathbb{N}$, where d_i^s is a suitably chosen input parameter. Binary variables x_{ij} denote the assignment of aircraft i to time window j on the first stage, binary variables y_{ij}^s denote the second-stage assignments for scenarios $s = 1, \ldots, N$. An IP formulation for the *two-stage bipartite b-matching problem for runway optimization* (RunOPT) is given as:

(RunOPT)

$$\min \quad \sum_{e \in E_0} c_e x_e + \sum_{s=1}^{N} p_s \sum_{e \in E_s} c_e^s y_e^s \tag{9}$$

$$\text{s.t.} \quad \sum_{\substack{j \in J \\ (i,j) \in E_0}} x_{ij} = 1, \qquad \sum_{\substack{j \in J \\ (i,j) \in E_s}} y_{ij}^s = 1, \qquad \forall i \in I,\ s = 1, \ldots, N, \tag{10}$$

$$\sum_{\substack{i \in I \\ (i,j) \in E_0}} x_{ij} \leq b_j^0, \qquad \sum_{\substack{i \in I \\ (i,j) \in E_s}} y_{ij}^s \leq b_j^s, \quad \forall j \in J,\ s = 1, \ldots N, \tag{11}$$

$$\mathcal{R}^s \begin{pmatrix} x \\ y^s \end{pmatrix} \leq \overline{d}^s, \qquad \forall s = 1, \ldots, N, \tag{12}$$

$$x_e,\ y_f^s \in \{0, 1\}, \qquad \forall e \in E_s,\ s = 0, \ldots, N. \tag{13}$$

Due to (10), each aircraft is assigned to exactly one time window at both stages, whereas (11) ensures that at most b_j^s aircraft can be scheduled to time window j in scenario $s \in \{0, \ldots, N\}$. For the sake of simplicity, we write J in (11) instead of $J_E := \{j \in J \colon \exists i \in I \text{ with } (i,j) \in E\}$. More precisely, we actually have the first-stage constraints for all $j \in J_{E_0}$ and the second-stage constraints for all $j \in J_{E_s}$, $s = 1, \ldots, N$. This more differentiated perspective will become important in the subsequent sections when it comes to the size of the matrix W. The resource constraints written in general terms as (12) restrict the distance of a replanning action from a time window at the first stage to a time window at the second stage for ensuring fairness. To this end, each $\mathcal{R}^s$ indicates a matrix in $\mathbb{Z}^{2\kappa \times |E|}$ and $\overline{d}^s$ is a vector of appropriate dimension. Here, κ describes the number of resource constraints involved. The factor 2 comes from the fact that the number of constraints is doubled when linearizing the absolute distance.

We next examine different values of κ. In the most realistic model for the ATM application called *classic RunOPT*, each individual aircraft has its own resource constraint, i.e. $\kappa = |I|$. The latter have ascending coefficients in the first- and second-stage variables. Then, (12) reads

$$\pm \Bigg(\sum_{\substack{j \in J \\ (i,j) \in E_0}} j \cdot x_{ij} - \sum_{\substack{j \in J \\ (i,j) \in E_s}} j \cdot y_{ij}^s \Bigg) \leq d_i^s, \qquad \forall i \in I,\ s = 1, \ldots, N. \tag{14}$$

Let A denote the total constraint matrix that corresponds to (RunOPT) and A^I and A^J the constraint matrices associated with (10) and (11). Then A reads

$$A = \begin{pmatrix} (A^I)^\top & (A^J)^\top & R^\top & -R^\top \end{pmatrix}^\top \tag{15}$$

with $A^I \in \mathbb{Z}^{(N+1)|I| \times |E|}$, $A^J \in \mathbb{Z}^{\left(\sum_{s=0}^{N} |J_{E_s}|\right) \times |E|}$ and $R \in \mathbb{Z}^{|I|N \times |E|}$.

To summarize, (RunOPT) is an assignment problem on a bipartite graph with $N+1$ connected components, together with resource constraints ((12)). In the remainder of this paper we will discuss different variants of resource constraints.

3.1. The b-Matching Problem with One Resource Constraint

We first consider a simplified version where in each disturbed scenario only a single resource constraint (12) restricts the entire replanning action as

$$\pm\left(\sum_{i \in I} \sum_{\substack{j \in J \\ (i,j) \in E_0}} j \cdot x_{ij} - \sum_{i \in I} \sum_{\substack{j \in J \\ (i,j) \in E_s}} j \cdot y_{ij}^s \right) \leq d^s, \qquad s = 1, \dots, N. \tag{16}$$

For ease of exposition, we study the special case of the nominal and one representative disturbed scenarios ($N = 1$). The latter can be derived, for example, by using only edges that are present in all scenarios.

A natural approach to reformulate (16) is to aggregate all assignment variables with the same coefficient by introducing an artificial variable z_j for each time window j:

$$z_j = \sum_{\substack{i \in I \\ (i,j) \in E_0}} x_{ij} - \sum_{\substack{i \in I \\ (i,j) \in E_1}} y_{ij}^1, \qquad \forall j \in J_E. \tag{17}$$

With coefficient matrix $W \in \mathbb{Z}^{|J_E| \times |E|}$ we write $\begin{pmatrix} z_1 & \cdots & z_{|J_E|} \end{pmatrix}^\top = W \begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top$. Thus, $W = \begin{pmatrix} \hat{A}_0^J & -\hat{A}_1^J \end{pmatrix}$, where $\hat{A}_0^J$ is derived from A_0^J by adding zero rows at the positions that correspond to time windows $j \in J_{E_1} \setminus J_{E_0}$, and $\hat{A}_1^J$ is derived analogously from A_1^J with zero rows added for $j \in J_{E_0} \setminus J_{E_1}$.

Then, the coefficient matrix R and $-R$ for (16) can be decomposed as

$$\begin{pmatrix} R \\ -R \end{pmatrix} = \begin{pmatrix} j_1 & j_2 & \cdots & j_{|J_E|} \\ -j_1 & -j_2 & \dots & -j_{|J_E|} \end{pmatrix} W = UW$$

with $\{j_1, \dots, j_{|J_E|}\} := J_E$ and A is expressed as

$$A = \begin{pmatrix} A^I \\ A^J \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ U \end{pmatrix} W \text{ with } \tilde{A}_W = \begin{pmatrix} A^I \\ A^J \\ W \end{pmatrix}. \tag{18}$$

With Theorem 2.1, $\tilde{A}_W$ is totally unimodular since W is the difference of two degree inequalities corresponding to two distinct connected components.

Lemma 3.1. *For (RunOPT) with resource constraint* (16) *and* $N = 1$*, an* $|J_E|$*-row affine TU decomposition is given by* (17) *and* (18)*.*

With the above affine TU decomposition at hand, this version of the classic RunOPT can be reformulated to an MIP with only $|J_E|$ integrality constraints instead of $|E| = |E_0|+|E_1|$ many. Without going into further detail, for $N > 1$ many disturbed scenarios, a $\left(\sum_{s=0}^{N} |J_{E_s}|\right)$-row affine TU decomposition can be derived (see Hupp [20]). Next, we discuss the case of more than one resource constraint.

3.2. The b-Matching Problem with Several Resource Constraints
For simplifying the exposition, we again assume $N = 1$.

For $\kappa > 1$ many resource constraints, Theorem 2.1 cannot be applied, since in general the rows of W are not the difference of rows of A^J, and toy examples exist where $\tilde{A}_W$ is not totally unimodular [20].

Yet, we prove in this section that for the classic RunOPT ($\kappa = |I|$) with right-hand sides $b_j^s = 1$, the corresponding aggregation can still be used to formulate an MIP with feasible set described by an integral polyhedron. Thus, although not being based on an affine TU decomposition, this MIP reformulation reduces the number of integrality constraints. Moreover, the computational results in Section 4 show that also solving the reformulated model while maintaining all integrality constraints is beneficial for a not too large number of resource constraints $\kappa \geq 1$.

Suppose I is partitioned into κ subsets $I_1, \ldots, I_\kappa$. Denoting the second-stage variables by $y_{ij} = y_{ij}^1$, the resource constraints (12) read:

$$\pm\Bigg(\sum_{i\in I_p}\sum_{\substack{j\in J\\(i,j)\in E_0}} j\cdot x_{ij} - \sum_{i\in I_p}\sum_{\substack{j\in J\\(i,j)\in E_1}} j\cdot y_{ij}\Bigg) \leq d, \qquad \forall p = 1,\ldots,\kappa. \tag{19}$$

(Case (16) corresponds to $\kappa = 1$, (14) is obtained if $|I_p| = 1$ for all $p \in \{1,\ldots,|I|\}$.) Aggregation (17) is now split according to the different constraints, i.e.,:

$$z_{p,j} = \sum_{\substack{i\in I_p\\(i,j)\in E_0}} x_{ij} - \sum_{\substack{i\in I_p\\(i,j)\in E_1}} y_{ij}, \qquad \forall j \in J_{E(p)},\ p = 1,\ldots,\kappa \tag{20}$$

with $J_{E(p)} := \{j \in J\colon \exists i \in I_p \text{ with } (i,j) \in E\}$. The matrix W described by $z = W\begin{pmatrix} x^\top & y^\top\end{pmatrix}^\top$ has the form

$$W = \Bigg(\operatorname{Diag}\left(\hat{A}_0^J(I_1),\ldots,\hat{A}_0^J(I_\kappa)\right) \qquad \operatorname{Diag}\left(-\hat{A}_1^J(I_1),\ldots,-\hat{A}_1^J(I_\kappa)\right)\Bigg), \tag{21}$$

where $\hat{A}_0^J(I_p)$, $p = 1,\ldots,\kappa$, consists of the columns of $\hat{A}_0^J$ corresponding to edges that contain a node from I_p, and the rows of $\hat{A}_0^J$ correspond to nodes $j \in J_{E(p)}$. Matrix $\hat{A}_1^J(I_p)$ is derived analogously. Thus, $W \in \mathbb{Z}^{\sigma\times|E|}$, where

$$\sigma := \sum_{p=1}^{\kappa} |J_{E(p)}|. \tag{22}$$

Further, the coefficient matrix from (19) can be decomposed into

$$\begin{pmatrix} R \\ -R \end{pmatrix} = \begin{pmatrix} U' \\ -U' \end{pmatrix} W = UW,$$

where

$$U' = \begin{pmatrix} j_1^1 & j_2^1 & \cdots & j_{|J_{E(1)}|}^1 & & & & & & & & \\ & & & & j_1^2 & j_2^2 & \cdots & j_{|J_{E(2)}|}^2 & & & & \\ & & & & & & & & \ddots & & & \\ & & & & & & & & & j_1^\kappa & j_2^\kappa & \cdots & j_{|J_{E(\kappa)}|}^\kappa \end{pmatrix}$$

with $\{j_1^p, \ldots, j_{|J_E(p)|}^p\} \coloneqq J_{E(p)}$ and, therefore, $U \in \mathbb{Z}^{2\kappa \times \sigma}$. With such defined W and U, we obtain an expression of form (18) again:

$$A = \begin{pmatrix} A^I \\ A^J \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ U \end{pmatrix} W \;\text{ with }\; \tilde{A}_W = \begin{pmatrix} A^I \\ A^J \\ W \end{pmatrix}. \tag{23}$$

Although (23) does not provide an affine TU decomposition leading to integral solutions for all possible integral right-hand sides [20], a matrix W as in (21) may still enable an MIP reformulation as in (2), such that the feasible set describes an integral polyhedron for certain right-hand sides.

Next, this is shown to be true for the practically relevant classic RunOPT with $b_j^s = 1$ (for all $j \in J_E,\ s \in \{0, \ldots, N\}$).

From now on, we focus on right-hand side $b_j^s =: b = 1$ for all $j \in J_E$ and $s \in \{0, \ldots, N\}$. Let $\mathcal{C}$ be defined as the set of edges (i,j) that exist both in graph components $G_0 = (I \cup J, E_0)$ and in $G_1 = (I \cup J, E_1)$, i.e., $\mathcal{C} \coloneqq \{(i,j) \colon (i,j) \in E_0 \text{ and } (i,j) \in E_1\}$. For $\kappa = |I|$, aggregation (20) reads

$$\left.\begin{array}{ll} z_{ij} = x_{ij} - y_{ij}, & \forall j \in J : (i,j) \in \mathcal{C}, \\ z_{ij} = x_{ij}, & \forall j \in J : (i,j) \in E_0 \backslash \mathcal{C}, \\ z_{ij} = -y_{ij}, & \forall j \in J : (i,j) \in E_1 \backslash \mathcal{C}, \end{array}\right\} \quad \forall i \in I. \tag{24}$$

The number of aggregation constraints then is

$$\sigma \;=\; \sum_{p=1}^{\kappa} |J_{E(p)}| \;=\; \sum_{i=1}^{|I|} |\{j \in J \colon (i,j) \in E_0 \text{ or } (i,j) \in E_1\}| \qquad = \; |E| - |\mathcal{C}|.$$

In order to show that the corresponding matrix W enables a suitable MIP reformulation, we first indicate the polyhedron that is induced by the LP relaxation of the classic RunOPT:

$$\mathcal{P} = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{|E|} \;:\; \bar{A} \begin{pmatrix} x \\ y \end{pmatrix} \leq \bar{b} \right\},$$

with

$$\bar{A} = \begin{pmatrix} A^I \\ -A^I \\ A^J \\ R \\ -R \\ -I \end{pmatrix} \quad \text{and} \quad \bar{b} = \begin{pmatrix} 1 \\ -1 \\ b \\ d \\ d \\ 0 \end{pmatrix}.$$

The identity matrix I represents the non-negativity constraints $x_{ij} \geq 0$ and $y_{ij} \geq 0$. Then, the *two-stage bipartite b-matching polytope* associated with (RunOPT) is defined as

$$\mathcal{P}_{\text{RunOPT}} = \text{conv}(\mathcal{P} \cap \mathbb{Z}^{|E|}).$$

The following theorem leads to a reformulation of $\mathcal{P}_{\text{RunOPT}}$ with σ many integer variables.

Theorem 3.2. *Let $\mathcal{P}_{RunOPT}$, $\mathcal{P}$, $\mathcal{C}$, σ and W be defined as above. Further, let $b = 1$. Then*

$$\mathcal{P}_W = \text{conv}\left(\left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathcal{P} \,:\, W \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{Z}^\sigma \right\} \right)$$

is an integral polyhedron and $\mathcal{P}_{RunOPT} = \mathcal{P}_W$.

In order to prove Theorem 3.2 we use (23) and write

$$\bar{A} \begin{pmatrix} x \\ y \end{pmatrix} \leq \bar{b} \quad \Longleftrightarrow \quad \begin{pmatrix} A^I \\ -A^I \\ A^J \\ 0 \\ -I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ U \\ 0 \end{pmatrix} W \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} 1 \\ -1 \\ 1 \\ \bar{d} \\ 0 \end{pmatrix} \tag{25}$$

with $\overline{d} = \begin{pmatrix} d^\top & d^\top \end{pmatrix}^\top$. In (25), we set $z = W \begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top \in \mathbb{Z}^\sigma$ and obtain

$$\begin{pmatrix} A^I \\ -A^I \\ A^J \\ 0 \\ -I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} 1 \\ -1 \\ 1 \\ \bar{d} \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \\ U \\ 0 \end{pmatrix} z.$$

Since $0 \leq x_e \leq 1$ for all $e \in E_0$ and $0 \leq y_f \leq 1$ for all $f \in E_1$, definition (24) implies $z_{ij} \in \{-1, 0, 1\}$. We first show that for fixed $z \in \{-1, 0, 1\}^\sigma$ with $\bar{d} - Uz \geq 0$ the following polyhedron is integral:

$$\begin{aligned} \mathcal{P}_z = \Bigg\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{|E|} : \quad & A^I \begin{pmatrix} x \\ y \end{pmatrix} = 1,\ A^J \begin{pmatrix} x \\ y \end{pmatrix} \leq 1,\ x, y \geq 0, \\ & x_{ij} = z_{ij} \qquad \forall\ (i,j) \in E_0 \backslash \mathcal{C}, \\ & x_{ij} - y_{ij} = z_{ij} \qquad \forall\ (i,j) \in \mathcal{C}, \\ & -y_{ij} = z_{ij} \qquad \forall\ (i,j) \in E_1 \backslash \mathcal{C} \Bigg\}. \end{aligned} \tag{26}$$

We show that for fixed z, some x and y variables are implicitly fixed as well. After deleting the corresponding nodes and edges, the integrality of $\mathcal{P}_z$ over the remaining graph can easily be shown.
To this end, a vector $z \in \{-1,0,1\}^\sigma$ is called a *feasible z-configuration* if $\mathcal{P}_z \neq \emptyset$. The following lemma implies that each feasible z-configuration fixes a number of x and y variables uniquely to 0 and 1.

Lemma 3.3. *Let $z \in \{-1,0,1\}^\sigma$ be a feasible z-configuration.*

a) For each $i \in I$, exactly one of the following cases holds:

$(I1)$ $z_{ij} = 0, \quad \forall\, j \in J,$

$(I2)$ $\exists!\,(j,k \in J):\ z_{ij} = 1 \ \wedge\ z_{ik} = -1 \ \wedge\ z_{il} = 0, \quad \forall\, l \in J \setminus \{j,k\}.$

b) For each $j \in J$, exactly one of the following cases holds:

$(J1)$ $z_{ij} = 0, \quad \forall\, i \in I$

$(J2)$ $\exists!\,(i,k \in I):\ z_{ij} = 1 \ \wedge\ z_{kj} = -1 \ \wedge\ z_{lj} = 0, \quad \forall\, l \in I \setminus \{i,k\},$

$(J3)$ $\exists!\,(i \in I):\ z_{ij} \in \{-1,1\} \ \wedge\ z_{lj} = 0, \quad \forall\, l \in I \setminus \{i\}.$

Proof. Let $z \in \{-1,0,1\}^\sigma$ be a feasible z-configuration and $\begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top \in \mathcal{P}_z$.

a) Let $i \in I$. Obviously, $(I2)$ can not hold if $(I1)$ is true.
Now we assume that $(I1)$ does not hold, and show that $(I2)$ follows.

First, we assume $z_{ij} = 1$. With $0 \leq x_{ij}, y_{ij} \leq 1$, we have $1 = z_{ij} = x_{ij} - y_{ij}$ in case $(i,j) \in \mathcal{C}$ and, thus, $x_{ij} = 1$ and $y_{ij} = 0$. If otherwise $(i,j) \in E_0 \setminus \mathcal{C}$, we have $1 = z_{ij} = x_{ij}$. Then, the degree equations imply $x_{il} = 0$ for all $l \in J$, $l \neq j$. Hence, for $l \neq j$, we know $z_{il} = 0$ (if $(i,j) \in E_0 \setminus \mathcal{C}$) or $z_{il} = -y_{il}$, otherwise. With $z_{il} \in \{-1,0,1\}$ and $0 \leq y_{il} \leq 1$, it follows that $y_{il} \in \{0,1\}$. For the fixed index i, the degree equations with respect to y imply that there is exactly one $k \in J \setminus \{j\}$ with $y_{ik} = 1$. Therefore, it is $z_{ik} = -1$ and $z_{il} = 0$ for all $l \in J \setminus \{j,k\}$, so that $(I2)$ holds. The case $z_{ij} = -1$ can be treated similarly.

b) Let $j \in J$. Obviously, if one of the three cases $(J1)$-$(J3)$ applies, the other two cannot apply, because they contradict each other. In order to show that indeed one case applies, we assume w.l.o.g. that $(J1)$ does not hold and show that $(J2)$ or $(J3)$ follows.

First, let $i \in I$ with $z_{ij} = 1$. With the same argumentation as in *a)*, it follows for $l \neq i$ that $z_{lj} = 0$ in case $(i,j) \in E_0 \setminus \mathcal{C}$, or $z_{lj} = -y_{lj}$ otherwise. We also get $y_{lj} \in \{0,1\}$. However, for the fixed index j, the degree equalities regarding y yield that there is *at most* one $k \in I \setminus \{i\}$ with $y_{kj} = 1$. Therefore, we obtain either $z_{kj} = -1$ and $z_{lj} = 0$ for all $l \in I \setminus \{i,k\}$ $(J2)$ or $z_{lj} = 0$ for all $l \in I \setminus \{i\}$ $(J3)$. Similarly, this can be shown for $z_{ij} = -1$. □

For a feasible z-configuration, Lemma 3.3 implies that for each $z_{ij} = 1$ there exists exactly one $z_{ik} = -1$ and vice versa. As a consequence, for this node $i \in I$ the corresponding first-stage and second-stage degree equations (10) are fulfilled. Moreover, for the nodes $j, k \in J$ the first-stage and second-stage degree inequalities (11) are tight. Consequently, all variables x_{iu}, y_{iu}, $u \in J$, and x_{vj}, x_{vk} and y_{vj}, y_{vk} for $v \in I$ are uniquely fixed to either zero or one. Hence, these variables can be eliminated by projection. For the corresponding graph, the nodes and edges associated with these variables are deleted. More precisely, for each $z_{ij} = 1$ and $z_{ik} = -1$ the node $i \in I$ and the nodes $j, k \in J$ with all its incident edges are removed from each graph component G_0 and G_1. Additionally, all variables x_{ij} and y_{ij} for that $z_{ij} = x_{ij} = 0$ $((i,j) \in E_0 \setminus \mathcal{C})$ or $z_{ij} = -y_{ij} = 0$ $((i,j) \in E_1 \setminus \mathcal{C})$ holds are eliminated as well. The corresponding edges $(i,j) \notin \mathcal{C}$ are removed from their component G_0 or G_1.

Let $G' = (I' \cup J', E')$ be the graph that is obtained by the projection. The projection yields that G'_0 and G'_1 have the same node sets I'_s and J'_s and the same edge sets E'_s, $s \in \{0,1\}$. Thus, we set $E' = E'_0 = E'_1$, and $I' = I'_0 = I'_1$ and $J' = J'_0 = J'_1$.

Let $\mathcal{P}'_z$ be the polytope that results from $\mathcal{P}_z$ by the projection and $\mathcal{C}'$ from $\mathcal{C}$, i.e., $\mathcal{C}' = E'$ is the set of edges that are contained in both components G'_0 and G'_1. Then $\mathcal{P}'_z$ is defined by

$$\mathcal{P}'_z = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{|E'|} \colon \; A^{I'} \begin{pmatrix} x \\ y \end{pmatrix} = 1, \; A^{J'} \begin{pmatrix} x \\ y \end{pmatrix} \leq 1, \; x, y \geq 0, \right. \\ \left. x_{ij} - y_{ij} = 0 \qquad \forall\, (i,j) \in E' \right\}. \tag{27}$$

This yields the following lemma.

Lemma 3.4. *Let $\mathcal{P}_z$ be defined as in* (26) *and let $z \in \{-1,0,1\}^\sigma$ be a feasible z-configuration. Then $\mathcal{P}_z$ is an integral polyhedron.*

Proof. It suffices to show that $\mathcal{P}'_z$ is an integral polyhedron. Each edge $(i,j) \in E_s$ for $s \in \{0,1\}$ is also contained in $\mathcal{C}'$. In other words, the edge sets of both graph components are identical. As a result, $\mathcal{P}'_z$ as defined in (27) is equivalent to

$$\mathcal{P}'_z = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{|E'|} \colon \; A_x^{I'} x = 1, \; A_x^{J'} x \leq 1, \; x \geq 0, \right. \\ \left. x_{ij} = y_{ij} \qquad \forall\, (i,j) \in E' \right\}.$$

As the linear description of the bipartite matching polytope is given by the degree (in-)equalities and the non-negativity constraints [4], it is $\mathcal{P}'_z = \operatorname{conv}(\mathcal{P}'_z \cap \mathbb{Z}^{|E'|})$ and the vertices $(x^\top \quad y^\top)^\top$ of $\mathcal{P}'_z$ are integral vectors with entries $x_{ij} \in \{0,1\}$ and $y_{ij} \in \{0,1\}$ for $(i,j) \in E'$. Lifting $\mathcal{P}'_z$ back in the original variable space adds variables fixed to either zero or one. Therefore, $\mathcal{P}_z$ is an integral polyhedron and $\mathcal{P}_z = \operatorname{conv}(\mathcal{P}_z \cap \mathbb{Z}^{|E|})$ follows. □

Finally, the proof of Theorem 3.2 is similar to the proof of Theorem 2 in [2] and is given here for completeness.

Proof of Theorem 3.2. Let $\mathcal{F}$ be the set of all feasible z-configurations z with $\overline{d} - Uz \geq 0$. The integrality of the matrix W yields that for any $\begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top \in \mathcal{P}_{\text{RunOPT}}$ we have an integral product $W \begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top \in \mathbb{Z}^\sigma$. As a result,

$$\mathcal{P}_{\text{RunOPT}} = \operatorname{conv}\left(\mathcal{P} \cap \mathbb{Z}^{|E|}\right) \subseteq \operatorname{conv}\left(\left\{\begin{pmatrix} x \\ y \end{pmatrix} \in \mathcal{P} \colon W \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{Z}^\sigma \right\}\right) = \mathcal{P}_W$$

holds. We need to show that $\mathcal{P}_W$ is an integral polyhedron. From Lemma 3.4 follows that each $\mathcal{P}_z$ is an integral polyhedron, i.e., $\mathcal{P}_z = \operatorname{conv}\left(\mathcal{P}_z \cap \mathbb{Z}^{|E|}\right)$. Hence, for any $\begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top \in \mathcal{P}_{\text{RunOPT}}$ there exists a feasible z configuration with $\begin{pmatrix} x^\top & y^\top \end{pmatrix}^\top \in \mathcal{P}_z$. We obtain

$$\begin{aligned} \mathcal{P}_W &= \operatorname{conv}\left(\bigcup_{z \in \mathcal{F}} \mathcal{P}_z\right) = \operatorname{conv}\left(\bigcup_{z \in \mathcal{F}} \operatorname{conv}\left(\mathcal{P}_z \cap \mathbb{Z}^{|E|}\right)\right) \\ &= \operatorname{conv}\left(\bigcup_{z \in \mathcal{F}} \left(\mathcal{P}_z \cap \mathbb{Z}^{|E|}\right)\right) = \operatorname{conv}\left(\mathcal{P}_W \cap \mathbb{Z}^{|E|}\right). \end{aligned}$$

Therefore, $\mathcal{P}_W$ is an integral polyhedron and $\mathcal{P}_W = \mathcal{P}_{\text{RunOPT}}$. This provides a description of $\mathcal{P}_{\text{RunOPT}}$ with σ integrality constraints. □

4. Implementation and Computational Results

In the computational study, we evaluate the performance of solving the different reformulations for (RunOPT) from Section 3.

4.1. Problem Setting and Specific Approaches

We performed computational experiments both for one ($N = 1$) as well as for several disturbed second-stage scenarios ($N > 1$) with $\kappa \in \{1, |I|\}$ (for theoretical aspects we already referred to [20]). The results for several disturbed scenarios show characteristically similar behavior, when compared to $N = 1$, in terms of which approach delivers better performance. For ease of exposition, we therefore focus on examining the case with one second-stage scenario.

For the implementation we use the C++-API of Gurobi 7.0 with standard settings. For each instance we set a maximum time limit of 12 h (43,200 s). Each job was run on a single core with a virtual memory limit of 15 GB. The computational experiments were carried out on a queuing cluster of Intel Xeon E5-2690 3.00 GHz computers with 25 MB cache and 128 GB RAM.

For increasing instance sizes and for varying number of resource constraints ($\kappa \in \{1, \ldots, |I|\}$), the solution of the standard IP formulation of the problem was compared with solving the model where aggregation (20) was added. As shown in Section 3, in the cases $\kappa = 1$ and $\kappa = |I|$, $b = 1$ it is sufficient to demand integrality only for the z variables. However, running times were lower when x and y were declared binary as well, as then Gurobi can easier exploit the matching structure in the algorithm.

We also evaluated the idea in Bader et al. [2] of applying a Hermite normal transformation to the reformulation, so that the resulting problem is not defined in an increased variable space but has the same number of variables as the original model. However, for the tested instances, this approach does not improve the running times. Also here, the transformed problem does not clearly show an assignment structure any more.

As a consequence, we will limit ourselves to comparing the IP formulation with the aggregated approach, where all x and y variables are declared binary. We study the following settings in detail:

(a) **Rec** denotes solving the original IP formulation (RunOPT) with the resource constraints (19) by Gurobi. Only binary variables $x_{ij}, y_{ij} \in \{0,1\}$ for all $(i,j) \in E$ occur in the considered model.

(b) **RecZ** refers to solving the formulation in an extended variable space with Gurobi. More precisely, we solve the formulation (RunOPT) with binary x and y variables and added aggregation constraints (20), with integer z variables. Furthermore, in the resource constraints (19) we replace x and y with z:

$$\pm\Big(\sum_{j\in J_{E(p)}} j\cdot z_{p,j}\Big) \le d, \qquad\qquad \forall p = 1,\dots,\kappa.$$

We evaluated the running times for solving instances with different numbers of resource constraints $\kappa \in \{1,\dots,|I|\}$. In the pre-tactical planning phase, time window sizes of 10 min [14] are practically relevant. For the right-hand side of the resource constraints it is reasonable to allow reschedulings of up to five time windows, i.e., $d := 5$. This means that the difference between first and second stage assignments is always less than an hour. We performed computational experiments with different values than reported here. Apart from the fact that instances with increasing values for d get easier in practice, the performance results showed characteristically similar behavior and are skipped here.

4.2. The Benchmark Sets of Instances

As a benchmark instance set, we use instances for the ATM application from Section 3. For each instance, we generate the associate bipartite graphs $G = (I \cup J, E)$ with two connected components $G_0 = (I \cup J, E_0)$ and $G_1 = (I \cup J, E_1)$ that refer to the nominal and the disturbed second-stage scenario. In order to evaluate the performance of solving the reformulations for varying instance sizes, we create random instances as follows. For each aircraft $i \in I$ we consider as time windows $j \in J$:

- The *scheduled time window* is a fixed window $j_{\mathrm{ST}} \in J$ that containes the scheduled time from the flight plan or ticket.
- The *earliest time window* $j_{\mathrm{ET}} \in J$ and the *latest time window* $j_{\mathrm{LT}} \in J$ refer to the earliest and latest possible landing time, respectively, and depend on operational conditions.

The earliest and the latest times are uncertain. For each aircraft $i \in I$, we randomly choose the two first-stage time windows j^0_{ET} and j^0_{LT} uniformly distributed from J, with $j^0_{\mathrm{ET}} < j^0_{\mathrm{LT}}$. These determine the set of feasible first-stage assignments $W^0_i := \{j^0_{\mathrm{ET}}, j^0_{\mathrm{ET}}+1,\dots,j^0_{\mathrm{LT}}\}$.

Note that this implies property (8) for the graph components G_0 and G_1. A scheduled time window j_{ST} is then chosen randomly from W_i^0, also uniformly distributed. Afterwards, the feasible second-stage assignments $W_i^1 := \{j_{\mathrm{ET}}^1, \ldots, j_{\mathrm{LT}}^1\}$ are drawn randomly under the conditions $|W_i^1| = |W_i^0|$ and $W_i^1 \cap W_i^0 \neq \emptyset$. Precisely, the earliest time window j_{ET}^1 is drawn with uniform distribution from $\left\{j_{\mathrm{ET}}^0 - \left(j_{\mathrm{LT}}^0 - j_{\mathrm{ET}}^0\right), \ldots, j_{\mathrm{LT}}^0\right\}$, then the latest time window is set to $j_{\mathrm{LT}}^1 = j_{\mathrm{ET}}^1 + \left(j_{\mathrm{LT}}^0 - j_{\mathrm{ET}}^0\right)$.

The weights c_{ij}^0 and c_{ij}^1 in the objective function represent the distance of the time windows $j \in J$ to the scheduled time j_{ST} for each aircraft $i \in I$. Thus, they model delay or earliness for all aircraft. Due to fairness aspects, we penalize delay quadratically and earliness linearly [14]. Such objective functions are often used in this context. This leads to

$$c_{ij}^0 = c_{ij}^1 = \begin{cases} (j - j_{\mathrm{ST}})^2, & \text{if } j \geq j_{\mathrm{ST}}, \\ (j_{\mathrm{ST}} - j), & \text{if } j < j_{\mathrm{ST}}. \end{cases}$$

We choose the right-hand values b large enough to enable the generation of feasible assignments. We thus set

$$b_{\mathrm{average}} = \left\lfloor \frac{|I|}{|J|} \right\rfloor = \left\lfloor \frac{\text{number of aircraft}}{\text{number of time windows}} \right\rfloor$$

and randomly choose $b_j^0,\ b_j^1 \in [b_{\mathrm{average}} - 1, b_{\mathrm{average}} + 3]$.

We evaluated two instance sets, which together consist of 45 different instance sizes, i.e., different $|I|$ and $|J|$, for each of which we generated 10 different instances:

- `InstanceSet1` has 25 instance sizes, namely for each $|J| \in \{50, 100, 200, 300, 400\}$ we consider $|I|$ such that $|I|/|J| \in \{5, 10, 20, 50, 100\}$.
- `InstanceSet2` is generated by modeling the ATM situation more realistically. Assuming time windows of 10 min (see [14]) and minimum distance requirements between aircraft of approximately 100 s, each window can be assigned to about six aircraft. Examining a planning horizon of one day, discretized into 10-minute windows, then corresponds approximately to $|J| = 140$ and $|I| = 6 \cdot |J|$. We slightly decrease the factor $|I|/|J|$ in order to make the instances practically more difficult. To this end, a value of 0.2 turns out to be appropriate. Varying the time window sizes, we consider 20 instance sizes in `InstanceSet2` with $|J| \in \{140, 280\}$ and $|I|$ such that $|I|/|J| \in \{5.8, 6.8, 7.8, 8.8, 9.8, 19.8, 29.8, 39.8, 49.8, 99.8\}$. For example, $|J| = 140$ and $|I| = 99.8 \cdot |J|$ corresponds to a planning horizon of about two weeks, discretized in time windows of 2.5 hours. Choosing $|J| = 280$ then corresponds to a doubled time span of one month so that in summary we explore planning horizons between one day and one month.

In the following, we show experimental results for values $\kappa \in \left\{1, 2, 4, 8, 16, \lfloor \frac{n}{16} \rfloor, \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor, n\right\}$, where $n := |I|$. The partition of the set of aircraft $I = I_1 \dot{\cup} \ldots \dot{\cup} I_\kappa$ is implemented so that they have similar cardinality. For our tested κ values we set $|I_p| = \lfloor \frac{n}{\kappa} \rfloor$ for all $p < \kappa$, and $|I_\kappa| = n - (\kappa - 1)\lfloor \frac{n}{\kappa} \rfloor$.

4.3. Experimental Results and Comparison of the Approaches

We use performance profiles as proposed by Dolan and Moré [21] to facilitate the comparison of the two approaches Rec and RecZ. Assuming that approach a solves problem instance i in $\text{cpu}_{i,a}$ seconds, a *performance ratio* is calculated as $\frac{\text{cpu}_{i,a}}{\min_{a'} \text{cpu}_{i,a'}}$. For each approach, a performance profile shows on the y-axis the percentage of instances that could be solved with a performance ratio up to the corresponding value on the x-axis. All profiles are generated using Perprof-py [22]. The corresponding tables can be found in the Appendix.

4.3.1. Experimental Results for `InstanceSet1`

We first study the performance profiles for single κ values regarding the first instance set. In Figure 1, we see profiles for the two extreme cases $\kappa = 1$ and $\kappa = n$, respectively, whereas Figure 2 shows the remaining profiles.

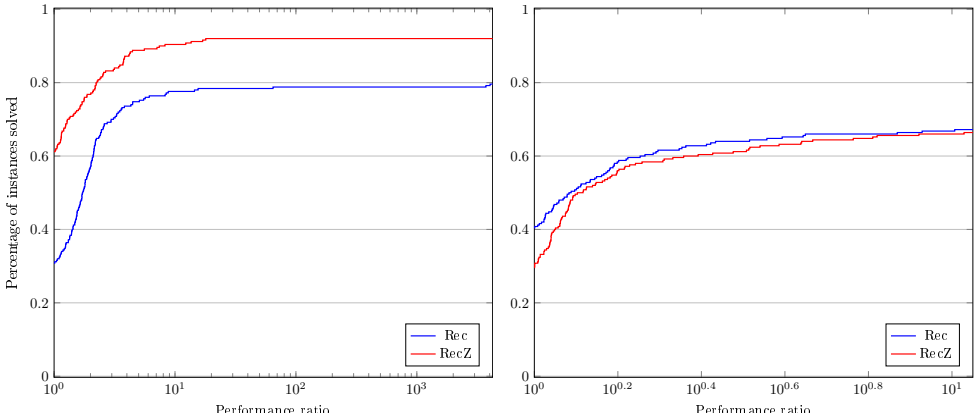

Figure 1: Performance profiles for `InstanceSet1`, $\kappa = 1$ (left) and $\kappa = n$ (right). They show the percentage of instances that could be solved with a performance ratio up to the corresponding value on the x-axis.

Figure 1 clearly shows that RecZ solves considerably more instances and provides faster runtimes than Rec for $\kappa = 1$. Indeed, for 61 % of the instances, the reformulation RecZ was the fastest method, whereas only for 31 % of the instances the standard IP was fastest. This trend continues for multiples of the fastest running time, as can be seen for increasing values on the x-axis. Eventually, RecZ solves 92 % of the instances within the given time limit, while Rec solves 80 %. Thus, in the setting with only one resource constraint, the aggregation approach RecZ has a clear advantage over solving the standard IP formulation. This can be understood because in this setting we have relatively few aggregation constraints (17), which in turn aggregate many variables.

In contrast, the case $\kappa = n$ describes the classic RunOPT where for each aircraft an individual restriction of reschedulings is imposed. Here we have more aggregation constraints, each aggregating fewer variables. Solving the standard IP now performs somewhat better than solving the reformulation, although the results are very tight. A clear winning strategy is not apparent at first sight. There are instances for which the reformulation approach RecZ gives better results, for other instances Rec is somewhat better. However, the number of instances solved per instance size differs by at most one.

For 21 of 25 instance sizes, this number is equal, three times Rec solves one instance more and once RecZ solves one more (for details, see Table A.10 in the Appendix).

Next, we compare the computational results for Rec and RecZ for other κ values between 1 and n. The remaining performance profiles for $\kappa \in \{2, 4, 8, 16, \lfloor \frac{n}{16} \rfloor, \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor\}$ are shown in Figure 2. For small values of κ, RecZ shows better performance than Rec. For both $\kappa = 2$ and $\kappa = 4$, RecZ is fastest for 45 % of the instances, whereas Rec is fastest for 42 %. And RecZ solves more instances within the given time limit (86 % and 84 %, respectively) than Rec (79 % and 78 %, respectively). For $\kappa = 8$, Rec solves more instances faster than RecZ (46 % vs. 39 %), however still more instances could be solved by RecZ (80 % vs. 75 %). These results show that RecZ is beneficial when compared to solving Rec in the sense that more instances could be solved for $\kappa \in \{2, 4, 8\}$, but Rec often solves instances faster.
For larger values of $\kappa \in \{16, \lfloor \frac{n}{16} \rfloor, \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor\}$, the picture changes in the sense that then solving Rec yields overall better performance than solving the reformulation RecZ. This can be understood because a larger κ increases the number σ of additional variables in RecZ (see (22)).

In summary, these results show that for not too large values of $\kappa \leq 8$, the reformulated model RecZ should be solved, whereas for values of κ larger than this, the standard IP should be used.

4.3.2. Computational Results for `InstanceSet2`

It turns out that the instances in `InstanceSet2` are computationally more demanding in practice, as can be seen in the tables in the Appendix. Indeed, for example 53.5 % of the instances of size $|J| = 300$, $|I|/|J| = 10$ from `InstanceSet1` can be solved within the time limit (considering all κ values), whereas for instances of similar size $|J| = 280$, $|I|/|J| = 9.8$ from `InstanceSet2` only 25.5 % are solved. It is thus of particular interest to compare the two models for these difficult instances. The performance profiles for `InstanceSet2` are shown in Figure 3.

Solving RecZ is clearly superior to solving Rec for $\kappa \in \{1, 2, 4, 8\}$. Regarding $\kappa = 1$, for 73 % of the instances, the reformulation RecZ was the fastest method, whereas only for 6 % of the instances the standard IP was fastest. Moreover, RecZ solves 79 % of the instances within the time limit and Rec solves only 34 %. For $\kappa = 8$, still solving RecZ is superior as it solves 26 % fastest and finally solves 47 %, contrary to Rec solving 23 % fastest and finally solves 27 %.
For $\kappa = 16$ solving RecZ still is beneficial in terms of the number of solved instances (37 % vs. 30 %), whereas the instances expressed in model Rec are often solved faster (23 % vs. 18 %). For an increasing number of κ, a similar picture as in `InstanceSet1` is visible, namely that more instances are solved by Rec than by RecZ and faster. As an exception, we note that for $\kappa = n$, eventually more instances could be solved in model RecZ than in the standard model Rec (63 % vs. 61 %).

To illustrate the reasons for the better performance of RecZ compared to Rec for a small up to a relatively large number of resource constraints, we now look at another parameter for grouping the results.

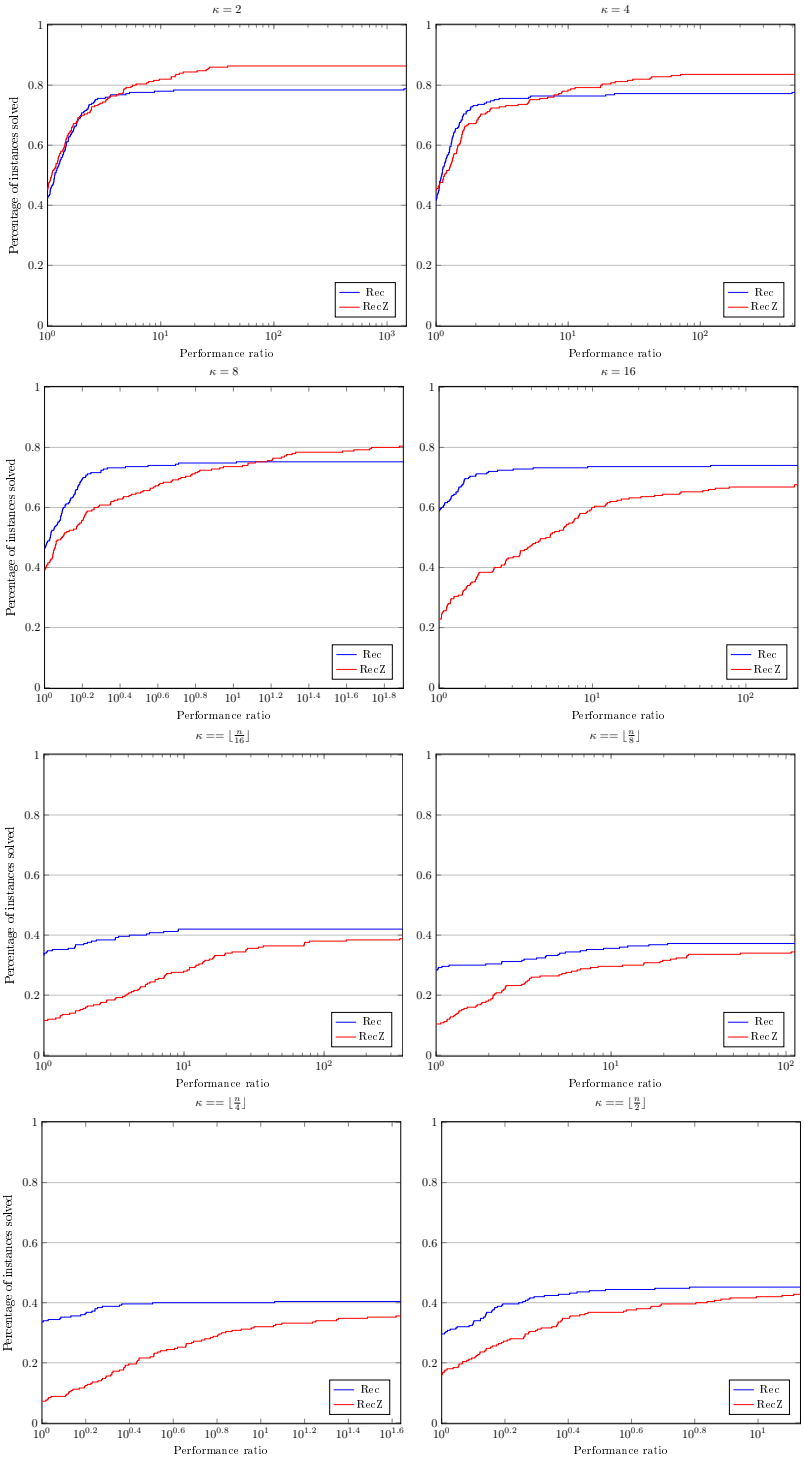

Figure 2: Performance profiles for `InstanceSet1`, $\kappa = 2, 4, 8, 16, \lfloor \frac{n}{16} \rfloor, \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor$. They show the percentage of instances that could be solved with a performance ratio up to the corresponding value on the x-axis.

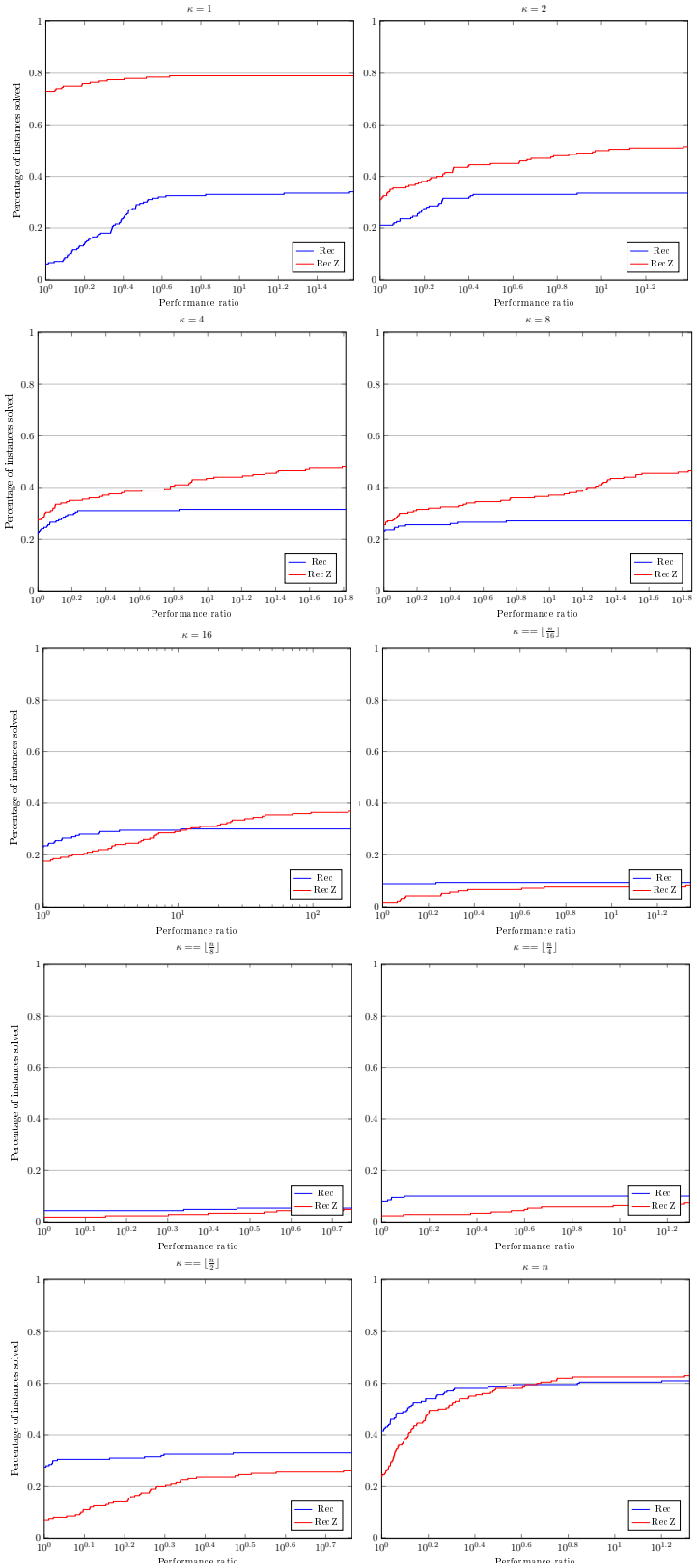

Figure 3: Performance profiles for `InstanceSet2`, $\kappa = 2, 4, 8, 16, \lfloor \frac{n}{16} \rfloor, \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor$. They show the percentage of instances that could be solved with a performance ratio up to the corresponding value on the x-axis.

The smaller the number of resource constraints, the lower the number of additional variables that need to be added in order to define the reformulations. We therefore take the relative increase in variables as a parameter. We note that in absolute terms, RecZ has σ more variables than Rec, with σ defined as in (22). As shown in the previous section, under certain conditions, these are the only variables in RecZ that must be claimed as integers. In our case, however, the instances could be solved more efficiently if we required all variables as integers. The relative increase in variables can now be described by $\frac{\sigma}{|\text{Vars(Rec)}|} \cdot 100\,\%$. The results for all 200 instances of `InstanceSet2` with all 10 κ values (i.e., 2,000 computation "tasks") yield a median of 6.4 %. Figure 4 shows both the result for an increase of less as well as that for more than 6.4 %, respectively. Thus, we have two groups of 1,000 tasks each. Further, Figure 5 shows four groups of 500 tasks each. From these figures it becomes clear that the aggregation approach is

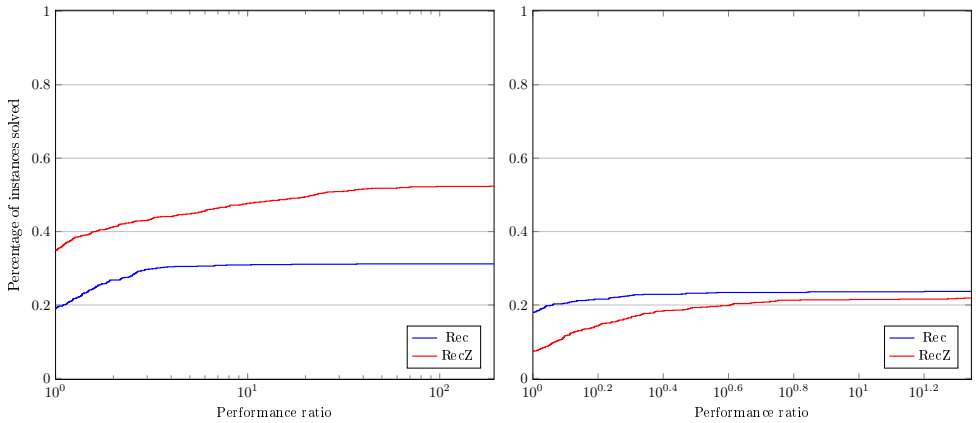

Figure 4: Performance profiles for`InstanceSet2`, relative increase of the number of variables $< 6.35\,\%$ (left) and $\geq 6.35\,\%$ (right). 6.35 % corresponds to the median value, so that we get two groups with 1,000 computation tasks each. The profiles show the percentage of instances that could be solved with a performance ratio up to the corresponding value on the x-axis.

advantageous if σ is relatively small, i.e., when many variables are aggregated. In fact, the reformulation approach RecZ yields better results even if we increase the grouping value from 6.4 %. Hence, an aggregation remains also helpful for somewhat larger σ's.

In summary, the reformulation approach works well for sufficiently difficult instances where relatively large numbers of variables are aggregated. The latter is usually the case for a low number of resource constraints.

Conclusions

In this work, we have given a concretization of the MIP reformulation approach of Bader et. al [2] for two-stage assignment problems with resource constraints as they appear in a challenging ATM application.

The presented computational results show that the new reformulation approach outperforms the original IP formulation of the considered ATM problem for the setting with small to medium number of resource constraints.

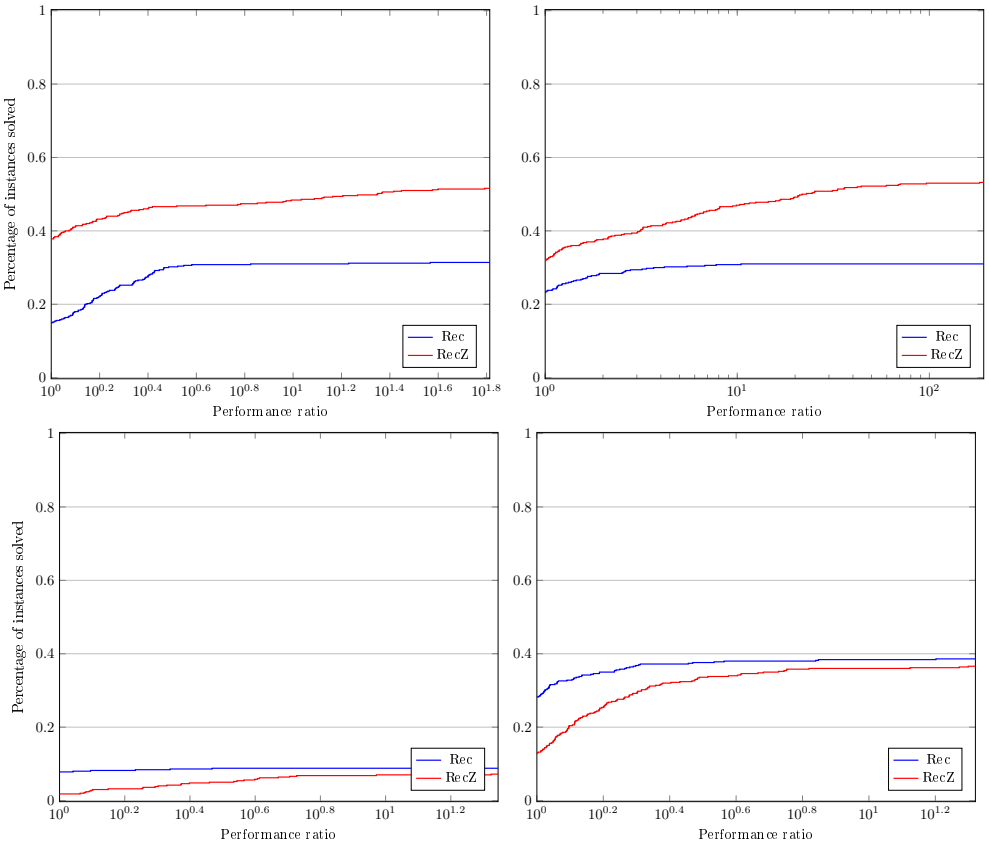

Figure 5: Performance profiles for `InstanceSet2`, relative increase of the number of variables grouped into four groups with 500 computation tasks each: $[0\,\%, 0.1863\,\%)$ (upper left), $[0.1863\,\%, 6.35\,\%)$ (upper right), $[6.35\,\%, 35.27\,\%)$ (lower left), $[35.27\,\%, \infty)$ (lower right). The profiles show the percentage of instances that could be solved with a performance ratio up to the corresponding value on the x-axis.

In future research, it is of interest to study the approach on applications that are not based on an assignment problem.

Acknowledgements

We gratefully acknowledge the computing resources provided by the group of Michael Jünger in Cologne. We are also thankful to Christian Biefel for proofreading and commenting on an earlier version of this paper. Further, we acknowledge the financial support by the Federal Ministry for Economic Affairs and Energy of Germany in the project HOTRUN (project number 20E1720B).

References

[1] J. Paat, M. Schlöter, R. Weismantel, The integrality number of an integer program, in: D. Bienstock, G. Zambelli (Eds.), Integer Programming and Combinatorial Optimization, Springer International Publishing, Cham, 2020, pp. 338–350.

[2] J. Bader, R. Hildebrand, R. Weismantel, R. Zenklusen, Mixed Integer Reformulations of Integer Programs and the Affine TU-Dimension of a Matrix, Mathematical Programming 169 (2018) 565–584.

[3] A. J. Hoffman, J. B. Kruskal, Integral Boundary Points of Convex Polyhedra, in: H. W. Kuhn, A. W. Tucker (Eds.), Linear Inequalities and Related Systems, Princeton, 1956, pp. 223–246.

[4] A. Schrijver, Combinatorial Optimization : Polyhedra and Efficiency, Algorithms and Combinatorics, Springer-Verlag, Berlin, Heidelberg, New York, N.Y., et al., 2003.

[5] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, Inc., New York, NY, USA, 1990.

[6] K. Truemper, Unimodular Matrices of Flow Problems with Additional Constraints, Networks 7 (4) (1977) 343–358.

[7] M. Dell'Amico, S. Martello, The k-Cardinality Assignment Problem, Discrete Applied Mathematics 76 (1) (1997) 103 – 121.

[8] M. Dell'Amico, A. Lodi, S. Martello, Efficient Algorithms and Codes for k-Cardinality Assignment Problems, Discrete Applied Mathematics 110 (1) (2001) 25 – 40, proceedings of the First Conference on Algorithms and Experiments.

[9] A. Volgenant, Solving the k-Cardinality Assignment Problem by Transformation, European Journal of Operational Research 157 (2) (2004) 322 – 331.

[10] A. Y. Alfakih, K. G. Murty, Adjacency on the Constrained Assignment Problem, Discrete Applied Mathematics 87 (1) (1998) 269 – 274.

[11] A. Ghouila-Houri, Caractérisation des Matrices Totalement Unimodulaires, C. R. Acad. Sci. Paris 254 (1962) 1192–1194.

[12] J. Beasley, M. Krishnamoorthy, Y. Sharaiha, D. Abramson, Scheduling aircraft landings - the static case, Transportation Science 34 (2) (2000) 180–197.

[13] M. Soomer, G. Franx, Scheduling aircraft landings using airlines' preferences, European Journal of Operational Research 190 (1) (2008) 277–291.

[14] M. Kapolke, N. Fürstenau, A. Heidt, F. Liers, M. Mittendorf, C. Weiß, Pre-tactical Optimization of Runway Utilization Under Uncertainty, Journal of Air Transport Management 56, Part A (2016) 48 – 56, long-term and Innovative Research in ATM.

[15] A. Heidt, H. Helmke, M. Kapolke, F. Liers, A. Martin, Robust runway scheduling under uncertain conditions, Journal of Air Transport Management 56 (2016) 28–37. doi:10.1016/j.jairtraman.2016.02.009.

[16] N. Kong, A. J. Schaefer, A Factor $\frac{1}{2}$ Approximation Algorithm for Two-Stage Stochastic Matching Problems, European Journal of Operational Research 172 (3) (2006) 740–746.

[17] B. Toktas, Addressing Capacity Uncertainty in Resource-Constrained Assignment Problems, Ph.D. thesis, University of Washington (2003).

[18] I. Katriel, C. Kenyon-Mathieu, E. Upfal, Commitment Under Uncertainty: Two-Stage Stochastic Matching Problems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 171–182.

[19] D. Adjiashvili, V. Bindewald, D. Michaels, Robust Assignments via Ear Decompositions and Randomized Rounding, in: I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, D. Sangiorgi (Eds.), 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), Vol. 55

of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2016, pp. 71:1–71:14.
[20] L. Hupp, Integer and mixed-integer reformulations of stochastic, resource-constrained, and quadratic matching problems, Ph.D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (2017).
[21] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical Programming 91 (2) (2002) 201–213. doi:10.1007/s101070100263.
[22] A. Siqueira, R. Gaia Costa da Silva, L.-R. Santos, Perprof-py: A python package for performance profile of mathematical optimization software, Journal of Open Research Software 4. doi:10.5334/jors.81.

Appendix A. Tables for `InstanceSet1`

In Table A.1–A.10, we see detailed results for the instances of `InstanceSet1`. In the first two columns of each table the instances are specified by the number of nodes $|I|$ and $|J|$. The presented results are averaged over 10 instances per instance size. For each method we report the running time in seconds and the number of nodes explored in the branch-and-bound tree of Gurobi. Furthermore, we indicate the number of instances that could be solved within the time limit (12 hours) of the instances that are feasible at the beginning.

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	0.3	0	10/10	**0.2**	0	10/10
500	50	1.6	52	10/10	**0.6**	17	10/10
1,000	50	2,852.7	260,388	7/10	4.6	298	**10/10**
2,500	50	13.1	0	2/8	12.4	150	**8/8**
5,000	50	18.4	0	1/4	25.8	0	**4/4**
500	100	1.1	0	10/10	**0.6**	0	10/10
1,000	100	4,296.6	368,776	8/10	5.4	140	**10/10**
2,000	100	99.8	1,926	5/10	19.8	192	**10/10**
5,000	100	141.2	235	4/10	91.1	401	**10/10**
10,000	100	260.7	259	7/8	149.9	142	**8/8**
1,000	200	6.3	0	10/10	**3.3**	0	10/10
2,000	200	22.3	0	10/10	**11.7**	0	10/10
4,000	200	797.9	2,876	10/10	**160.2**	330	10/10
10,000	200	551.6	265	10/10	**263.8**	94	10/10
20,000	200	1,276.9	208	7/10	1,800.6	441	**10/10**
1,500	300	19.2	0	10/10	**12.9**	0	10/10
3,000	300	97.4	57	10/10	**83.7**	62	10/10
6,000	300	634.7	392	10/10	**482.4**	292	10/10
15,000	300	2,877.1	633	10/10	**1,605.1**	170	10/10
30,000	300	**1,275.5**	0	10/10	5,223.9	285	10/10
2,000	400	**36.4**	0	10/10	43.0	0	10/10
4,000	400	**115.9**	0	10/10	170.8	44	10/10
8,000	400	**986.9**	199	10/10	1,645.5	573	10/10
20,000	400	4,205.6	398	8/10	9,225.7	984	**10/10**
40,000	400	–	–	0/10	–	–	0/10

Table A.1: Results for Rec and RecZ with $\kappa = 1$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Appendix B. Tables for `InstanceSet2`

In Table B.11–B.20, we see detailed results for `InstanceSet2`. The tables are structured similarly to those in Appendix A.

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	0.3	0	10/10	**0.2**	0	10/10
500	50	**1.5**	51	10/10	1.6	51	10/10
1,000	50	807.2	101,534	6/10	8.6	207	**10/10**
2,500	50	21.6	15	4/8	59.3	353	**7/8**
5,000	50	19.3	0	1/4	229.0	476	**2/4**
500	100	1.2	0	10/10	**0.9**	0	10/10
1,000	100	12.8	462	8/10	17.1	160	**10/10**
2,000	100	21.0	43	5/10	176.5	1,297	**10/10**
5,000	100	142.9	18	4/10	588.2	1,934	**7/10**
10,000	100	4,908.0	2,075	7/8	**2,625.0**	1,373	7/8
1,000	200	5.5	0	10/10	**4.3**	0	10/10
2,000	200	27.9	0	10/10	**16.4**	0	10/10
4,000	200	**325.3**	341	9/10	1,033.3	590	9/10
10,000	200	**484.0**	0	10/10	2,004.7	719	10/10
20,000	200	2,311.7	0	7/10	6,308.7	478	**8/10**
1,500	300	18.8	0	10/10	**15.9**	0	10/10
3,000	300	203.5	65	10/10	**182.4**	105	10/10
6,000	300	2,885.3	411	10/10	**2,306.9**	878	10/10
15,000	300	835.9	0	9/10	6,089.2	831	**10/10**
30,000	300	2,766.2	0	**10/10**	3,808.3	0	9/10
2,000	400	41.6	0	10/10	**38.7**	0	10/10
4,000	400	**168.1**	0	10/10	323.3	90	10/10
8,000	400	1,526.0	149	**10/10**	1,454.8	297	9/10
20,000	400	2,186.8	0	7/10	21,226.2	474	**8/10**
40,000	400	–	–	0/10	–	–	0/10

Table A.2: Results for Rec and RecZ with $\kappa = 2$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.2**	0	10/10	0.3	0	10/10
500	50	**1.1**	0	10/10	1.3	0	10/10
1,000	50	485.8	58,522	5/10	18.6	208	**10/10**
2,500	50	76.9	622	3/8	423.9	2,388	**7/8**
5,000	50	15.9	0	2/5	636.1	848	**4/5**
500	100	1.3	0	10/10	**1.1**	0	10/10
1,000	100	5.1	3	8/10	22.3	156	**10/10**
2,000	100	29.4	22	5/10	581.0	1,677	**10/10**
5,000	100	415.5	479	6/10	2,090.1	1,769	**8/10**
10,000	100	**1,954.6**	885	6/8	2,911.6	1,034	6/8
1,000	200	7.3	0	10/10	**6.7**	0	10/10
2,000	200	43.2	0	10/10	**26.0**	0	10/10
4,000	200	**847.1**	443	9/10	1,758.8	560	9/10
10,000	200	**2,184.9**	609	9/10	4,962.3	749	9/10
20,000	200	**4,022.8**	597	7/10	5,722.2	141	7/10
1,500	300	24.4	0	10/10	**22.1**	0	10/10
3,000	300	**316.6**	63	10/10	715.9	131	10/10
6,000	300	4,206.8	441	**10/10**	6,434.0	786	9/10
15,000	300	4,770.6	168	**8/10**	3,442.5	41	5/10
30,000	300	4,323.4	0	**10/10**	5,063.2	0	9/10
2,000	400	55.5	0	10/10	**47.1**	0	10/10
4,000	400	**208.0**	0	10/10	912.3	156	10/10
8,000	400	**3,467.8**	66	10/10	7,192.3	292	10/10
20,000	400	**4,208.3**	0	6/10	11,226.7	210	6/10
40,000	400	–	–	0/10	–	–	0/10

Table A.3: Results for Rec and RecZ with $\kappa = 4$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.3**	0	10/10	0.6	0	10/10
500	50	3.1	52	10/10	**2.0**	0	10/10
1,000	50	39.1	2,082	2/10	92.9	2,240	**10/10**
2,500	50	7.4	0	2/8	581.1	2,035	**6/8**
5,000	50	16.7	0	2/5	124.4	208	**3/5**
500	100	**1.3**	0	10/10	1.9	0	10/10
1,000	100	456.0	17,334	9/10	98.6	461	**10/10**
2,000	100	88.7	83	5/10	1,520.1	3,574	**10/10**
5,000	100	1,311.0	606	7/10	1,429.2	1,102	**9/10**
10,000	100	9,817.5	1,737	**4/8**	1,148.1	263	3/8
1,000	200	8.9	0	10/10	**8.8**	0	10/10
2,000	200	63.2	0	10/10	**51.9**	0	10/10
4,000	200	2,844.8	1,004	9/10	4,318.7	1,568	**10/10**
10,000	200	**4,089.7**	260	8/10	12,889.9	1,277	8/10
20,000	200	**1,662.3**	0	7/10	9,640.5	376	7/10
1,500	300	29.1	0	10/10	**25.7**	0	10/10
3,000	300	**164.6**	0	10/10	748.2	43	10/10
6,000	300	8,021.9	391	**10/10**	7,666.9	546	7/10
15,000	300	2,716.0	3	**8/10**	4,059.9	64	6/10
30,000	300	3,868.7	0	**10/10**	4,004.8	0	8/10
2,000	400	73.1	0	10/10	**55.5**	0	10/10
4,000	400	303.9	0	**10/10**	2,507.4	105	9/10
8,000	400	1,353.7	0	8/10	11,775.6	737	**10/10**
20,000	400	8,970.9	0	**7/10**	16,291.1	8	5/10
40,000	400	–	–	0/10	–	–	0/10

Table A.4: Results for Rec and RecZ with $\kappa = 8$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.5**	0	10/10	6.3	68	10/10
500	50	**2.7**	0	10/10	7.7	2	10/10
1,000	50	2,039.5	229,533	3/10	152.1	1,774	**10/10**
2,500	50	149.9	227	4/8	542.7	1,542	**7/8**
5,000	50	22.6	0	2/5	**21.3**	0	2/5
500	100	**2.5**	0	10/10	71.8	314	10/10
1,000	100	3,183.7	88,578	7/10	683.0	2,703	**10/10**
2,000	100	944.1	2,439	7/10	2,425.8	2,432	**10/10**
5,000	100	640.5	55	6/10	7,358.0	1,929	**7/10**
10,000	100	1,688.6	94	6/8	2,963.7	236	**7/8**
1,000	200	**15.6**	0	10/10	56.6	0	10/10
2,000	200	**154.5**	0	10/10	2,945.7	801	10/10
4,000	200	6,497.7	522	**9/10**	4,588.3	208	5/10
10,000	200	10,529.2	151	**5/10**	21,403.5	580	2/10
20,000	200	2,642.3	0	**10/10**	8,450.9	69	9/10
1,500	300	**61.9**	0	10/10	129.5	0	10/10
3,000	300	888.1	18	**10/10**	5,667.6	300	7/10
6,000	300	8,106.1	63	**5/10**	775.2	0	2/10
15,000	300	2,850.5	0	**7/10**	7,583.1	0	3/10
30,000	300	9,397.2	0	**9/10**	19,186.7	0	6/10
2,000	400	146.7	0	10/10	**120.9**	0	10/10
4,000	400	732.6	0	**10/10**	1,512.1	0	7/10
8,000	400	2,241.1	0	**8/10**	15,340.6	15	3/10
20,000	400	6,459.2	0	**7/10**	22,767.2	0	2/10
40,000	400	–	–	0/10	–	–	0/10

Table A.5: Results for Rec and RecZ with $\kappa = 16$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.4**	0	10/10	5.3	91	10/10
500	50	**18.7**	248	10/10	119.4	2,077	10/10
1,000	50	222.9	2,032	9/10	2,531.6	4,598	**10/10**
2,500	50	**517.0**	723	10/10	779.4	452	10/10
5,000	50	3,456.9	3,738	10/10	**837.5**	260	10/10
500	100	**6.8**	3	10/10	139.6	181	10/10
1,000	100	**3,720.9**	40,196	9/10	6,431.7	17,786	9/10
2,000	100	9,101.6	6,533	**6/10**	11,333.3	3,016	5/10
5,000	100	22,154.5	8,986	2/10	23,706.0	6,701	**3/10**
10,000	100	–	–	0/10	17,645.7	282	**1/10**
1,000	200	**886.4**	258	10/10	6,447.4	1,598	10/10
2,000	200	28,557.6	4,467	**6/10**	21,086.4	9,300	4/10
4,000	200	–	–	0/10	–	–	0/10
10,000	200	–	–	0/10	–	–	0/10
20,000	200	–	–	0/10	–	–	0/10
1,500	300	18,129.2	2,967	**10/10**	24,707.0	5,505	5/10
3,000	300	–	–	0/10	–	–	0/10
6,000	300	–	–	0/10	–	–	0/10
15,000	300	–	–	0/10	–	–	0/10
30,000	300	–	–	0/10	–	–	0/10
2,000	400	25,261.3	2,466	**3/10**	–	–	0/10
4,000	400	–	–	0/10	–	–	0/10
8,000	400	–	–	0/10	–	–	0/10
20,000	400	–	–	0/10	–	–	0/10
40,000	400	–	–	0/10	–	–	0/10

Table A.6: Results for Rec and RecZ with $\kappa = \frac{n}{16}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**3.5**	71	10/10	12.4	276	10/10
500	50	**31.1**	612	10/10	64.5	568	10/10
1,000	50	**90.0**	635	10/10	173.9	607	10/10
2,500	50	1,081.3	3,948	10/10	**346.7**	418	10/10
5,000	50	8,275.4	16,551	10/10	**3,212.5**	2,490	10/10
500	100	**176.2**	1,954	10/10	2,002.6	25,976	10/10
1,000	100	1,061.4	1,870	7/10	8,896.9	8,070	**9/10**
2,000	100	**8,602.8**	8,631	7/10	16,785.2	10,383	7/10
5,000	100	24,982.1	31,190	**7/10**	36,012.5	17,344	1/10
10,000	100	–	–	0/10	–	–	0/10
1,000	200	10,353.4	7,976	**9/10**	14,320.6	4,617	8/10
2,000	200	33,825.4	7,147	**3/10**	–	–	0/10
4,000	200	–	–	0/10	–	–	0/10
10,000	200	–	–	0/10	–	–	0/10
20,000	200	–	–	0/10	–	–	0/10
1,500	300	–	–	0/10	38,835.9	12,542	**1/10**
3,000	300	–	–	0/10	–	–	0/10
6,000	300	–	–	0/10	–	–	0/10
15,000	300	–	–	0/10	–	–	0/10
30,000	300	–	–	0/10	–	–	0/10
2,000	400	–	–	0/10	–	–	0/10
4,000	400	–	–	0/10	–	–	0/10
8,000	400	–	–	0/10	–	–	0/10
20,000	400	–	–	0/10	–	–	0/10
40,000	400	–	–	0/10	–	–	0/10

Table A.7: Results for Rec and RecZ with $\kappa = \frac{n}{8}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	22.2	1,102	10/10	**18.4**	1,045	10/10
500	50	**11.2**	189	10/10	26.1	271	10/10
1,000	50	**83.2**	540	10/10	115.2	848	10/10
2,500	50	**1,141.7**	6,106	10/10	1,172.3	2,790	10/10
5,000	50	**3,996.5**	18,011	10/10	9,430.3	8,461	10/10
500	100	**499.6**	2,717	10/10	1,493.0	6,666	10/10
1,000	100	**655.5**	1,877	10/10	6,422.9	8,774	10/10
2,000	100	**6,844.5**	12,611	8/10	9,561.9	11,361	8/10
5,000	100	30,592.2	25,144	**9/10**	13,019.9	654	1/10
10,000	100	–	–	0/10	–	–	0/10
1,000	200	6,099.8	3,882	**10/10**	25,024.2	13,625	9/10
2,000	200	22,781.9	17,342	**3/10**	–	–	0/10
4,000	200	–	–	0/10	–	–	0/10
10,000	200	–	–	0/10	–	–	0/10
20,000	200	–	–	0/10	–	–	0/10
1,500	300	**38,627.5**	22,415	1/10	40,585.5	15,166	1/10
3,000	300	–	–	0/10	–	–	0/10
6,000	300	–	–	0/10	–	–	0/10
15,000	300	–	–	0/10	–	–	0/10
30,000	300	–	–	0/10	–	–	0/10
2,000	400	–	–	0/10	–	–	0/10
4,000	400	–	–	0/10	–	–	0/10
8,000	400	–	–	0/10	–	–	0/10
20,000	400	–	–	0/10	–	–	0/10
40,000	400	–	–	0/10	–	–	0/10

Table A.8: Results for Rec and RecZ with $\kappa = \frac{n}{4}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**3.4**	74	10/10	3.5	36	10/10
500	50	**3.0**	0	10/10	3.5	0	10/10
1,000	50	**12.9**	3	10/10	23.1	73	10/10
2,500	50	98.9	161	10/10	**90.8**	52	10/10
5,000	50	696.2	815	10/10	**514.1**	236	10/10
500	100	252.7	1,051	10/10	**192.8**	961	10/10
1,000	100	**142.7**	837	10/10	443.1	2,727	10/10
2,000	100	**4,252.0**	6,170	10/10	4,368.4	7,743	10/10
5,000	100	**11,532.1**	13,129	10/10	13,741.3	19,620	10/10
10,000	100	31,073.0	22,149	**6/10**	25,438.6	11,262	1/10
1,000	200	**4,218.5**	6,545	10/10	8,721.2	11,757	10/10
2,000	200	21,806.6	21,603	**6/10**	18,370.8	15,999	4/10
4,000	200	–	–	0/10	–	–	0/10
10,000	200	–	–	0/10	–	–	0/10
20,000	200	–	–	0/10	–	–	0/10
1,500	300	33,568.3	24,929	1/10	21,731.5	15,710	**2/10**
3,000	300	–	–	0/10	–	–	0/10
6,000	300	–	–	0/10	–	–	0/10
15,000	300	–	–	0/10	–	–	0/10
30,000	300	–	–	0/10	–	–	0/10
2,000	400	–	–	0/10	–	–	0/10
4,000	400	–	–	0/10	–	–	0/10
8,000	400	–	–	0/10	–	–	0/10
20,000	400	–	–	0/10	–	–	0/10
40,000	400	–	–	0/10	–	–	0/10

Table A.9: Results for Rec and RecZ with $\kappa = \frac{n}{2}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**1.1**	5	10/10	1.4	54	10/10
500	50	**1.1**	0	10/10	1.1	0	10/10
1,000	50	**11.1**	358	10/10	14.1	512	10/10
2,500	50	**5.9**	0	10/10	6.5	0	10/10
5,000	50	17.2	0	10/10	**16.9**	0	10/10
500	100	142.9	1,224	10/10	**38.5**	796	10/10
1,000	100	10.7	3	10/10	**10.6**	0	10/10
2,000	100	157.9	560	10/10	**114.3**	312	10/10
5,000	100	**39.3**	0	10/10	46.8	0	10/10
10,000	100	141.2	27	10/10	**128.0**	0	10/10
1,000	200	**2,512.2**	3,993	10/10	3,380.1	5,085	10/10
2,000	200	**2,004.4**	1,076	10/10	2,800.0	2,025	10/10
4,000	200	**4,169.9**	3,870	9/10	4,297.4	3,804	9/10
10,000	200	16,171.4	5,316	**7/10**	10,078.9	3,590	6/10
20,000	200	25,299.0	7,128	2/10	**14,242.2**	1,540	2/10
1,500	300	8,269.2	7,182	8/10	**5,983.6**	8,007	8/10
3,000	300	**10,973.2**	10,224	5/10	21,033.3	10,520	5/10
6,000	300	18,355.5	10,220	**5/10**	20,311.0	10,502	4/10
15,000	300	–	–	0/10	35,494.1	17,312	**1/10**
30,000	300	–	–	0/10	–	–	0/10
2,000	400	8,957.1	10,340	7/10	**6,617.2**	7,018	7/10
4,000	400	18,902.7	10,400	**5/10**	17,193.2	10,620	4/10
8,000	400	–	–	0/10	–	–	0/10
20,000	400	–	–	0/10	–	–	0/10
40,000	400	–	–	0/10	–	–	0/10

Table A.10: Results for Rec and RecZ with $\kappa = n$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	19.8	579	5/10	14.0	203	**9/10**
952	140	5.7	0	2/10	250.0	12,574	**10/10**
1,092	140	67.3	2,282	4/9	19.8	301	**9/9**
1,232	140	30.3	380	5/9	21.5	283	**9/9**
1,372	140	39.9	22	2/8	30.9	333	**8/8**
2,772	140	24.4	0	2/9	97.7	392	**8/9**
4,172	140	39.6	0	4/7	77.3	153	**7/7**
5,572	140	60.6	0	2/5	162.6	314	**5/5**
6,972	140	70.3	0	1/3	243.2	343	**3/3**
13,972	140	–	–	0/0	–	–	0/0
1,624	280	60.0	0	3/10	211.5	1,450	**10/10**
1,904	280	80.6	52	3/10	174.0	904	**10/10**
2,184	280	79.7	0	4/10	134.6	400	**10/10**
2,464	280	218.1	168	6/10	137.2	324	**10/10**
2,744	280	54.6	0	2/10	883.3	4,884	**10/10**
5,544	280	279.4	142	7/10	445.3	369	**10/10**
8,344	280	1,257.0	665	3/10	965.4	459	**10/10**
11,144	280	6,226.5	2,612	7/10	2,235.5	1,328	**10/10**
13,944	280	393.8	0	5/9	2,449.7	1,322	**9/9**
27,944	280	1,007.5	0	1/2	**777.9**	0	1/2

Table B.11: Results for Rec and RecZ with $\kappa = 1$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	26.2	656	6/10	227.8	6,333	**8/10**
952	140	4.2	0	2/10	175.3	1,696	**7/10**
1,092	140	197.3	8,400	2/9	241.3	1,404	**7/9**
1,232	140	8.0	0	5/9	779.9	10,965	**7/9**
1,372	140	27.5	0	2/8	477.8	2,169	**6/8**
2,772	140	72.1	47	2/9	990.9	1,744	**4/9**
4,172	140	131.3	9	4/7	1,159.9	1,261	**6/7**
5,572	140	66.0	0	2/5	492.5	336	**3/5**
6,972	140	**91.7**	0	1/3	93.6	0	1/3
13,972	140	–	–	0/0	–	–	0/0
1,624	280	68.8	48	3/10	281.1	720	**5/10**
1,904	280	63.6	0	3/10	1,135.4	2,257	**8/10**
2,184	280	314.6	164	4/10	1,343.9	2,364	**8/10**
2,464	280	223.4	36	6/10	678.8	899	**7/10**
2,744	280	335.7	480	2/10	4,640.8	3,302	**4/10**
5,544	280	**2,060.1**	2,254	8/10	2,533.7	972	8/10
8,344	280	18,394.1	3,153	**4/10**	4,975.3	1,569	3/10
11,144	280	**2,678.8**	1,233	7/10	7,921.5	1,754	7/10
13,944	280	747.0	0	3/9	**541.1**	0	3/9
27,944	280	2,276.4	0	1/2	**1,853.3**	0	1/2

Table B.12: Results for Rec and RecZ with $\kappa = 2$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	40.6	399	6/10	1,055.7	39,408	**8/10**
952	140	853.6	39,836	3/10	2,588.4	48,700	**7/10**
1,092	140	5.4	0	1/9	6,368.1	98,393	**7/9**
1,232	140	23.1	48	4/9	477.3	1,806	**7/9**
1,372	140	7.5	0	2/8	373.5	593	**4/8**
2,772	140	262.1	25	1/9	3,746.9	4,683	**3/9**
4,172	140	77.2	0	5/8	1,986.9	1,382	**6/8**
5,572	140	77.4	0	2/5	2,270.2	2,340	**3/5**
6,972	140	–	–	0/3	18,646.5	10,205	**1/3**
13,972	140	–	–	0/0	–	–	0/0
1,624	280	27.2	0	3/10	247.9	128	**4/10**
1,904	280	78.1	1	4/10	2,400.2	1,169	**8/10**
2,184	280	1,369.3	1,233	4/10	4,057.6	4,006	**6/10**
2,464	280	262.6	113	5/10	5,079.1	2,297	**7/10**
2,744	280	54.9	0	1/10	11,352.4	6,163	**5/10**
5,544	280	**1,602.9**	164	8/10	3,844.1	1,043	8/10
8,344	280	9,034.0	3,656	**3/10**	272.4	0	2/10
11,144	280	5,846.2	555	**7/10**	15,629.4	1,839	6/10
13,944	280	**748.1**	0	3/9	1,002.1	0	3/9
27,944	280	**2,091.3**	0	1/2	3,083.1	0	1/2

Table B.13: Results for Rec and RecZ with $\kappa = 4$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	285.6	13,149	6/10	736.4	19,182	**9/10**
952	140	235.9	2,194	2/10	1,002.8	4,134	**7/10**
1,092	140	11.6	0	2/9	497.7	1,574	**8/9**
1,232	140	9.6	0	2/9	569.4	1,378	**8/9**
1,372	140	53.8	1	1/8	938.5	3,615	**2/8**
2,772	140	458.5	421	3/10	6,002.4	4,256	**6/10**
4,172	140	2,123.7	909	1/8	10,049.7	5,882	**4/8**
5,572	140	–	–	0/5	13,690.3	7,110	**2/5**
6,972	140	**124.2**	0	2/3	2,150.3	350	2/3
13,972	140	–	–	0/0	–	–	0/0
1,624	280	39.4	0	2/10	2,103.3	1,586	**3/10**
1,904	280	55.6	0	4/10	6,291.3	3,516	**8/10**
2,184	280	1,930.9	1,462	3/10	5,731.6	2,785	**5/10**
2,464	280	2,156.0	908	5/10	16,428.5	10,132	**8/10**
2,744	280	76.2	0	1/10	23,191.0	6,471	**5/10**
5,544	280	6,336.6	1,709	**7/10**	13,784.7	1,808	5/10
8,344	280	11,589.3	1,952	**4/10**	24,426.6	978	2/10
11,144	280	**705.4**	0	4/10	5,168.0	382	4/10
13,944	280	**875.9**	0	3/9	6,305.3	328	3/9
27,944	280	**3,321.8**	0	2/3	4,241.9	0	2/3

Table B.14: Results for Rec and RecZ with $\kappa = 8$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	9,742.2	407,308	4/10	2,796.1	5,720	**7/10**
952	140	318.4	596	3/10	4,129.6	9,227	**9/10**
1,092	140	127.8	0	1/9	3,939.0	5,110	**3/9**
1,232	140	48.5	0	2/9	4,924.5	6,098	**4/9**
1,372	140	–	–	0/9	7,678.0	8,749	**6/9**
2,772	140	3,648.1	1,692	5/10	6,035.9	1,426	**6/10**
4,172	140	**1,676.2**	176	1/8	2,577.5	330	1/8
5,572	140	496.5	0	2/5	**346.8**	0	2/5
6,972	140	**352.0**	0	1/3	12,680.5	1,711	1/3
13,972	140	–	–	0/0	–	–	0/0
1,624	280	**178.6**	0	2/10	13,215.3	2,182	2/10
1,904	280	**8,821.4**	2,929	7/10	16,297.1	3,350	7/10
2,184	280	**3,053.7**	239	4/10	14,900.1	1,740	4/10
2,464	280	12,267.4	1,603	5/10	**11,182.6**	1,741	5/10
2,744	280	1,881.7	204	**6/10**	25,013.4	2,906	3/10
5,544	280	**2,056.3**	0	4/10	12,674.8	414	4/10
8,344	280	**1,285.8**	0	3/10	13,138.2	325	3/10
11,144	280	3,636.4	0	**5/10**	2,420.0	0	3/10
13,944	280	3,010.9	0	**3/9**	3,671.2	0	2/9
27,944	280	8,696.8	0	2/3	**8,655.9**	0	2/3

Table B.15: Results for Rec and RecZ with $\kappa = 16$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	6,401.1	47,425	3/10	**5,358.4**	6,668	3/10
952	140	**3,526.1**	4,662	2/10	8,718.9	6,926	2/10
1,092	140	31,891.0	41,336	**1/10**	–	–	0/10
1,232	140	**2,794.9**	1,370	2/9	15,388.4	8,048	2/9
1,372	140	**9,494.0**	8,470	5/10	12,613.8	5,549	5/10
2,772	140	26,993.7	13,109	**4/10**	36,912.0	4,432	2/10
4,172	140	–	–	0/9	32,307.9	10,775	**2/9**
5,572	140	–	–	0/10	–	–	0/10
6,972	140	–	–	0/8	–	–	0/8
13,972	140	–	–	0/10	–	–	0/10
1,624	280	30,317.0	3,088	**1/10**	–	–	0/10
1,904	280	–	–	0/10	–	–	0/10
2,184	280	–	–	0/10	–	–	0/10
2,464	280	–	–	0/10	–	–	0/10
2,744	280	–	–	0/10	–	–	0/10
5,544	280	–	–	0/10	–	–	0/10
8,344	280	–	–	0/10	–	–	0/10
11,144	280	–	–	0/10	–	–	0/10
13,944	280	–	–	0/10	–	–	0/10
27,944	280	–	–	0/9	–	–	0/9

Table B.16: Results for Rec and RecZ with $\kappa = \frac{n}{16}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	15,415.1	42,432	**3/10**	21,006.0	26,074	2/10
952	140	**2,257.0**	6,049	1/10	5,644.7	5,928	1/10
1,092	140	–	–	0/10	–	–	0/10
1,232	140	22,718.3	29,462	2/9	25,725.3	12,793	**3/9**
1,372	140	**7,752.9**	12,869	3/10	18,677.0	10,196	3/10
2,772	140	28,405.6	31,178	**2/10**	24,247.2	18,342	1/10
4,172	140	–	–	0/9	–	–	0/9
5,572	140	–	–	0/10	–	–	0/10
6,972	140	–	–	0/10	–	–	0/10
13,972	140	–	–	0/10	–	–	0/10
1,624	280	–	–	0/10	–	–	0/10
1,904	280	–	–	0/10	–	–	0/10
2,184	280	–	–	0/10	–	–	0/10
2,464	280	–	–	0/10	–	–	0/10
2,744	280	–	–	0/10	–	–	0/10
5,544	280	–	–	0/10	–	–	0/10
8,344	280	–	–	0/10	–	–	0/10
11,144	280	–	–	0/10	–	–	0/10
13,944	280	–	–	0/10	–	–	0/10
27,944	280	–	–	0/10	–	–	0/10

Table B.17: Results for Rec and RecZ with $\kappa = \frac{n}{8}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	**11,591.4**	22,628	3/10	12,893.3	12,083	3/10
952	140	13,409.6	27,703	**3/10**	18,925.8	15,628	2/10
1,092	140	**8,766.1**	21,938	2/10	34,654.4	27,442	2/10
1,232	140	8,847.2	19,499	**3/9**	39,016.8	22,241	2/9
1,372	140	10,476.9	39,370	**5/10**	31,589.7	23,824	4/10
2,772	140	32,072.7	23,498	**4/10**	31,803.2	16,867	1/10
4,172	140	–	–	0/10	18,064.3	2,554	**1/10**
5,572	140	–	–	0/10	–	–	0/10
6,972	140	–	–	0/10	–	–	0/10
13,972	140	–	–	0/10	–	–	0/10
1,624	280	–	–	0/10	–	–	0/10
1,904	280	–	–	0/10	–	–	0/10
2,184	280	–	–	0/10	–	–	0/10
2,464	280	–	–	0/10	–	–	0/10
2,744	280	–	–	0/10	–	–	0/10
5,544	280	–	–	0/10	–	–	0/10
8,344	280	–	–	0/10	–	–	0/10
11,144	280	–	–	0/10	–	–	0/10
13,944	280	–	–	0/10	–	–	0/10
27,944	280	–	–	0/10	–	–	0/10

Table B.18: Results for Rec and RecZ with $\kappa = \frac{n}{4}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	4,412.0	8,179	**7/10**	2,637.8	6,129	6/10
952	140	5,367.1	11,349	**7/10**	6,679.4	15,327	6/10
1,092	140	5,507.6	8,150	6/10	**5,062.7**	8,543	6/10
1,232	140	7,376.8	10,198	6/9	**7,326.1**	11,962	6/9
1,372	140	5,188.3	8,994	**8/10**	7,393.0	10,681	7/10
2,772	140	10,153.4	14,114	**9/10**	9,960.8	12,081	7/10
4,172	140	18,507.7	16,051	**10/10**	18,032.5	16,450	6/10
5,572	140	21,868.1	17,277	**7/10**	31,970.1	12,784	5/10
6,972	140	27,609.9	12,025	**3/10**	20,644.6	2,809	1/10
13,972	140	–	–	0/10	–	–	0/10
1,624	280	–	–	0/10	–	–	0/10
1,904	280	**26,509.8**	13,531	1/10	30,129.6	10,885	1/10
2,184	280	**29,023.3**	9,357	1/10	39,189.3	6,953	1/10
2,464	280	–	–	0/10	–	–	0/10
2,744	280	25,019.6	2,030	**1/10**	–	–	0/10
5,544	280	–	–	0/10	–	–	0/10
8,344	280	–	–	0/10	–	–	0/10
11,144	280	–	–	0/10	–	–	0/10
13,944	280	–	–	0/10	–	–	0/10
27,944	280	–	–	0/10	–	–	0/10

Table B.19: Results for Rec and RecZ with $\kappa = \frac{n}{2}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
812	140	**685.9**	3,528	8/8	728.3	3,273	8/8
952	140	**541.5**	1,860	10/10	672.1	1,750	10/10
1,092	140	**1,307.7**	1,621	10/10	1,374.9	1,842	10/10
1,232	140	**1,507.4**	3,457	9/9	2,078.8	3,721	9/9
1,372	140	683.9	1,397	9/9	**640.4**	1,100	9/9
2,772	140	**1,353.3**	1,452	9/9	2,128.0	2,737	9/9
4,172	140	**3,147.1**	4,612	10/10	3,542.6	4,869	10/10
5,572	140	2,408.4	2,840	9/10	4,908.3	3,250	**10/10**
6,972	140	**2,106.6**	1,818	7/7	4,022.5	3,033	7/7
13,972	140	10,335.1	4,171	**9/9**	11,179.8	3,416	8/9
1,624	280	7,592.0	8,959	6/10	**6,845.0**	7,891	6/10
1,904	280	26,541.4	15,070	**5/10**	30,973.6	18,810	4/10
2,184	280	13,585.1	6,632	4/10	21,024.6	12,338	**5/10**
2,464	280	**13,633.8**	8,339	5/10	19,923.0	16,718	5/10
2,744	280	10,159.0	9,380	4/10	22,480.3	13,832	**7/10**
5,544	280	30,020.3	12,308	6/10	**27,608.9**	12,558	6/10
8,344	280	**23,543.7**	14,244	2/10	37,359.4	20,074	2/10
11,144	280	–	–	0/10	42,844.9	4,912	**1/10**
13,944	280	–	–	0/9	–	–	0/9
27,944	280	–	–	0/10	–	–	0/10

Table B.20: Results for Rec and RecZ with $\kappa = n$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Appendix C. Results for $|I| >> |J|$

Finally, we show results for $|J| = 50$ and $|I|$ getting very large, such that $|I|/|J| \in \{5, 10, 20, 50, 100, 200, 300, 400, 500, 1000\}$. Note that $|I|/|J| \leq 100$ was already contained in instance set 1.

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	0.3	0	10/10	**0.2**	0	10/10
500	50	1.6	52	10/10	**0.6**	17	10/10
1,000	50	2,852.7	260,388	7/10	4.6	298	**10/10**
2,500	50	13.1	0	2/8	12.4	150	**8/8**
5,000	50	18.4	0	1/4	25.8	0	**4/4**
10,000	50	43.1	0	1/2	68.7	0	**2/2**
15,000	50	93.4	0	2/2	**44.5**	0	2/2
20,000	50	134.6	0	2/6	234.5	86	**6/6**
25,000	50	–	–	0/1	490.4	330	**1/1**
50,000	50	583.9	0	1/1	**307.1**	0	1/1

Table C.21: Results for Rec and RecZ with $\kappa = 1$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

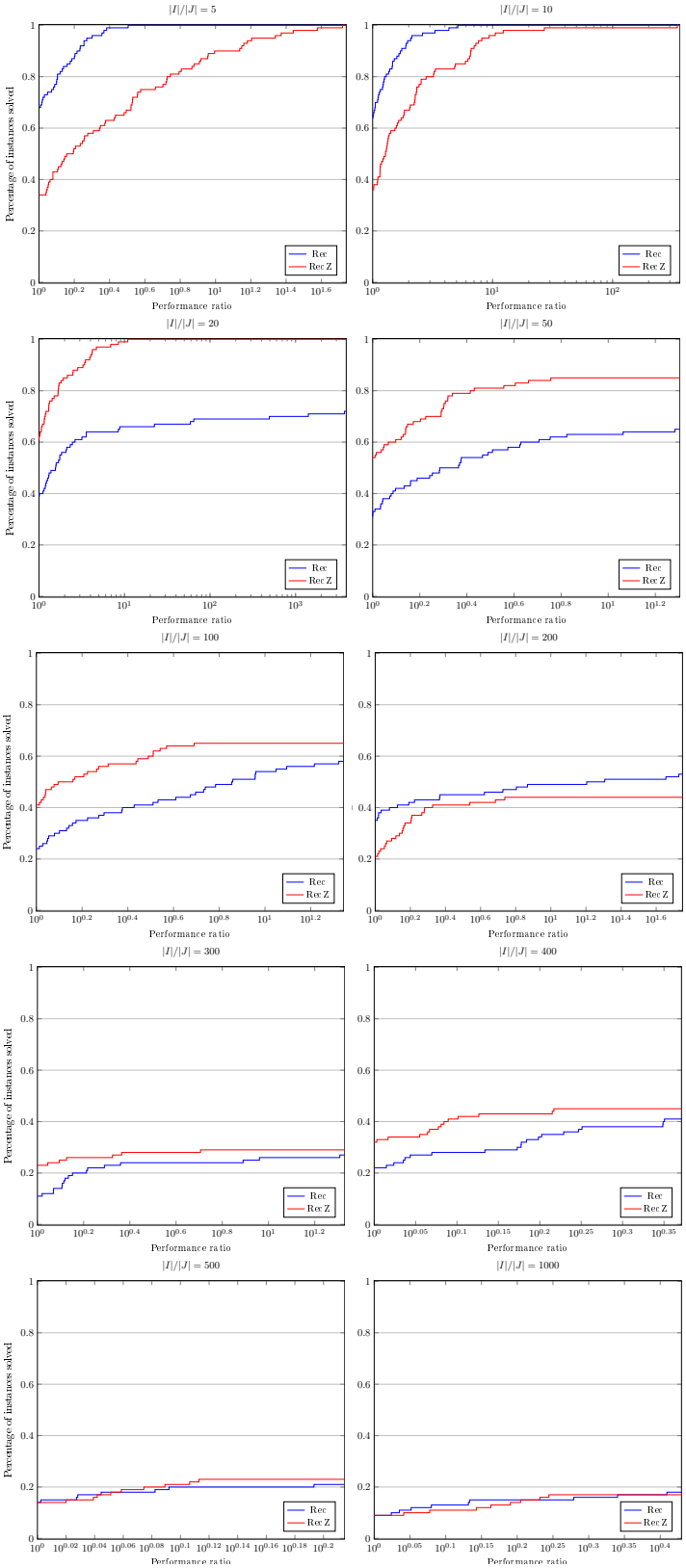

Figure C.6: Performance Profiles grouped by $|I|/|J|$.

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	0.3	0	10/10	**0.2**	0	10/10
500	50	**1.5**	51	10/10	1.6	51	10/10
1,000	50	807.2	101,534	6/10	8.6	207	**10/10**
2,500	50	21.6	15	4/8	59.3	353	**7/8**
5,000	50	19.3	0	1/4	229.0	476	**2/4**
10,000	50	–	–	0/2	–	–	0/2
15,000	50	122.2	0	1/2	**95.7**	0	1/2
20,000	50	**201.3**	0	4/6	240.8	0	4/6
25,000	50	–	–	0/1	–	–	0/1
50,000	50	1,749.8	0	1/1	**799.1**	0	1/1

Table C.22: Results for Rec and RecZ with $\kappa = 2$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.2**	0	10/10	0.3	0	10/10
500	50	**1.1**	0	10/10	1.3	0	10/10
1,000	50	485.8	58,522	5/10	18.6	208	**10/10**
2,500	50	76.9	622	3/8	423.9	2,388	**7/8**
5,000	50	15.9	0	2/5	636.1	848	**4/5**
10,000	50	65.0	0	1/2	**38.8**	0	1/2
15,000	50	–	–	0/2	–	–	0/2
20,000	50	322.6	0	5/6	**229.1**	0	5/6
25,000	50	**415.0**	0	1/1	454.2	0	1/1
50,000	50	1,924.9	0	1/1	**1,421.3**	0	1/1

Table C.23: Results for Rec and RecZ with $\kappa = 4$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.3**	0	10/10	0.6	0	10/10
500	50	3.1	52	10/10	**2.0**	0	10/10
1,000	50	39.1	2,082	2/10	92.9	2,240	**10/10**
2,500	50	7.4	0	2/8	581.1	2,035	**6/8**
5,000	50	16.7	0	2/5	124.4	208	**3/5**
10,000	50	73.7	0	1/2	**68.2**	0	1/2
15,000	50	–	–	0/2	–	–	0/2
20,000	50	352.0	0	4/6	**257.0**	0	4/6
25,000	50	655.7	0	2/3	**544.6**	0	2/3
50,000	50	2,110.7	0	1/1	**1,552.5**	0	1/1

Table C.24: Results for Rec and RecZ with $\kappa = 8$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.5**	0	10/10	6.3	68	10/10
500	50	**2.7**	0	10/10	7.7	2	10/10
1,000	50	2,039.5	229,533	3/10	152.1	1,774	**10/10**
2,500	50	149.9	227	4/8	542.7	1,542	**7/8**
5,000	50	22.6	0	2/5	**21.3**	0	2/5
10,000	50	**84.0**	0	2/3	85.6	0	2/3
15,000	50	265.3	0	1/2	**187.1**	0	1/2
20,000	50	370.6	0	7/7	**327.0**	0	7/7
25,000	50	650.5	0	3/3	**643.9**	0	3/3
50,000	50	3,069.4	0	3/3	**2,104.4**	0	3/3

Table C.25: Results for Rec and RecZ with $\kappa = 16$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**0.4**	0	10/10	5.3	91	10/10
500	50	**18.7**	248	10/10	119.4	2,077	10/10
1,000	50	222.9	2,032	9/10	2,531.6	4,598	**10/10**
2,500	50	**517.0**	723	10/10	779.4	452	10/10
5,000	50	3,456.9	3,738	10/10	**837.5**	260	10/10
10,000	50	18,935.0	14,852	8/10	1,978.7	337	**10/10**
15,000	50	31,829.6	10,181	3/10	3,646.4	399	**10/10**
20,000	50	–	–	0/10	10,188.7	2,101	**9/10**
25,000	50	–	–	0/10	9,498.6	1,655	**6/10**
50,000	50	–	–	0/10	–	–	0/10

Table C.26: Results for Rec and RecZ with $\kappa = \frac{n}{16}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**3.5**	71	10/10	12.4	276	10/10
500	50	**31.1**	612	10/10	64.5	568	10/10
1,000	50	**90.0**	635	10/10	173.9	607	10/10
2,500	50	1,081.3	3,948	10/10	**346.7**	418	10/10
5,000	50	8,275.4	16,551	10/10	**3,212.5**	2,490	10/10
10,000	50	21,012.3	40,939	**10/10**	16,437.7	11,558	4/10
15,000	50	–	–	0/10	–	–	0/10
20,000	50	–	–	0/10	–	–	0/10
25,000	50	–	–	0/10	–	–	0/10
50,000	50	–	–	0/10	–	–	0/10

Table C.27: Results for Rec and RecZ with $\kappa = \frac{n}{8}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	22.2	1,102	10/10	**18.4**	1,045	10/10
500	50	**11.2**	189	10/10	26.1	271	10/10
1,000	50	**83.2**	540	10/10	115.2	848	10/10
2,500	50	**1,141.7**	6,106	10/10	1,172.3	2,790	10/10
5,000	50	**3,996.5**	18,011	10/10	9,430.3	8,461	10/10
10,000	50	24,276.4	41,507	**10/10**	35,940.8	25,992	4/10
15,000	50	–	–	0/10	–	–	0/10
20,000	50	–	–	0/10	–	–	0/10
25,000	50	–	–	0/10	–	–	0/10
50,000	50	–	–	0/10	–	–	0/10

Table C.28: Results for Rec and RecZ with $\kappa = \frac{n}{4}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**3.4**	74	10/10	3.5	36	10/10
500	50	**3.0**	0	10/10	3.5	0	10/10
1,000	50	**12.9**	3	10/10	23.1	73	10/10
2,500	50	98.9	161	10/10	**90.8**	52	10/10
5,000	50	696.2	815	10/10	**514.1**	236	10/10
10,000	50	**3,667.9**	2,598	10/10	5,425.1	2,215	10/10
15,000	50	11,915.6	3,560	**10/10**	10,029.6	4,167	5/10
20,000	50	18,569.6	4,493	**9/10**	–	–	0/10
25,000	50	18,582.6	3,666	**5/10**	–	–	0/10
50,000	50	28,538.9	575	**1/10**	–	–	0/10

Table C.29: Results for Rec and RecZ with $\kappa = \frac{n}{2}$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).

Instances		Rec (IP)			RecZ		
$\|I\|$	$\|J\|$	runtime	nodes	solved/feasible	runtime	nodes	solved/feasible
250	50	**1.1**	5	10/10	1.4	54	10/10
500	50	**1.1**	0	10/10	1.1	0	10/10
1,000	50	**11.1**	358	10/10	14.1	512	10/10
2,500	50	**5.9**	0	10/10	6.5	0	10/10
5,000	50	17.2	0	10/10	**16.9**	0	10/10
10,000	50	**42.9**	0	10/10	46.7	0	10/10
15,000	50	100.4	0	10/10	**82.7**	0	10/10
20,000	50	**160.0**	0	10/10	160.4	0	10/10
25,000	50	**218.3**	0	10/10	235.6	0	10/10
50,000	50	**781.0**	0	10/10	1,039.1	0	10/10

Table C.30: Results for Rec and RecZ with $\kappa = n$ resource constraint. Tested instances are specified by number of nodes ($|I|$ and $|J|$). Averaged results over 10 instances per instance size. Bold means more instances have been solved (in case of equality the faster average runtime is printed in bold).