European Journal of Operational Research 298 (2022) 821–833

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Exact and heuristic algorithms for the maximum weighted submatrix coverage problem

Markus Sinnl

Institute of Production and Logistics Management/JKU Business School, Johannes Kepler University Linz, Linz, Austria

ARTICLE INFO

Article history:
Received 8 September 2020
Accepted 16 July 2021
Available online 24 July 2021

Keywords:
Combinatorial optimization
Data mining
Branch-and-cut
Benders decomposition
Local search

ABSTRACT

The maximum weighted submatrix coverage problem is a recently introduced problem with applications in data mining. It is concerned with selecting K submatrices of a given numerical matrix such that the sum of the matrix-entries, which occur in at least one of the selected submatrices, is maximized. In the paper introducing the problem, a problem-specific constraint programming approach was developed and embedded in a large neighborhood-search to obtain a heuristic. A compact integer linear programming formulation was also presented, but deemed inefficient due to its size.

In this paper, we introduce new integer linear programming formulations for the problem, one of them is based on Benders decomposition. The obtained Benders decomposition-based formulation has a nice combinatorial structure, i.e., there is no need to solve linear programs to separate Benders cuts. We present preprocessing procedures and valid inequalities for all formulations. We also develop a greedy randomized adaptive search procedure for the problem, which is enhanced with a local search.

A computational study using the instances from literature is done to evaluate the effectiveness of our new approaches. Our algorithms manage to find improved primal solutions for ten out of 17 real-world instances, and optimality is proven for two real-world instances. Moreover, for over 700 of 1617 large-scale synthetic instances, our algorithms find improved primal solutions compared to the heuristics from the literature.

1. Introduction

The *maximum weighted submatrix coverage problem (MWSCP)* is a recently introduced problem with applications in data mining (Derval, Branders, Dupont, & Schaus, 2019a). The MWSCP is concerned with selecting K submatrices of a given numerical matrix, such that the sum of the matrix-entries, which occur in at least one of the selected submatrices, is maximized. The problem is related to the family of so-called *biclustering problems*, which got much attention by researchers in the previous years, see, e.g., the surveys (Busygin, Prokopyev, & Pardalos, 2008; Madeira & Oliveira, 2004; Tanay, Sharan, & Shamir, 2005). In biclustering problems, we are also interested in selecting submatrices of a given matrix. The selected submatrices are called biclusters in this context, as the problem class emerged as extension of clustering problems. Contrary to the MWSCP, in biclustering problems to goal is usually to select submatrices satisfying some given similarity criteria. Both the MWSCP and biclustering problems have applications in bioinformatics (Derval et al., 2019a; Madeira & Oliveira, 2004), other applications of the MWSCP include analysis of migration data or preprocessing for data visualization (Derval et al., 2019a). A formal definition of the MWSCP is as follows.

Definition 1 (The maximum weighted submatrix coverage problem (MWSCP))**.** Let $\mathcal{M} \in \mathbb{R}^{m \times n}$ be an $m \times n$ matrix, and $\mathcal{M}_{ij}$ indicate the value of the cell in row i and column j. Let $R=\{1,\dots,m\}$ and $C=\{1,\dots,n\}$ indicate the index-set of the rows and columns of $\mathcal{M}$, respectively. Let K be a given integer and $(\mathcal{R}_k, \mathcal{C}_k) \subset (R, C)$, $k = 1,\dots,K$, i.e., each $(\mathcal{R}_k, \mathcal{C}_k)$ is a (possibly empty) submatrix of $\mathcal{M}$. For a set of K submatrices $(\mathcal{R}_k, \mathcal{C}_k)$, let $COV(\cup_{k=1,\dots,K}(\mathcal{R}_k, \mathcal{C}_k), i, j) = 1$, iff cell (i, j) is occurring in a least one of the submatrices $(\mathcal{R}_k, \mathcal{C}_k)$, $k = 1,\dots,K$, and zero otherwise. The goal of the MWSCP is to find K submatrices $(\mathcal{R}_1^*, \mathcal{C}_1^*),\dots,(\mathcal{R}_K^*, \mathcal{C}_K^*)$, which maximize $\sum_{i\in R, j\in C} \mathcal{M}_{ij} \cdot COV(\cup_{k=1,\dots,K}(\mathcal{R}_k, \mathcal{C}_k), i, j)$

We note that even for a fixed set of rows $\mathcal{R}_1,\dots,\mathcal{R}_K$, finding the optimal columns is NP-hard (Derval et al., 2019a). The MWSCP is an extension of the *maximal-sum submatrix problem* (introduced in Branders, Schaus, & Dupont, 2017), which is obtained for $K = 1$.

E-mail address: markus.sinnl@jku.at

https://doi.org/10.1016/j.ejor.2021.07.035

	1	2	3	4	5	6	7
1	−4.4	−2.2	1.0	2.8	−0.3	−1.8	−1.9
2	2.3	2.4	−1.1	5.3	1.0	1.8	0.8
3	−4.1	1.0	0.4	−4.2	3.0	2.1	−2.2
4	−5.6	1.8	4.1	3.5	−2.2	−4.2	−4.3
5	0.3	2.2	−3.9	3.2	4.1	−1.1	−2.1
6	−4.3	−2.0	−3.1	4.1	2.0	5.1	1.1

(a) Exemplary instance

	1	2	3	4	5	6	7
1	−4.4	−2.2	1.0	2.8	−0.3	−1.8	−1.9
2	2.3	2.4	−1.1	5.3	1.0	1.8	0.8
3	−4.1	1.0	0.4	−4.2	3.0	2.1	−2.2
4	−5.6	1.8	4.1	3.5	−2.2	−4.2	−4.3
5	0.3	2.2	−3.9	3.2	4.1	−1.1	−2.1
6	−4.3	−2.0	−3.1	4.1	2.0	5.1	1.1

(b) Optimal solution for $K = 2$

Fig. 1. Exemplary instance of the MWSCP and optimal solution for $K = 2$. The selected submatrices are given in red ($\mathcal{R} = \{1, 3, 4\}$, $\mathcal{C} = \{3, 4\}$) and blue ($\mathcal{R} = \{2, 3, 5, 6\}$, $\mathcal{C} = \{2, 4, 5, 6\}$), note that they overlap in cell (3,4). The solution value is 41.8. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 1 gives an exemplary instance of the MWSCP together with the optimal solution for $K = 2$.

In Derval et al. (2019a), various solution approaches were proposed. The main focus was the development of a problem-specific constraint programming (CP) approach, which was then embedded as a subroutine in a large neighborhood search (LNS) to obtain a heuristic, as the CP approach could only solve very small instances to optimality. Moreover, a greedy heuristic also using CP, and a compact integer linear programming (ILP) formulation as well as a compact integer quadratically constrained programming (IQCP) formulation was presented. The computational study of Derval et al. (2019a) concentrated on the heuristic performance of the proposed algorithms, i.e., the quality of upper bounds obtained by the CP, ILP and IQCP was not analyzed. The study used synthetic as well as real-world instances and the LNS was shown to be the most effective in providing good results within short computing times (for the synthetic instances, near-optimal solutions are known by construction, so the obtained solution quality could be classified without having an exact method solving the instances to optimality). In particular, the ILP and IQCP approaches were only tested on smaller instances, and dismissed due to their poor performance, which was explained by the size of the formulations.

1.1. Contribution and outline

In this paper, we introduce new ILP formulations for the problem and develop branch-and-cut algorithms based on them. These frameworks also include preprocessing procedures and valid inequalities. We also present a greedy randomized adaptive search procedure (GRASP) and a local search for the problem. These heuristics are used to create initial solutions for the branch-and-cut algorithms, and we also consider their performance as "standalone" algorithms to provide heuristic solutions for the MWSCP. A computational study on the instances from Derval et al. (2019a) is done to assess the performance of our proposed approaches. Our algorithms manage to find improved primal solutions for ten out of 17 real-world instances, and optimality is proven for two real-world instances. Moreover, for over 700 of 1617 large-scale synthetic instances, our algorithms find improved primal solutions compared to the heuristic from literature.

The paper is organized as follows: In Section 2, we first recall the ILP-formulation of Derval et al. (2019a), and show how it can be simplified to obtain a new formulation with fewer variables. We also prove that the linear programming (LP) relaxation bound of the formulation of Derval et al. (2019a) is just the sum of the positive entries in the matrix. Moreover, we introduce valid inequalities and preprocessing, and present implementation details for a branch-and-cut framework based on our new formulation. In Section 3, we apply Benders decomposition to obtain another new formulation and describe a branch-and-cut approach based on it. Section 4 contains the description of the GRASP and local search. Finally, Section 5 contains a computational study using the instances of Derval et al. (2019a) and Section 6 concludes the paper.

2. A first reformulation

Let binary variable $r_i^k = 1$ iff row $i \in R$ is selected in submatrix k and let binary variable $c_j^k = 1$ iff column $j \in C$ is selected in submatrix k. Let binary variable $e_{ij}^k = 1$, $i \in R$, $j \in C$, iff cell (i, j) is chosen in submatrix k and let binary variable $s_{ij} = 1$, $i \in R$, $j \in C$, iff the cell (i, j) is chosen in at least one of the selected submatrices. Using these variables, the MWSCP is formulated in Derval et al. (2019a) as ILP as follows, we denote this formulation as ($ILP_{\text{Derval et al. (2019a)}}$).

$$\max \sum_{i \in R, j \in C} \mathcal{M}_{ij} s_{ij}$$

$$s_{ij} \geq e_{ij}^k \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{SE-LINK1}$$

$$s_{ij} \leq \sum_{k=1}^{K} e_{ij}^k \quad \forall i \in R, j \in C \tag{SE-LINK2}$$

$$e_{ij}^k + 1 \geq r_i^k + c_j^k \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{RCE-LINK1}$$

$$2e_{ij}^k \leq r_i^k + c_j^k \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{RCE-LINK2}$$

$$s_{ij} \in \{0, 1\} \quad \forall i \in R, j \in C$$

$$e_{ij}^k \in \{0, 1\} \quad \forall (i, j) \in R \times C; k = 1, \ldots, K$$

$$r_i^k \in \{0, 1\} \quad \forall i \in R; k = 1, \ldots, K$$

$$c_j^k \in \{0, 1\} \quad \forall j \in C; k = 1, \ldots, K$$

Constraints (SE-LINK1) and (SE-LINK2) link the **s** and **e**-variables, i.e., the first set of constraints ensure that s_{ij} is one, if entry (i, j) is occuring in one of the selected submatrices, and the second set ensures that in at least one submatrix, entry ij is contained, if the

corresponding s_{ij}-variable is one. The remaining two sets of constraints (RCE-LINK1) and (RCE-LINK2), link the **r** and **c**-variables with the **e**-variables, i.e., the first set ensures that the variable for cell (i, j) is one, if both the row i and column j are selected for a given submatrix, and the second set make sure that the r_i and c_j-variables are one, if cell (i, j) is selected in a submatrix. The formulation has $O(K \cdot |R| \cdot |C|)$-constraints and $O(K \cdot |R| \cdot |C|)$-variables. It showed poor performance in the computational study (Derval et al., 2019a) already for small instances, and thus was not considered for the main-part of their computational experiments.

As a first potential improvement, we note that constraints (RCE-LINK2) can be disaggregated to the following two sets of constraints

$$e_{ij}^k \leq r_i^k \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{RE-LINK}$$

$$e_{ij}^k \leq c_j^k \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{CE-LINK}$$

Let the resulting formulation be $(ILP'_{\text{Derval et al. (2019a)}})$. We note that for a given k, constraints (RCE-LINK1), (RE-LINK), (CE-LINK) share similarities with the *boolean quadric polytope* (BQP, also known as *cut polytope*) (Letchford & Sørensen, 2014; Padberg, 1989), and ILP approaches using such formulations often suffer from bad LP-relaxation bounds (see, e.g., Billionnet, 2005). Regarding the quality of the LP-relaxation bounds of $(ILP'_{\text{Derval et al. (2019a)}})$, we can show the following.

Theorem 1. *The value of the LP-relaxation of* $(ILP'_{\text{Derval et al. (2019a)}})$ *is* $\sum_{i,j} \max(\mathcal{M}_{ij}, 0)$ *for* $K \geq 2$.

Proof. The value is the sum of all positive entries in $\mathcal{M}$. As we have $0 \leq s_{ij} \leq 1$, this is the largest possible value of the objective function of the LP-relaxation. A solution $(\mathbf{r}^*, \mathbf{c}^*, \mathbf{s}^*, \mathbf{e}^*)$ to the LP-relaxation achieving this value can be constructed as follows: Set $r_i^{*k} = c_j^{*k} = \frac{1}{k}$ for all $i \in R, j \in C, k = 1, \ldots, K$. For these values of $\mathbf{r}^*, \mathbf{c}^*$, constraints (RCE-LINK1) are trivially satisfied for any value of **e**.

For all $i \in R, j \in C$ with $\mathcal{M}_{ij} > 0$, set $s_{ij}^* = 1$ and $e_{ij}^{*k} = \frac{1}{k}$ for $k = 1, \ldots, K$. With these values, constraints (SE-LINK1), (SE-LINK2) and also (RE-LINK), (CE-LINK) are satisfied for $i \in R, j \in C$ with $\mathcal{M}_{ij} > 0$.

Finally, set the remaining variables s_{ij}^* and e_{ij}^{*k} to zero. With these values, constraints (SE-LINK1), (SE-LINK2) and also (RE-LINK), (CE-LINK) are satisfied for $i \in R, j \in C$ with $\mathcal{M}_{ij} \leq 0$ □

We note that for the formulations (*ILP*) and (*B*) introduced in the following, similar solutions can be constructed. However, we then also propose valid inequalities which cut off such solutions.

2.1. A formulation with less variables

Let binary variable $f_{ij}^k = 1$, $(i, j) \in R \times C : \mathcal{M}_{ij} > 0$; iff cell (i, j) is selected in submatrix k. The MWSCP can be formulated as follows, we denote the formulation by (*ILP*).

$$\max \sum_{i \in R, j \in C} \mathcal{M}_{ij} s_{ij}$$

$$s_{i'j} + 1 \geq f_{ij}^k + f_{i'j'}^k \quad \forall (i, j), (i', j') \in R \times C : \mathcal{M}_{ij} > 0, \mathcal{M}_{i'j'} > 0, \mathcal{M}_{i'j} < 0; k = 1, \ldots, K \tag{SF-LINK1}$$

$$s_{ij} \leq \sum_{k=1}^{K} f_{ij}^k \quad \forall (i, j) \in R \times C : \mathcal{M}_{ij} > 0 \tag{SF-LINK2}$$

$$s_{ij} \in \{0, 1\} \quad \forall i \in R, j \in C$$

$$f_{ij}^k \in \{0, 1\} \quad \forall (i, j) \in R \times C : \mathcal{M}_{ij} > 0$$

While formulation (*ILP*) still has $O(K \cdot |R| \cdot |C|)$-variables, in the instances from literature, the number of cells (i, j) with $\mathcal{M}_{ij} > 0$ is usually quite small compared to the total size of the matrix. Thus, in practice the size of the model is dominated by the number of **s**-variables, i.e., $O(|R| \cdot |C|)$. The formulation exploits the fact that the objective is the maximization of the sum of selected entries and that some entries have negative values. Thus, these negative entries will only be taken in the solution, if some combination of positive entries in a submatrix (i.e., all combinations of rows and columns of the selected positive entries) force them to be there, and this is ensured with constraints (SF-LINK1). We note that this also holds for cells with $\mathcal{M}_{ij} = 0$, but we do not need to explicitly force the s_{ij}-variables of such cells to be taken, as they will not change the value objective function. In other words, this means that the formulation (*ILP*) also allows some infeasible solutions with respect to cells with $\mathcal{M}_{ij} = 0$, but for any given binary vector **f** a feasible solution can be found with the same objective function value by taking the submatrices implied by the values of **f**. Alternatively, one could define the **f**-variables for all (i, j) with $\mathcal{M}_{ij} \geq 0$, and modify the conditions on the coefficients in constraints (SF-LINK1) and (SF-LINK2) to "$\geq$" and "$\leq$" instead of strict inequalities.

2.2. Valid inequalities

Next, we present some valid inequalities for (*ILP*). The first family operates purely on the **s**-variables and thus is valid for all the formulations presented in the paper, as they all use these variables.

Theorem 2. *Let* $S = \{(i_1, j_1), \ldots, (i_L, j_L)\}$ *be a set of* $L \geq K + 1$ *matrix-cells, where each i-coordinate and j-coordinate appears only once. Let* $T(S) = \{(i_\ell, j_{\ell'}) \vee (i_{\ell'}, j_\ell) : (i_\ell, j_\ell), (i_{\ell'}, j_{\ell'}) \in S\}$ *i.e., a set, which for each pair of cells* $(i_\ell, j_\ell), (i_{\ell'}, j_{\ell'})$ *in S contains one of the two pairs* $(i_\ell, j_{\ell'}), (i_{\ell'}, j_\ell)$ *obtained by switching the j-coordinate. The following inequalities are valid for* (*ILP*).

$$\sum_{(i,j) \in S} s_{ij} \leq K + \sum_{(i,j) \in T(S)} s_{ij} \tag{SUBMTX}$$

Proof. If only K of the variables s_{ij} on the left-hand-side are one, the inequality is trivially valid. Thus, we only need to consider the case of $K + 1$ or more variables having the value one. First, suppose $K + 1$ are one. As there are only K submatrices, this means at least two cells, say (i_ℓ, j_ℓ) and $(i_{\ell'}, j_{\ell'})$ of S with their associated s-variables set to one must occur in the same submatrix in a solution. This means, both cells $(i_\ell, j_{\ell'})$ and $(i_{\ell'}, j_\ell)$ must also occur in this submatrix, and for one of them, the corresponding s-variable is on the right-hand-side of the inequality. The same argument also holds, if their are more than $K + 1$ variables which are one on the left-hand-side, as each must be in a submatrix together with other cells, where the variables are also one, and each of these pairs forces a variable on the right-hand-side to be one, due to the uniqueness-assumption on the coordinates. □

We note that constraints (SUBMTX) would also be valid if the set $T(S)$ would contain both pairs $(i_\ell, j_{\ell'}), (i_{\ell'}, j_\ell)$, however, the resulting constraints would be weaker as the right-hand-side would be larger. There is an exponential number of constraints (SUBMTX) even for a fixed set S due to the definition of $T(S)$. Moreover, there is also an exponential number of ways how S can be chosen. Thus, we do not add all these constraints at initialization, but separate them on-the-fly, i.e., we use branch-and-cut. Separation of (SUBMTX) is discussed in Section 2.3. It can be easily seen that constraints (SUBMTX) can potentially cut off the solutions constructed in the proof of Theorem 1 by picking a set S where all cells have $\mathcal{M}_{ij} > 0$ and a set $T(S)$ where all cells have $\mathcal{M}_{ij} < 0$ if such sets S and $T(S)$ exist for the considered instance.

The next set of valid inequalities is in the space of both **s**-variables and **f**-variables and is based on a similar idea. The in-

equalities can be seen as a generalization of inequalities (SF-LINK1) and also share some similarities with the clique-inequalities of the BQP. These inequalities are also of exponential number and their separation is discussed in Section 2.3.

Theorem 3. *Let $S = \{(i_1, j_1), \ldots, (i_L, j_L)\}$ be a set of $L \geq 2$ matrix-cells with $\mathcal{M}_{ij} > 0$, where each i-coordinate and j-coordinate appears only once. Let $T(S) = \{(i_\ell, j_{\ell'}) \vee (i_{\ell'}, j_\ell) : (i_\ell, j_\ell), (i_{\ell'}, j_{\ell'}) \in S\}$ i.e., a set, which for each pair of cells $(i_\ell, j_\ell), (i_{\ell'}, j_{\ell'})$ in S contains one of the two pairs $(i_\ell, j_{\ell'}), (i_{\ell'}, j_\ell)$ obtained by switching the j-coordinate. The following inequalities are valid for (ILP).*

$$\sum_{(i,j)\in S} f^k_{ij} \leq 1 + \sum_{(i,j)\in T(S)} s_{ij} \quad k = 1, \ldots, K \qquad \text{(SF-LINKG)}$$

Proof. For any pair of cells (i, j), (i', j') which is in submatrix k if the corresponding variables on the left-hand-side are one, the cells (i, j') and (i', j) must also be in the submatrix (and thus also chosen in the overall solution), and one of the two s-variables $s_{i'j}$ or $s_{ij'}$ is on the right-hand-side for each of these pairs. □

The formulation (*ILP*) (and also ($ILP'_{\text{Derval et al. (2019a)}}$)) suffers from symmetry: Suppose $K = 2$ and a feasible solution consists of two submatrices A and B. Both the solution where $\mathbf{f^1}$ encodes A and $\mathbf{f^2}$ encodes B, and the opposite solution, where $\mathbf{f^2}$ encodes A and $\mathbf{f^1}$ encodes B are valid. In order to break these symmetries (to avoid the exploration of unnecessary nodes containing symmetric solutions in the branch-and-cut tree), we propose the following symmetry breaking constraints (ILP-SYM). They stipulate that the submatrices in a solution are ordered in a lexicographically minimal way with respect to the indices of their first entry.

$$f^{k+1}_{ij} \leq \sum_{i'=1}^{i-1} \sum_{j'=1}^{n:\mathcal{M}_{i'j'}>0} f^k_{i'j'} + \sum_{j'=1}^{j:\mathcal{M}_{ij'}>0} f^k_{ij'} \quad \forall i \in R, j \in C : \mathcal{M}_{ij} > 0, \; k = 1, \ldots, K-1 \qquad \text{(ILP-SYM)}$$

2.3. Implementation details

In this section, we describe details of the branch-and-cut algorithm we developed based on formulation (*ILP*). Note that in all our approaches, we remove all rows/columns where all entries in $\mathcal{M}$ for these rows/columns are non-positive, as there always exists an optimal solution without such rows/columns. Moreover, before starting the branch-and-cut algorithm, we switch the rows and columns of the input matrix if $m > n$. This is done as our heuristics (see Section 4) work in an asymmetric fashion and the operations on the rows are computationally more costly.

Initialization and separation of inequalities

At the start of the branch-and-cut, we do not initialize the solver with the complete model (*ILP*), but only with constraints (SF-LINK2) (we note that inequalities (SF-LINK1) are unnecessary in an initial relaxation, due to the results of Theorem 1). Inequalities (ILP-SYM) are not added at initialization, but separated by enumeration.

Let $\mathbf{s}^*$ be the fractional solution values of the $\mathbf{s}$-variables in the LP-relaxation at a branch-and-cut node. The separation algorithm for (SUBMTX) is given in Algorithm 1 and works in an iterative fashion, i.e., more than one violated inequality (SUBMTX) can be found.

By design of the separation algorithm, each variable s_{ij} with $\mathcal{M}_{ij} > 0$ will only occur in at most one violated inequality. To construct an inequality (SUBMTX), the algorithm row-wise builds a set S by considering cells (i, j) with $\mathcal{M}_{ij} > 0$ and $s^*_{ij} \geq 1 - \epsilon$, with $\epsilon = 0.00001$. While this criterion on the value of s^*_{ij} may seem restrictive, in practice even with this restriction, a huge number of violated inequalities could be found. The set $T(S)$ is updated each

Input: matrix $\mathcal{M}$, fractional solution values $\mathbf{s}^*$, cutlimit `maxCut`, $\epsilon = 0.00001$, `scoreCutoff`
Output: set `VIneqs` of violated inequalities (SUBMTX), possibly updated `scoreCutoff`

```
usedInS_ij ← false, ∀i ∈ R, j ∈ C : M_ij > 0
nViolatedIneqs ← 0
for i ∈ R do
    for j ∈ C do
        // pick the next available cell to start building a new set S
        if s*_ij ≥ 1 − ε and M_ij > 0 and not usedInS_ij then
            S ← (i, j), T(S) ← ∅
            usedRows_i' ← false, ∀i' ∈ R : i' ≠ i; usedRows_i = true
            usedCols_j' ← false, ∀j' ∈ C : j' ≠ j; usedCols_j = true
            // build the set S row-wise, randomly shuffle the rows to obtain a more diverse set of inequalities
            randomly shuffle R
            for i' ∈ R : i ≠ i' and not usedRows_i' do
                j* ← null, j*_score ← scoreCutoff, T(S, j*) ← ∅
                for j' ∈ C do
                    // try to find cell in row i' to grow S
                    if s*_i'j' ≥ 1 − ε and M_i'j' > 0 and not usedInS_i'j' and not usedCols_j' then
                        score ← 0, T(S, j') ← ∅
                        // calculate the score for adding (i', j') to S and also keep track of the set T(S, j') which would be obtained by adding (i', j') to T(S)
                        for (i'', j'') ∈ S do
                            if s*_i''j' < s*_i'j'' then
                                score ← score + s*_i''j',
                                T(S, j) ← T(S, j) ∪ (i'', j')
                            else
                                score ← score + s*_i'j'',
                                T(S, j) ← T(S, j) ∪ (i', j'')
                        if score < j*_score then
                            j* ← j, j*_score ← score,
                            T(S, j*) ← T(S, j)
                // grow the set S if a suitable cell (i', j*) was found and update T(S) accordingly
                if j* is not null and j*_score < 1 then
                    S ← S ∪ (i', j*), T(S) ← T(S) ∪ T(S, j*)
                    usedRows_i' ← true, usedCols_j* ← true
            if (SUBMTX) induced by S and T(S) is violated by s* then
                usedInS_i'j' ← true, ∀(i', j') ∈ S
                VIneqs ← VIneqs ∪ (SUBMTX) induced by S and T(S)
                nViolatedIneqs ← nViolatedIneqs + 1
                if nViolatedIneqs ≥ maxCut then
                    return VIneqs, scoreCutoff
if |VIneqs| ≤ 10 then
    scoreCutoff ← scoreCutoff + 0.1
return VIneqs, scoreCutoff
```

Algorithm 1: Separation heuristic for inequalities (SUBMTX).

time a new cell is added to S. This is done in the following way: Let (i', j') be a cell to be added to S. This means (i, j') or (i', j) would need to be added to $T(S)$ for all (i, j) already in S. To update $T(S)$, we pick the cell (i, j') if $s^*_{ij'} < s^*_{i'j}$ and (i', j) otherwise. This is done in order to obtain an inequality (SUBMTX) with a small right-hand-side value.

For each row, among all the cells (i, j) satisfying $\mathcal{M}_{ij} > 0$ and $s^*_{ij} \geq 1 - \epsilon$, at most one is added to S according to the following greedy rule: Let (i', j') be a candidate cell to be added to S. We calculate a score which consists of the sum of LP-values of all the corresponding s-variables which would be added to $T(S)$ according to the above update rule. For each row, we add at most one cell, namely the one with the smallest score. If the value of the smallest score is larger than the given value $scoreCutoff$, no cell is added for this row. The value $scoreCutoff$ is initialized with 0.1 and globally (i.e., for all subsequent calls of the separation routine) increased by 0.1 if less than ten violated inequalities were found during the current call of the separation routine. This is done to obtain inequalities with a small value of $\sum_{(i,j)\in T(S)} s^*_{ij}$ in a fast way, as due to the size of the instances a more exhaustive procedure (e.g., checking always all available rows for finding the next cell to add to S, instead of checking one row at-a-time) was too time consuming in preliminary experiments. We randomly shuffle the rows in each iteration of the separation routine. This is done to obtain a diverse set of inequalities containing many different rows.

When a cell (i', j') gets added to S, we will not consider column j' in the remainder of the current separation pass. Once we have iterated through all rows in this fashion, we check if the inequality induced by S and $T(S)$ is violated, and add it in case of violation. Moreover, all the cells in S are removed from further consideration for the separation, and then the separation procedure is restarted for the next available cell with $\mathcal{M}_{ij} > 0$ and $s^*_{ij} \geq 1 - \epsilon$. We have a limit $maxCut = 100$ of violated inequalities which can be added.

Separation of inequalities (SF-LINKG) uses a similar procedure, however, in this separation, we do not iterate through all rows and then check violation, but check violation after each row, and add the corresponding inequality, if it is violated (and then restart separation with the next available cell). Moreover, in each row, we use the first available (i', j') for which the score is smaller than $s^*_{i'j'}$. We again use the same limit $maxCut$ of violated inequalities to be added before stopping the separation.

In case we encounter an integer solution $(\mathbf{s}^*, \mathbf{f}^*)$ in separation, we just separate inequalities (SF-LINK1) by enumeration and add at most one for each (i, j). This is done to avoid overloading the LP-relaxation with unnecessary cuts, as (infeasible) integer solutions are quite often produced by heuristics of CPLEX, the ILP-solver we used. In the root-node of the branch-and-bound tree, we pursue a very aggressive cut-generation policy and allow for up to $nCutLoopIterations = 200$ cut-loop iterations (with at most five consecutive iterations without upper bound-improvement) to build a strong initial model, while at the other branch-and-bound nodes we only allow for one cut-loop iterations to allow for a quick node-throughput. In Section 5.2 we also provide results with other values for $maxCut$ and $nCutLoopIterations$. We provide a feasible starting solution by calling the heuristic from Section 4 ten times and taking the best solution obtained.

3. A Benders Decomposition-based reformulation

In this section, we obtain a formulation without **e**-variables from ($ILP'_{\text{Derval et al. (2019a)}}$) using Benders feasibility cuts. As a first step for removing the **e**-variables, we can replace (SE-LINK1) and (RCE-LINK1) by combining them to $s_{ij} + 1 \geq r^k_i + c^k_j, \forall i \in R, j \in C; k = 1, \ldots, K$. Now a given binary solution $(\mathbf{r}^*, \mathbf{c}^*, \mathbf{s}^*)$, is feasible, if there exists a solution to the following system (it can be easily seen that the originally binary **e**-variables can be relaxed to be continuous)

$$s^*_{ij} \leq \sum_{k=1}^{K} e^k_{ij} \quad \forall i \in R, j \in C \tag{SE-LINK2-fixed}$$

$$e^k_{ij} \leq r^{*k}_i \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{RE-LINK-fixed}$$

$$e^k_{ij} \leq c^{*k}_j \quad \forall i \in R, j \in C; k = 1, \ldots, K \tag{CE-LINK-fixed}$$

To project out the **e**-variables, we use Benders feasibility cuts based on the above system. We observe, that for a given binary solution $(\mathbf{r}^*, \mathbf{c}^*, \mathbf{s}^*)$, above system decomposes for each (i, j). For a given binary solution and entry (i, j), let α_{ij} be the dual variable for (SE-LINK2-fixed), β^k_i be the dual variables for (RE-LINK-fixed) and γ^k_j be the dual variables for (CE-LINK-fixed). The corresponding dual reads as follows.

$$\max \quad s^*_{ij}\alpha_{ij} + \sum_{k=1}^{K}(-r^{*k}_i\beta^k_i - c^{*k}_j\gamma^k_j) \tag{D-OBJ}$$

$$\alpha_{ij} - \beta^k_i - \gamma^k_j = 0 \quad k = 1, \ldots, K \tag{D-CONS}$$
$$(\alpha_{ij}, \boldsymbol{\beta}, \boldsymbol{\gamma}) \geq 0$$

A feasible solution for the dual always exists (every dual variable set to zero). Thus, the primal system is infeasible, only if the dual is unbounded (Wolsey & Nemhauser, 1999). A Benders feasibility cut cutting off the infeasible $(\mathbf{r}^*, \mathbf{c}^*, \mathbf{s}^*)$ is given by $\alpha^*_{ij}s_{ij} - \sum_{k=1}^{K}(\beta^{*k}_i r^k_i + \gamma^{*k}_j c^k_j) \leq 0$, where $(\alpha^*_{ij}, \beta^{*k}_i, \gamma^{*k}_j)$ is a ray for the dual (Conforti, Cornuéjols, & Zambelli, 2014). The only way to potentially get an unbounded solution for the dual is by increasing α_{ij}, as the coefficients of the β^k_i, γ^k_j are non-positive in the objective and all variables need to be non-negative. From this, it follows, that when $s^*_{ij} = 0$, there will be no violated cut for this (i, j) (which of course is expected, as the cell is not chosen in the solution matrix in this case). Thus, suppose $s^*_{ij} = 1$. Now if we want to increase α_{ij}, to stay feasible, we also need to increase either β^k_i or γ^k_j for each k with the same amount, due to constraints (D-CONS). Hence, if for any k, both $r^{*k}_i = 1$ and $c^{*k}_j = 1$, we again do not get an unbounded solution, as the effect of the increased β^k_i/γ^k_j for this k cancels out the increase caused by α_{ij} in the objective function. From this, it follows that to obtain a ray, we need to have either $r^{*k}_i = 0$ or $c^{*k}_j = 0$ (or both zero) for each k. In the ray, α_{ij} and the β^k_i/γ^k_j with the zero coefficient in the objective will take the same positive value (in case $r^{*k}_i = 0$ and $c^{*k}_j = 0$, we can pick any of the two; we could also share the value of α_{ij} between them, but then, the obtained cut will be the corresponding combination of the two cuts where we picked one of the two), while all other dual variables have value zero. Normalizing $\alpha_{ij} = 1$, we obtain the following Benders feasiblity cuts

$$s_{ij} \leq \sum_{k\in S} r^k_i + \sum_{k\in\{1,\ldots,K\}\setminus S} c^k_j \quad \forall S \subseteq \{1, \ldots, K\}.$$

The resulting formulation is denoted with (B) and given below. Note that using similar objective-based arguments as for (ILP), we only need to pose the Benders cuts for (i, j) with $\mathcal{M}_{ij} > 0$ and constraints (SRC-LINK1) for (i, j) with $\mathcal{M}_{ij} < 0$. There is an exponential number of Benders feasibility cuts (SRC-LINK2), separation is discussed in Section 3.2.

$$\max \sum_{i\in R,\, j\in C} \mathcal{M}_{ij} s_{ij}$$

$$s_{ij} + 1 \geq r_i^k + c_j^k \quad \forall (i,j) \in R \times C : \mathcal{M}_{ij} < 0; k = 1, \ldots, K \tag{SRC-LINK1}$$

$$s_{ij} \leq \sum_{k\in S} r_i^k + \sum_{k\in\{1,\ldots,K\}\setminus S} c_j^k \quad \forall (i,j) \in R \times C : \mathcal{M}_{ij} > 0; \forall S \subseteq \{1,\ldots,K\} \tag{SRC-LINK2}$$

$$r_i^k \in \{0,1\} \quad \forall i \in R; k = 1, \ldots, K$$

$$c_j^k \in \{0,1\} \quad \forall j \in C; k = 1, \ldots, K$$

$$s_{ij} \in \{0,1\} \quad \forall i \in R, j \in C$$

3.1. Optimality cuts

The following optimality cuts can be derived based on the number of positive entries in a row/column and optimality arguments.

Theorem 4. *For a given row i, let $|i^+| = |\{\mathcal{M}_{ij} > 0 : j \in C\}|$ be the number of positive entries in row i. For a given column j, let $|j^+| = |\{\mathcal{M}_{ij} > 0 : i \in R\}|$ be the number of positive entries in column j. There always exists at least one optimal solution for MWSCP, where the following cardinality constraints are satisfied.*

$$\sum_{k=1}^{K} r_i^k \leq |i^+|, \quad \sum_{k=1}^{K} c_j^k \leq |j^+| \tag{NPOS}$$

Proof. Suppose $\sum_{k=1}^{K} r_i^k > |i^+|$. This means, in at least one submatrix, let us say k', where row i is selected, only cells (i,j) with $\mathcal{M}_{ij} \leq 0$ are selected. Thus, a solution with at least the same objective value can be obtained by not selecting row i in submatrix k'. The proof for the column-inequality is similar. □

Clearly, these inequalities are only useful, if $|i^+| < K$ (resp., $|j^+| < K$), but for a surprising number of the instances from literature there were some rows and columns satisfying this condition. We note that in case $|i^+| = 1$ (resp., $|j^+| = 1$), the inequalities can be disaggregated and strengthened to $r_i^k \leq c_j^k$ for the single entry (i,j) with $\mathcal{M}_{ij} > 0$.

Finally, we use row-based symmetry breaking constraints for (B), which are defined as follows.

$$r_i^{k+1} \leq \sum_{i' \leq i} r_{i'}^k \quad \forall i \in R; k = 1, \ldots, K-1 \tag{$B-SYM$}$$

3.2. Implementation details

We implemented a branch-and-cut algorithm, in which inequalities (SRC-LINK2) and also inequalities (SRC-LINK1) are separated. We initialize with (SRC-LINK2) for $S = \emptyset$ and $S = \{1,\ldots,K\}$, inequalities (NPOS) and the symmetry-breaking inequalities (B-SYM). Inequalities (SRC-LINK1) and the remaining inequalities (SRC-LINK2) are only separated for integer solutions $(\mathbf{r}^*, \mathbf{c}^*, \mathbf{s}^*)$. Their separation is done by enumeration. For each row, at most one violated inequality is added per separation call in order to avoid overloading the LP-relaxation. We also use inequalities (SUBMTX). They are separated for fractional $\mathbf{s}^*$ with Algorithm 1 using *maxCut*. Similar to the branch-and-cut for (ILP) described in Section 2.3, a feasible starting solution is provided using the heuristic described in Section 4.

4. A greedy randomized adaptive search procedure and a local search

In order to find good initial solutions for our branch-and-cut algorithms, we designed a *greedy randomized adaptive search* procedure (GRASP) (Feo & Resende, 1995; Resende & Ribeiro, 2003) for the MWSCP. We also apply a *local search* after the GRASP to try to improve the solutions found. The values chosen for the parameters in the heuristics are discussed in Section 5.2.

The GRASP is detailed in Algorithm 2 . It iteratively constructs

input : Instance $(\mathcal{M}, K)$, `acceptanceRate`
output: Solution $\{(\mathcal{R}_k, \mathcal{C}_k), k = 1,\ldots,K\}$ with objective value z^G

$\mathcal{M}'_{ij} \leftarrow \mathcal{M}_{ij}, i = 1,\ldots,m, j = 1,\ldots,n$
$z^G \leftarrow 0$
for $k = 1,\ldots,K$ **do**
  $\mathcal{R}_k = \emptyset$ `// rows of the currently constructed submatrix`
  $\text{colSum}_j \leftarrow 0, j = 1,\ldots,n$ `// column-sum for the currently considered rows`
  `found` $\leftarrow$ *false*
  `currentBestObjSubmatrix` $\leftarrow 0$
  do
    `found` $\leftarrow$ *false*
    $i^* \leftarrow$ *null* `// chosen row to add`
    $C(i^*) \leftarrow \emptyset$ `// best columns for the chose row to add`
    for $i \in R, i \notin \mathcal{R}_k$ **do**
      `rowValue` $\leftarrow 0$ `// contains the objective of the current submatrix when row` i `is included`
      $C(i) \leftarrow \emptyset$
      `// check for all columns if they should be included when selecting rows` $\mathcal{R}_k \cup \{i\}$
      for $j = 1,\ldots,n$ **do**
        $\text{colValueTesteRate}_j \leftarrow \text{colSum}_j + \mathcal{M}'_{ij}$
        `// colum-sum when including row` i
        if $\text{colValueTesteRate}_j > 0$ **then**
          $C(i) \leftarrow C(i) \cup \{j\}$
          `rowValue` $\leftarrow$ `rowValue` $+ \text{colValueTesteRate}_j$
      `// adding row` i `would improve the current submatrix`
      if *rowValue* $>$ *currentBestObjSubmatrix* **then**
        if *random*$(0,1) \geq$ *acceptanceRate* **then**
          `found` $\leftarrow$ *true*
          $C(i^*) \leftarrow C(i)$
          `currentBestObjSubmatrix` $\leftarrow$ `rowValue`
          $i^* \leftarrow i$
      `// Update the selected rows, columns and colSum`$_j$ `for the next iteration`
      if *found* **then**
        $\mathcal{C}_k \leftarrow C(i^*)$
        $\mathcal{R}_k \leftarrow \mathcal{R}_k \cup \{i^*\}$
        for $j \in 1,\ldots,n$ **do**
          $\text{colSum}_j \leftarrow \text{colSum}_j + \mathcal{M}'_{i^*j}$
  while *found*
  `// Update the objective value and` M'_{ij}
  for $i \in \mathcal{R}_k$ **do**
    for $j \in \mathcal{C}_k$ **do**
      if $\mathcal{M}'_{ij} \neq 0$ **then**
        $z^G \leftarrow z^G + \mathcal{M}'_{ij}$
        $\mathcal{M}'_{ij} \leftarrow 0$
return $\{(\mathcal{R}_k, \mathcal{C}_k), k = 1,\ldots,K\}$ and z^G

Algorithm 2: A GRASP for the MWSCP.

the submatrices $(\mathcal{R}_1, \mathcal{C}_1), \ldots (\mathcal{R}_K, \mathcal{C}_K)$. Each of these submatrices is created by greedily growing the set of selected row-indices $\mathcal{R}_k$. We check for each $i \notin \mathcal{R}_k$ the change in objective value caused by adding i to the rows $\mathcal{R}_k$. For a given set of rows $\mathcal{R}_k$, the columns

$\mathcal{C}_k$ to be included are determined as all columns j with positive column-sums $\sum_{i' \in \mathcal{R}_k} \mathcal{M}'_{i'j}$. In this calculation, $\mathcal{M}'_{ij}$ is either $\mathcal{M}_{ij}$ or zero, if the cell (i, j) was already selected in a previous submatrix. The calculation can be done efficiently by storing and updating for each column $j \in C$ its current column-sum for the currently selected $\mathcal{R}_k$ in a vector $colSum_j$.

As a greedy criteria, we would add the row i giving the largest improvement in objective function to $\mathcal{R}_k$. If there is no more row $i \notin \mathcal{R}_k$ giving an improvement in the objective value, we stop and move on to the next submatrix. To obtain a GRASP, we modify this greedy criteria, such that with 30% probability a row which would give an improvement is not added. Once a submatrix $(\mathcal{R}_k, \mathcal{C}_k)$ is completed, we set $\mathcal{M}'_{ij}$ of all the cells $(i, j) \in (\mathcal{R}_k, \mathcal{C}_k)$ to zero for the remaining iterations. This is done as by the definition of the problem, each cell only contributes once to the objective function.

The local search is described in Algorithm 3 . It takes a feasible solution $\left\{(\mathcal{R}_k, \mathcal{C}_k), k = 1, \ldots, K\right\}$ and tries to improve it by applying the following three operators in order, namely: (i) *row removal*, (ii) *row addition*, (iii) *row move*. The operators are explained in detail below. Whenever there is no more improvement possible with an operator, we move on to the next one. We use a *first improvement* strategy, i.e., whenever a move results in an improvement in the objective function, it is applied. The order of the rows is randomly shuffled before each iteration to consider a diverse set of rows in subsequent iterations. Moreover, when an improving move is found, we go back to the first operator for the next iteration. Finally, if the last operator also does not find any improving move anymore, we perform a random *row exchange* move to try to escape the current local maximum, and then go back to the first operator. This is done for *nPerturbations* times in each call of the local search.

In a *row removal* operation, we remove a row i from a given $\mathcal{R}_{k'}$ and check, if this brings an improvement in the objective function. The operation is denoted as $rowRemoval(\left\{(\mathcal{R}_k), k = 1, \ldots, K\right\}, k', i)$ in Algorithm 3. Note that a row-removal in the k'th submatrix can have an impact also on the (optimal) structure of all the other submatrices. In particular, recall that even for a fixed set of rows $\mathcal{R}_1, \ldots, \mathcal{R}_K$, finding the optimal columns is NP-hard (Derval et al., 2019a). Note that this is exactly the problem we are facing when we want to evaluate the impact of a row removal operation. To obtain an efficient local search, we thus solve this problem heuristically. This is done greedily as follows: For each column j, check for each given $\mathcal{R}_1, \ldots, \mathcal{R}_K$ the change in objective function if j is added to the respective submatrix. If there is one or more k' where the addition of column j to submatrix k' gives a positive improvement, add j to the submatrix k^* where the improvement is maximal (ties are broken by smallest index). Moreover, set all cell-entries $\mathcal{M}_{ij} : i \in \mathcal{R}_{k^*}$ to zero. Then go back to the start and check, if there are further submatrices k' where addition of j gives a positive improvement in the objective function under these modified cell-entries. This procedure is repeated, until there is no more submatrix k' where addition of column j gives a positive improvement in the objective function. Note that when doing the repeated checks for a given column j, we do not need to recompute the change in objective value from scratch, but can use *lazy evaluation* (Leskovec et al., 2007) as the change in objective function is non-increasing for each submatrix and iteration.

If an improved solution is obtained by applying the operator, the obtained solution is used to update the current solution. This is indicated by the function $rowRemovalSolution(\left\{(\mathcal{R}_k), k = 1, \ldots, K\right\}, k', i)$ in Algorithm 3. As for all the other operators used in our local search we also need to find a solution for a fixed set of rows, we use the same heuristic procedure as described above to solve the resulting problem. In a *row addition* operation, we add a row i to a given $\mathcal{R}_{k'}$. In a *row move* operation, we move a row i from a given $\mathcal{R}_{k'}$ to a given $\mathcal{R}_{k''}$ with $i' \notin \mathcal{R}_{k''}$ (i.e., row i is removed from $\mathcal{R}_{k'}$ and added to $\mathcal{R}_{k''}$). Finally, in a *row exchange* operation, we randomly pick two submatrices k' and k'' and two row-indices $i' \in \mathcal{R}_{k'}, i' \notin \mathcal{R}_{k''}$ and $i'' \in \mathcal{R}_{k''}, i'' \notin \mathcal{R}_{k'}$ and move i' from $\mathcal{R}_{k'}$ to $\mathcal{R}_{k''}$ and i'' from $\mathcal{R}_{k''}$ to $\mathcal{R}_{k'}$.

input : Instance $(\mathcal{M}, K)$ and solution $\left\{(\mathcal{R}_k, \mathcal{C}_k), k = 1, \ldots, K\right), \texttt{nPertubations}$
output: Solution $\left\{(\mathcal{R}_k, \mathcal{C}_k), k = 1, \ldots, K\right\}$

```
1  pertubations ← 0
2  do
3      improved ← false
4      for k' = 1, ..., K do
5          for i ∈ R_k' do
6              improved ← rowRemoval({(R_k, C_k), k = 1, ..., K}, k', i)
7              if improved then
8                  {(R_k, C_k), k = 1, ..., K} ← rowRemovalSolution({(R_k), k = 1, ..., K}, k', i)
9                  go to line 2
10     for k' = 1, ..., K do
11         for i ∈ R : i ∉ R_k' do
12             improved ← rowAddition({(R_k, C_k), k = 1, ..., K}, k', i)
13             if improved then
14                 {(R_k, C_k), k = 1, ..., K} ← rowAdditionSolution({(R_k), k = 1, ..., K}, k', i)
15                 go to line 2
16     for k' = 1, ..., K do
17         for i ∈ R_k' do
18             for k'' = 1, ..., K : k'' ≠ k', i ∉ R_k'' do
19                 improved ← rowMove({(R_k), k = 1, ..., K}, k', k'', i)
20                 if improved then
21                     {(R_k, C_k), k = 1, ..., K} ← rowMoveSolution({(R_k), k = 1, ..., K}, k', k'', i)
22                     go to line 2
       // perform a random rowExchange as the other operators were not successfull
23     k' ← random({1, ..., K})
24     i' ← random row from R_k'
25     k'' ← random({1, ..., K} \ {k'})
26     i'' ← random row from R_k''
27     if i'' ∈ R_k' ∨ i' ∈ R_k'' then
28         go to line 22
29     {(R_k, C_k), k = 1, ..., K} ← rowExchangeSolution({(R_k), k = 1, ..., K}, k', k'', i', i'')
30     pertubations ← pertubations + 1
31 while improved and pertubations < nPertubations
32 return {(R_k, C_k), k = 1, ..., K}
```

Algorithm 3: Scheme of the local search for the MWSCP.

5. Computational results

The algorithms were implemented in C++ and IBM ILOG CPLEX 12.9 with default settings was used as branch-and-cut framework. The runs were made on a single core of an Intel Xeon E5-2670v2 machine with 2.5 GHz and 6 GB of RAM. An executable of

Table 1
Effect of different parameter values for (*acceptanceRate*, *nPerturbations*) on the heuristic performance .

(30,5)			(30,3)			(30,7)			(20,5)			(40,5)		
z^H	#B	t[s]	z^H	#B	t[s]	z^H	#B	t[s]	z^H	#B	t[s]	z^H	#B	t[s]
13219.50	106	412.86	13192.19	92	344.83	13259.71	118	420.89	13204.93	95	381.44	13237.10	105	379.89

Table 2
Effect of different parameter values for (*maxCut*, *nCutLoopiterations*) on the branch-and-cut .

	(*ILP*) − (100, 200)			(*ILP*) − (10, 5)			(*B*) − (100, 200)			(*B*) − (10, 5)		
UB_t	UB_r	UB	#BB	UB_r	UB	#BB	UB_r	UB	#BB	UB_r	UB	#BB
18004.03	17854.76	17854.76	0.33	17916.67	17916.65	1.51	17720.81	17720.81	15.08	17863.73	17863.64	50.46

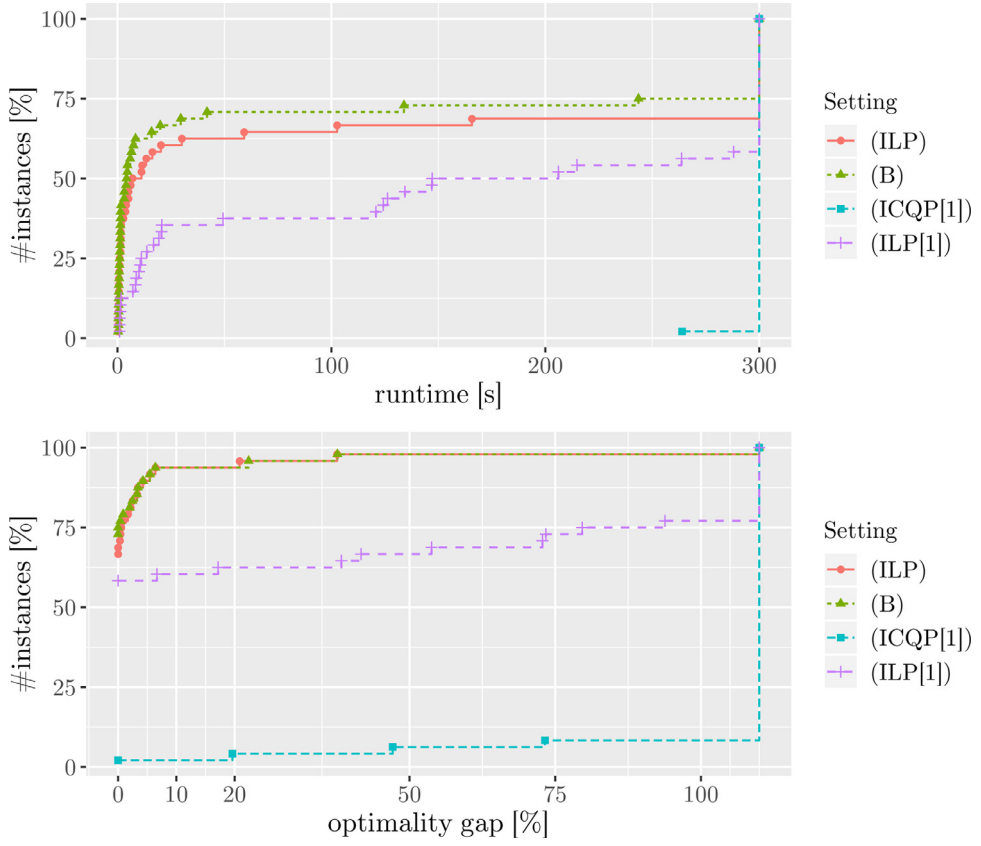

Fig. 2. Runtime and optimality-gap for dataset `small`. For better readability, the optimality-gap is truncated at 100%.

our algorithms is made available at https://msinnl.github.io/pages/instancescodes.html for other researchers.

5.1. Instances

We tested our approaches on the instances also used in Derval et al. (2019a). These include *synthetic instances*, which are available at Derval, Branders, Dupont, & Schaus (2018) and *real-world instances*, which we obtained by request from the authors of Derval et al. (2019a). The *synthetic instances* were constructed in such a way that at least a near-optimal solution is known by construction: To create an $m \times n$-instance, K submatrices (of a given dimension $r \times s$) were randomly created (Gaussian with variance $\frac{r \text{ or } s}{20}$) and then placed into an "empty" $m \times n$-matrix. For placing the submatrices in the instance, a parameter o was used which gives the minimum overlap (in % of cells) between submatrices the placement needs to have. After placing the submatrices in the $m \times n$-matrix, noise with given variance ρ and mean zero is added to every entry of the $m \times n$-matrix. With this procedure, the K submatrices give the optimal (or a near-optimal, due to the added noise) solution. Using this procedure `small` and `large` instances were created. The `small` instances are available at Derval et al. (2018) in subfolders with names `k_noise`, `k_overlap`, `size_noise`, `size_overlap`. The names give the parameters which were changed for creating the corresponding instances in the folder, while the other parameters were kept fixed. In each folder there are twelve instances. We note that in different subfolders, some instances with the same name (which is defined by the parameter combination for creation of the instance), but different matrix-entries are included. The instances have $(m, n) \in \{(50, 50), (100, 100), (200, 200), (400, 400)\}$, for the values of the other parameters we refer to Table 3.

For creating the set of `large` instances, the following parameters were used: $(m, n) \in \{(800, 200), (640, 250), (400, 400)\}$,

Table 3
Detailed results for the `small` synthetic instances from folders `k_overlap`, `k_noise`, `size_overlap` and `size_noise`. In $t[s]$, "TL" indicates timelimit of 300 seconds reached and "ML" indicates memorylimit reached. The entry "*" indicates that for this instance CPLEX stopped due to the timelimit before managing to solve the initial LP-relaxation.

Instance					(*ILP*)			(*B*)			($ILP_{\text{Derval et al. (2019a)}}$)			*H*10		*H*100	
k	*m*	o	ρ	*r*	t[s]	*g*[%]	*pg*[%]	t[s]	*g*[%]	*pg*[%]	t[s]	*g*[%]	*pg*[%]	t[s]	*pg*[%]	t[s]	*pg*[%]
2	100	0.0	0.0	25	14	**0.00**	**0.00**	2	**0.00**	**0.00**	9	**0.00**	**0.00**	1	5.02	4	5.02
2	100	0.2	0.0	25	2	**0.00**	**0.00**	2	**0.00**	**0.00**	8	**0.00**	**0.00**	2	**0.00**	7	**0.00**
2	100	0.4	0.0	25	2	**0.00**	**0.00**	2	**0.00**	**0.00**	9	**0.00**	**0.00**	2	**0.00**	5	**0.00**
2	100	0.6	0.0	25	4	**0.00**	**0.00**	2	**0.00**	**0.00**	10	**0.00**	**0.00**	1	3.17	2	1.30
4	100	0.0	0.0	25	TL	**6.39**	**0.00**	TL	**6.39**	**0.00**	TL	**6.39**	**0.00**	5	**0.00**	39	**0.00**
4	100	0.2	0.0	25	7	**0.00**	**0.00**	7	**0.00**	**0.00**	21	**0.00**	**0.00**	7	**0.00**	55	**0.00**
4	100	0.4	0.0	25	TL	**2.46**	**0.00**	TL	**2.46**	**0.00**	TL	**2.46**	**0.00**	2	**0.00**	19	**0.00**
4	100	0.6	0.0	25	TL	1.68	1.68	4	**0.00**	**0.00**	252	**0.00**	**0.00**	2	1.68	12	1.68
6	100	0.0	0.0	25	17	**0.00**	**0.00**	17	**0.00**	**0.00**	17	**0.00**	**0.00**	17	**0.00**	159	**0.00**
6	100	0.2	0.0	25	21	**0.00**	**0.00**	21	**0.00**	**0.00**	39	**0.00**	**0.00**	21	**0.00**	214	**0.00**
6	100	0.4	0.0	25	31	**0.00**	**0.00**	31	**0.00**	**0.00**	51	**0.00**	**0.00**	30	**0.00**	269	**0.00**
6	100	0.6	0.0	25	TL	**0.88**	0.31	TL	**0.88**	0.31	TL	**0.88**	0.31	15	0.31	125	**0.00**
2	50	0.0	0.0	16	1	**0.00**	**0.00**	1	**0.00**	**0.00**	2	**0.00**	**0.00**	1	**0.00**	2	**0.00**
2	50	0.0	0.5	16	2	**0.00**	**0.00**	2	**0.00**	**0.00**	7	**0.00**	**0.00**	2	**0.00**	3	**0.00**
2	50	0.0	1.0	16	TL	0.31	**0.00**	43	**0.00**	**0.00**	153	**0.00**	**0.00**	1	**0.00**	3	**0.00**
3	50	0.0	0.0	16	2	**0.00**	**0.00**	2	**0.00**	**0.00**	2	**0.00**	**0.00**	2	**0.00**	6	**0.00**
3	50	0.0	0.5	16	3	**0.00**	**0.00**	2	**0.00**	**0.00**	36	**0.00**	**0.00**	2	**0.00**	9	**0.00**
3	50	0.0	1.0	16	TL	0.40	**0.00**	TL	**0.39**	**0.00**	TL	10.84	**0.00**	2	**0.00**	10	**0.00**
4	50	0.0	0.0	16	2	**0.00**	**0.00**	2	**0.00**	**0.00**	3	**0.00**	**0.00**	2	**0.00**	14	**0.00**
4	50	0.0	0.5	16	TL	**4.27**	4.27	TL	**4.27**	4.27	TL	4.49	3.94	3	4.27	14	**0.00**
4	50	0.0	1.0	16	TL	2.15	**0.00**	TL	**2.00**	**0.00**	TL	11.04	**0.00**	3	**0.00**	19	**0.00**
5	50	0.0	0.0	16	TL	**5.45**	5.45	TL	**5.45**	5.45	TL	**5.45**	5.45	5	5.45	36	**0.00**
5	50	0.0	0.5	16	8	**0.00**	**0.00**	5	**0.00**	**0.00**	TL	0.31	**0.00**	5	**0.00**	37	**0.00**
5	50	0.0	1.0	16	TL	**3.25**	**0.00**	TL	3.25	**0.00**	TL	8.84	**0.00**	6	**0.00**	51	**0.00**
2	50	0.0	0.0	16	1	**0.00**	**0.00**	2	**0.00**	**0.00**	2	**0.00**	**0.00**	1	**0.00**	2	**0.00**
2	50	0.1	0.0	16	1	**0.00**	**0.00**	1	**0.00**	**0.00**	2	**0.00**	**0.00**	1	**0.00**	2	**0.00**
2	50	0.2	0.0	16	1	**0.00**	**0.00**	1	**0.00**	**0.00**	2	**0.00**	**0.00**	1	**0.00**	3	**0.00**
2	50	0.4	0.0	16	2	**0.00**	**0.00**	1	**0.00**	**0.00**	2	**0.00**	**0.00**	1	3.15	2	**0.00**
2	100	0.0	0.0	33	2	**0.00**	**0.00**	2	**0.00**	**0.00**	10	**0.00**	**0.00**	2	**0.00**	16	**0.00**
2	100	0.1	0.0	33	2	**0.00**	**0.00**	2	**0.00**	**0.00**	6	**0.00**	**0.00**	2	**0.00**	8	**0.00**
2	100	0.2	0.0	33	2	**0.00**	**0.00**	2	**0.00**	**0.00**	7	**0.00**	**0.00**	2	**0.00**	14	**0.00**
2	100	0.4	0.0	33	13	**0.00**	**0.00**	3	**0.00**	**0.00**	12	**0.00**	**0.00**	2	4.89	8	4.89
2	200	0.0	0.0	50	6	**0.00**	**0.00**	6	**0.00**	**0.00**	95	**0.00**	**0.00**	6	**0.00**	52	**0.00**
2	200	0.1	0.0	50	5	**0.00**	**0.00**	4	**0.00**	**0.00**	85	**0.00**	**0.00**	4	**0.00**	28	**0.00**
2	200	0.2	0.0	50	167	**0.00**	**0.00**	8	**0.00**	**0.00**	109	**0.00**	**0.00**	2	11.35	12	11.35
2	200	0.4	0.0	50	104	**0.00**	**0.00**	8	**0.00**	**0.00**	125	**0.00**	**0.00**	3	4.94	9	4.94
2	50	0.0	0.0	16	1	**0.00**	**0.00**	2	**0.00**	**0.00**	2	**0.00**	**0.00**	1	**0.00**	2	**0.00**
2	50	0.0	0.5	16	2	**0.00**	**0.00**	2	**0.00**	**0.00**	5	**0.00**	**0.00**	1	**0.00**	3	**0.00**
2	50	0.0	1.0	16	TL	0.55	**0.00**	245	**0.00**	**0.00**	261	**0.00**	**0.00**	1	**0.00**	3	**0.00**
2	100	0.0	0.0	33	2	**0.00**	**0.00**	2	**0.00**	**0.00**	9	**0.00**	**0.00**	2	**0.00**	16	**0.00**
2	100	0.0	0.5	33	5	**0.00**	**0.00**	5	**0.00**	**0.00**	85	**0.00**	**0.00**	3	**0.00**	17	**0.00**
2	100	0.0	1.0	33	TL	20.86	**0.00**	TL	22.40	**0.00**	TL	**18.59**	**0.00**	3	**0.00**	18	**0.00**
2	200	0.0	0.0	50	6	**0.00**	**0.00**	6	**0.00**	**0.00**	92	**0.00**	**0.00**	5	**0.00**	52	**0.00**
2	200	0.0	0.5	50	60	**0.00**	**0.00**	135	**0.00**	**0.00**	TL	1.92	**0.00**	13	**0.00**	119	**0.00**
2	200	0.0	1.0	50	TL	**37.58**	**0.00**	TL	37.63	**0.00**	TL	44.18	**0.00**	14	**0.00**	136	**0.00**
2	400	0.0	0.0	50	12	**0.00**	**0.00**	9	**0.00**	**0.00**	TL*	–	–	10	**0.00**	108	**0.00**
2	400	0.0	0.5	50	TL	**3.40**	**0.00**	TL	3.42	**0.00**	ML	–	–	34	**0.00**	TL	**0.00**
2	400	0.0	1.0	50	TL	182.43	**0.00**	TL	**182.38**	**0.00**	TL	189.00	**0.00**	34	**0.00**	TL	**0.00**

$(r, s) \in \{(35, 70), (50, 50)\}$, $K \in \{2, 4, 8\}$, $o \in \{0\%, 30\%, 60\%\}$, $\rho \in \{0, 0.5, 1\}$.For each parameter combination, ten instances were created for a total of 1617 `large` instances.[1] Following Derval et al. (2019a), instances created with a given K were only tested for this K in our computational study.

The real-world instances were based on *migration data* (Dao, Docquier, Maurel, & Schaus, 2018), *gene expression data* (de Souto, Costa, de Araujo, Ludermir, & Schliep, 2008) and results of the *Olympic games* (IOC Research & Reference Service, 2021), which are three application-areas for the MWSCP described in Derval et al. (2019a). For the *migration data* and *Olympic games data*, three instances were obtained for each of the sets, for the *gene expression data*, each dataset corresponds to an instance. These instances were studied for $K = 4$ in Derval et al. (2019a).

[1] for three parameters combinations out of the 162 combinations, only nine instances exist, and not ten.

5.2. Analyzing the influence of the algorithm-parameters

In this section, we use a subset of the `large` instances to analyze the influence of different values for the parameters of our algorithms. To create this set, we took one of the ten instances for each parameter-combination used for creating the instances. Thus, the test-set contains 162 instances.

We first consider the parameters of the heuristics. In Table 1 we give the solution value (z^H), runtime ($t[s]$), and number of best solutions (#*B*) found for different pairs of (*acceptanceRate*, *nPerturbations*).

The table shows that all values of *acceptanceRate* perform quite similar, and a larger value for *nPerturbations* slightly increases the solution quality, while it also increases the runtime.

In Table 2 we consider the parameters (*maxCut*, *nCutLoopiterations*) of the branch-and-cut and give results for two different settings, namely (10,5) and (100,200). In the table, we report the value of the trivial upper bound which would be ob-

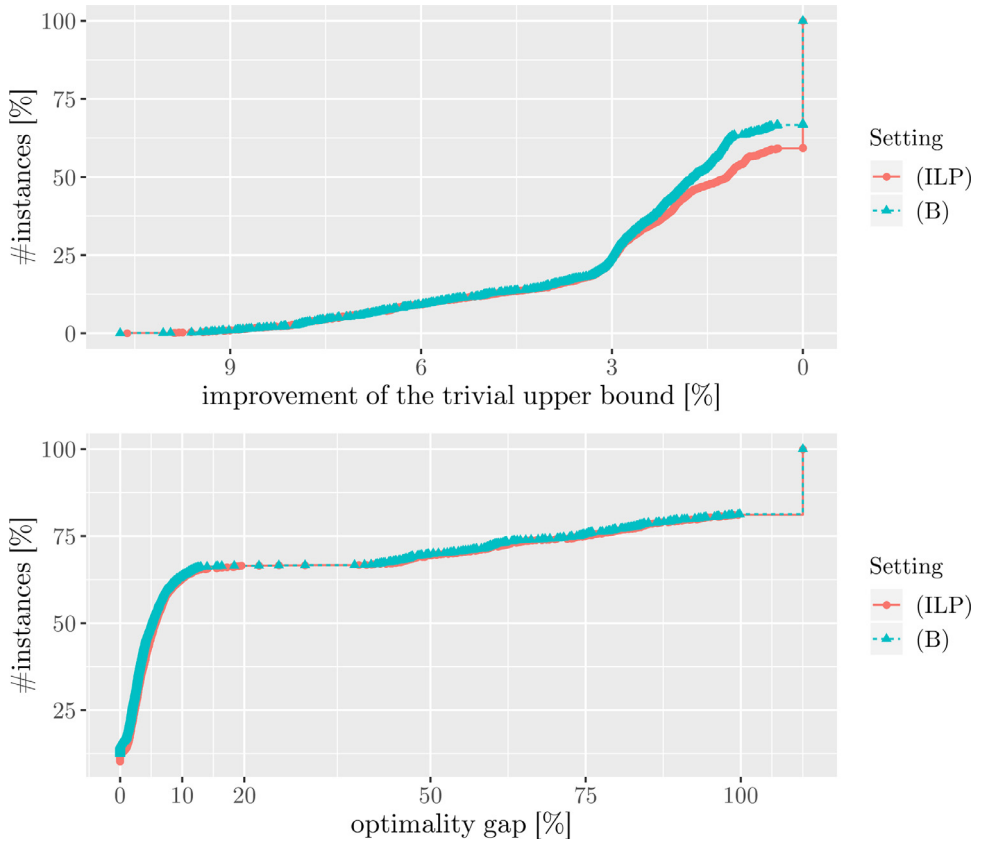

Fig. 3. Improvement to the trivial upper bound and optimality-gap for dataset `large`. For better readability, the optimality-gap is truncated at 100%.

tained with e.g., ($ILP_{\text{Derval et al. (2019a)}}$) and our models without the valid inequalities (i.e., the sum of all positive entries) in column *UB*, the obtained root upper bounds in columns UB_r, the obtained upper bounds after the given timelimit of 20 minutes in columns *UB* and the number of branch-and-bound nodes in column #*BB*. From the table, we see that the valid inequalities have an effect on the bound (comparing (10,5) and (100,200)), while branching appears to be very ineffective in improving the bound (comparing UB_r and *UB*). We also see that (*ILP*) struggles even with the setting (10,5) to leave the root-node of the branch-and-cut tree within the given timelimit of 20 minutes (column #*BB*), while (*B*) seems to be faster. This can be explained with the smaller size of the formulation (*B*) which allows for faster solution of the LP-relaxation.

We used $(acceptanceRate, nPerturbations) = (30, 5)$ and $(maxCut, nCutLoopiterations) = (100, 200)$ for the results reported in the following sections.

5.3. Results for the synthetic instances

Next, we are interested in the results for the synthetic instances and start with the `small`-dataset. These instances were only used for testing the ILP and ICQP approaches in Derval et al. (2019a). Recall that in Derval et al. (2019a), all approaches were only tested with regard to their heuristic performance (i.e., quality of the primal solution the approach obtained after a given timelimit compared against the "optimal" solution known by construction). We note that in Derval et al. (2019a), results were only presented for the instances from folders `k_noise` and `size_noise`. Thus, in order to allow for a meaningful comparison with the ILP and ICQP of Derval et al. (2019a) also including the quality of the obtained upper bounds, we reimplemented both the ILP and ICQP. The ILP of Derval et al. (2019a) is already stated in Section 2 (formulation ($ILP_{\text{Derval et al. (2019a)}}$)). The ICQP proposed in Derval et al. (2019a) is denoted as ($ICQP_{\text{Derval et al. (2019a)}}$) and restated below.

$$\max \sum_{i\in R, j\in C} \mathcal{M}_{ij} s_{ij}$$

$$Ks_{i,j} \geq \sum_{k=1,\dots,K} r_i^k c_j^k \quad \forall i \in R, j \in C$$

$$s_{ij} \leq \sum_{k=1,\dots,K} r_i^k c_j^k \quad \forall i \in R, j \in C$$

$$r_i^k \in \{0,1\} \quad \forall i \in R; k = 1,\dots,K$$

$$c_j^k \in \{0,1\} \quad \forall j \in C; k = 1,\dots,K$$

$$s_{ij} \in \{0,1\} \quad \forall i \in R, j \in C$$

The timelimit for these runs was 300 seconds, as in Derval et al. (2019a). Within (*ILP*) and (*B*), the heuristic described in Section 4 is run with ten restarts to generate an initial starting solution. Fig. 2 gives a plot of the runtime and optimality-gap for the instances from the `small`-dataset. The plots give the percentage of instances solved within a given time, resp., the percentage of instances within a given optimality-gap a the timelimit. The optimality-gap is computed as $(z^{UB} - z^*)/z^* \cdot 100$, where z^{UB} is the best upper bound found by the respective approach, and z^* the value of the best solution found by the respective approach.

In the figures, we see that our new approaches based on (*ILP*) and (*B*) outperform the previous approaches. The approach (*B*) works slighlty better than (*ILP*) and manages to solve 75% of the instances to optimality within the timelimit, and nearly all of these solved instances are solved within 50 seconds. Looking at the optimality-gap, we see that both (*ILP*) and (*B*) obtain an

Table 4
Aggregated results for the large synthetic instances.

Category	#*nI*	(*ILP*)					(*B*)					*H*10				Derval et al. (2019a)
		g[%]	*pg*[%]	*bi*[%]	>Derval et al. (2019a)	≥ Derval et al. (2019a)	*g*[%]	*pg*[%]	*bi*[%]	>Derval et al. (2019a)	≥ Derval et al. (2019a)	*t*[*s*]	*pg*[%]	>Derval et al. (2019a)	≥ Derval et al. (2019a)	*pg*[%]
$m = 400, n = 400$	540	46.61	1.53	1.81	215	337	45.69	1.13	2.11	222	364	429	1.15	245	354	2.50
$m = 640, n = 250$	540	47.05	1.33	1.95	227	359	46.53	1.08	2.11	230	373	389	1.07	247	360	2.89
$m = 800, n = 200$	537	47.24	1.82	2.09	232	347	46.88	1.55	2.14	235	359	361	1.65	245	333	3.25
$K = 2$	540	74.62	0.83	3.39	185	467	74.07	0.32	3.39	186	502	25	1.26	184	403	2.82
$K = 4$	540	42.44	1.80	1.92	240	324	42.17	1.66	1.96	241	331	149	1.71	242	328	3.87
$K = 8$	537	23.71	2.05	0.53	249	252	22.73	1.78	1.01	260	263	1008	0.90	311	316	1.93
$o = 0.0$	540	38.20	2.22	1.68	138	340	37.60	1.77	1.76	147	359	289	1.69	159	363	1.77
$o = 0.3$	537	46.10	1.41	1.91	245	345	45.42	1.06	2.13	246	370	492	1.26	266	350	4.47
$o = 0.6$	540	56.58	1.05	2.25	291	358	56.06	0.92	2.48	294	367	399	0.92	312	334	2.41
$\rho = 0.0$	539	2.99	2.71	0.00	44	227	2.34	2.05	0.00	47	268	234	2.86	55	175	0.15
$\rho = 0.5$	539	4.84	1.26	3.98	251	350	4.46	1.11	4.18	255	354	451	0.81	275	375	2.79
$\rho = 1.0$	539	133.06	0.70	1.87	379	466	132.30	0.60	2.18	385	474	493	0.19	407	497	5.70
$r = 35, s = 70$	807	46.46	1.50	1.93	302	486	45.87	1.21	2.11	310	515	395	1.14	334	500	2.48
$r = 50, s = 50$	810	47.46	1.62	1.96	372	557	46.85	1.30	2.14	377	581	391	1.44	403	547	3.27

Table 5
Detailed results for the real-world instances; "TL" indicates timelimit of 3 hours reached.

Name	m	n	(*ILP*)				(*B*)				*H*10		*H*100		Derval et al. (2019a)
			t[s]	*g*[%]	*pg*[%]	*#BB*	t[s]	*g*[%]	*pg*[%]	*#BB*	t[s]	*pg*[%]	t[s]	*pg*[%]	*pg*[%]
alizadeh-2000-v1095	42	1095	TL	66.12	**0.000**	0	TL	**64.96**	**0.000**	0	3	**0.000**	33	**0.000**	**0.000**
armstrong-2002-v1095	72	1081	TL	50.88	**0.000**	0	TL	**49.79**	**0.000**	0	30	**0.000**	310	**0.000**	**0.000**
bhattacharjee-2001095	203	1543	TL	22.02	0.004	0	TL	**21.97**	0.004	0	695	**0.000**	6502	**0.000**	0.623
bittner-2000095	38	2201	TL	144.37	**0.000**	0	TL	**136.16**	**0.000**	0	4	**0.000**	41	**0.000**	0.248
bredel-2005095	50	1739	TL	185.35	**0.000**	0	TL	**170.32**	**0.000**	0	7	**0.000**	73	**0.000**	**0.000**
chen-2002095	85	179	TL	18.40	**0.000**	0	TL	**18.35**	**0.000**	1153	2	**0.000**	22	**0.000**	8.891
chowdary-2006095	104	182	TL	3.93	0.077	0	TL	**3.77**	0.077	689	5	0.077	53	**0.000**	**0.000**
dyrskjot-2003095	40	1203	TL	**18.29**	**0.000**	0	TL	30.99	**0.000**	55	7	**0.000**	79	**0.000**	6.792
garber-2001095	66	4553	TL	**279.34**	**0.000**	0	TL	330.43	**0.000**	0	22	**0.000**	225	**0.000**	1.827
golub-1999-v1095	72	1868	TL	30.69	0.846	0	TL	**30.52**	0.846	0	55	0.846	559	**0.000**	0.659
immigration0.001	32	69	TL	1.75	**0.000**	622	TL	**1.51**	**0.000**	3920	1	**0.000**	6	**0.000**	**0.000**
immigration0.003	32	69	3007	**0.00**	**0.000**	1491	86	**0.00**	**0.000**	1545	0	0.064	2	0.064	0.064
immigration0.005	32	69	1	**0.00**	**0.000**	0	0	**0.00**	**0.000**	6	0	**0.000**	1	**0.000**	**0.000**
olympic0.01	75	149	TL	**13.74**	0.547	0	TL	19.82	0.547	135	7	0.547	79	**0.000**	0.236
olympic0.02	75	149	TL	20.88	0.227	0	TL	**19.81**	0.227	34	7	0.227	68	**0.000**	0.388
olympic0.04	75	149	TL	33.09	2.409	0	TL	**32.63**	2.409	290	3	2.409	27	1.472	**0.000**
olympic0.06	75	149	TL	31.28	0.312	0	TL	**30.82**	0.312	812	2	0.312	19	**0.000**	0.312

optimality-gap less than 10% for more than 90% of the instances within the timelimit.

Detailed results are given in Table 3. In this table, aside from results for (*ILP*), (*B*) we also provide results for the heuristic of Section 4 obtained with ten and 100 restarts (columns *H*10 and *H*100). Moreover, instead of ($ILP_{\text{Derval et al. (2019a)}}$) as shown in the figures, we present result for ($ILP'_{\text{Derval et al. (2019a)}}$) together with initialization by the heuristics (similar as done for(*ILP*), (*B*)) to get some impression of the effect of the heuristics when they are used with the (slightly improved) model from literature. For better readability, results for ($ICQP_{\text{Derval et al. (2019a)}}$) are not given, as in Fig. 2 the poor performance of this approach is already quite clear. To compare the solution quality of the primal solution obtained by the heuristics, we define the primal gap $pg[\%] = (z^B - z^*)/z^* \cdot 100$, where z^B is the best solution value found by any approach, and z^* is the best solution value found by the respective approach in the table-column.

In Table 3 we see that for the instances without noise (i.e., from folders `k_overlap` and `size_overlap`) our approaches (*ILP*) and (*B*) find the optimal solution for nearly all the instances. Our heuristics also perform quite well, and in some cases using more restarts (i.e., *H*100) improves the solution quality. For larger instances, i.e., instances from the folder `size_noise` with $m = 400$ and $\rho = 0.5, 1.0$ the heuristic setting *H*100 reaches the timelimit. In general, the instances with noise (i.e., from folders `k_noise` and `size_noise`) are harder to solve. The results for ($ILP'_{\text{Derval et al. (2019a)}}$) with initialization heuristics improve compared to the results observed by Derval et al. (2019a) for ($ILP_{\text{Derval et al. (2019a)}}$), however (*B*) performs better with respect to runtime and manages to solves two more instances to optimality within the given timelimit.

Results for set `large` For these runs, we considered (*ILP*), (*B*) and *H*10, i.e., the heuristic with ten restarts. The runs had a timelimit of 20 minutes (same as Derval et al., 2019a). For these instances, it was not possible to use ($ILP'_{\text{Derval et al. (2019a)}}$) due to the size of the instances. Within (*ILP*) and (*B*), the heuristic is run with at most ten restarts and a timelimit of 200 seconds to generate an initial starting solution. In Fig. 3 we give a plot of the bound improvement compared to the trivial upper bound and a plot of the optimality-gap. The bound improvement is computed as $bi[\%] = (z^{TB} - z^{UB})/z^{TB} \cdot 100$ where $z^{TB} = \sum_{i,j:\mathcal{M}_{ij}>0} \mathcal{M}_{ij}$, which is the trivial bound consisting of matrix-entries with positive value (i.e., this is the bound obtained by all models without valid inequalities, see Theorem 1). In the bound improvement plot, we report the percentage of instances with a *bi* of at least a given percentange. In the figures, we see that the performance of (*ILP*) and (*B*) is quite similar. The proposed valid inequalities are successful in improving the bound. Note that due to the structure of the instances, for some instances, the optimal solution value is equal to the trivial bound. For around 62.5% of instances, the optimality-gap at the timelimit is within 10%, for the remaining instances there is a steep increase in the gap.

For these instances, results from the heuristics proposed in Derval et al. (2019a) are available online at (Derval, Branders, Dupont, & Schaus, 2019b). We calculated *pg*[%] including the best results from Derval et al. (2019a) and include it for comparison. The results are given in Table 4 and aggregated by instance-parameters in the same way as in Derval et al. (2019a). Column *#nI* gives the number of instances for each aggregation. We do not explicitly mention the runtime in the table for (*ILP*) and (*B*), as the timelimit was reached in most of the runs (only some of the instances with $\rho = 0.0$ could be solved to optimality). In column *bi*[%] we report the obtained bound improvement compared to the trivial bound. Moreover, the table also includes columns # $>$(Derval et al., 2019a) and # $\geq$(Derval et al., 2019a) which give the number of instances, where our approaches found a solution with improved, resp., at least similar solution value compared to the best solution value from Derval et al. (2019a).

The results show that noise, i.e., $\rho = 1.0$ seems to be the main parameter influencing the difficulty of the instances. While for instances with $\rho = 0.0$ and $\rho = 0.5$, the aggregated optimality-gaps are under 5% for both (*ILP*) and (*B*), for $\rho = 1.0$ they are around 133%. The parameter with the second largest impact seems to be the number of submatrices K. While for $K = 2$, the aggregated optimality-gaps are around 74%, for $K = 8$ they are around 23%. An explanation for this could be, that for larger K the optimal solution can take a larger share of the cells with positive entries, and thus the solution value will be nearer the trivial upper bound of Theorem 1. The overlap o also seems to have some effect on the difficulty of an instance, but there is only about 18% difference between the aggregated gaps for $o = 0.0$ and $o = 0.6$. The shape of the underlying matrix (i.e., m, n) and the shape of the embedded submatrices (i.e., r, s) does not seem to influence the difficulty of an instance.

Regarding the effectiveness of our approaches to find good primal solutions, we note that for all aggregation-categories except $\rho = 0.0$, settings (*B*) and *H*10 have an improved aggregated primal

gap compared to the primal gap of the best solutions from Derval et al. (2019a). Looking at the number of improved solution values, we see that for example for 407 out of 539 instances with $\rho = 1.0$ our heuristic $H10$ found an improved solution value compared to Derval et al. (2019a). With respect to other aggregation categories, we note that when K and o is larger, the heuristic $H10$ finds improved solutions more often compared to smaller value for these parameters. In total, $H10$ finds improved solution values compared to Derval et al. (2019a) for 737 out of 1627 instances, i.e., for nearly half the number of instances.

5.4. Results for the real-world instances

The results for the real-world instances are given Table 5. For these instances, results from the heuristics proposed in Derval et al. (2019a) are available online at (Derval et al., 2019b). Thus, similar to Table 4 for the `large` synthetic instances, we include *pg*[%] calculated with the best solution from Derval et al. (2019a). We also included the number of branch-and-bound nodes (column *#BB*) for our approaches. The timelimit for these runs was set to three hours.

In the table, we can see that instances `immigration0.003` and `immigration0.005` were solved to optimality. Moreover, for ten of these 17 instances, our approaches improve on the heuristic results of Derval et al. (2019a), and in all the other instances except `olympic0.04` our heuristic setting $H100$ at least matches the performance of the heuristic result of Derval et al. (2019a). For all instances except `bhattacharjee-2001095` these results are obtained very quickly.

We note that for most of the instances, our branch-and-cut approaches are still in the root-node when the timelimit occurs. However, recall that our root-cut-loop has a stopping criterion of "at-most five consecutive iterations without improvement". Thus, there was still improvement in the upper bound at the root-node when the timelimit occurred. In preliminary computations we also tried variants, where we branched earlier, but this was less effective in improving the upper bound (likely due to the symmetries inherent to the problem, which potentially make branching very ineffective).

6. Conclusion

In this paper, we developed new ILP approaches for the recently introduced *maximum weight submatrix coverage problem (MWSCP)*, including a branch-and-cut based on Benders decomposition. The MWSCPis concerned with selecting K submatrices of a given numerical matrix such that the sum of the matrix-entries, which occur in at least one of the selected submatrices, is maximized. The developed solution frameworks also include valid inequalities and preprocessing procedures. Moreover, we also presented a greedy randomized adaptive search procedure and a local search for the problem. We also show that the LP-relaxation of the previous ILP from Derval et al. (2019a) only achieves a trivial bound.

In a computational study on instances from literature, we investigated the performance of our algorithms in comparison with previous approaches from literature (Derval et al., 2019a) for the problem. The study showed that our new approaches outperform the previous approaches. We were able to find the proven optimal solution for two previously unsolved real-world instances. Moreover, for ten out of 17 real-world instances and over 700 of 1617 large-scale synthetic instances our methods managed to find an improved solution compared to the best known solution from Derval et al. (2019a).

There are several avenues for further work, as due to several factors the MWSCP seems to be a very challenging problem. For example, a "natural formulation" of the problem shares similarities with the boolean quadric polytope, and problems with such structure often have very weak LP-relaxations. The challenge is further compounded by symmetries, as the problem asks for K submatrices. On the other hand, each cell should only be counted at most once in the objective which seems to make (further) decomposition quite hard. Moreover, even for fixed rows in the K submatrices, the remaining problem remains NP-hard (Derval et al., 2019a). Finally, the instances from literature for the problem are quite large.

To tackle these issues, a column generation/branch-and-price approach, where submatrices are combined could be an interesting solution approach. However, due to the "shared objective function", such an approach may still have to use variables for each matrix-cell in the formulation in order to model the objective function. From a heuristic perspective, the design of new data structures for effective neighborhood search could be fruitful, as for larger instance, even evaluating simple heuristic moves can be computationally costly, as a move within one submatrix may impact all submatrices.

Acknowledgments

The research was supported by the Linz Institute of Technology (Project LIT-2019-7-YOU-211) and the JKU Business School.

References

Billionnet, A. (2005). Different formulations for solving the heaviest k-subgraph problem. *INFOR: Information Systems and Operational Research, 43*(3), 171–186.
Branders, V., Schaus, P., & Dupont, P. (2017). Combinatorial optimization algorithms to mine a sub-matrix of maximal sum. In *International workshop on new frontiers in mining complex patterns* (pp. 65–79). Springer.
Busygin, S., Prokopyev, O., & Pardalos, P. M. (2008). Biclustering in data mining. *Computers and Operations Research, 35*(9), 2964–2987.
Conforti, M., Cornuéjols, G., & Zambelli, G. (2014). *Integer programming*: 271. Springer.
Dao, T., Docquier, F., Maurel, M., & Schaus, P. (2018). Global migration in the 20th and 21st centuries: The unstoppable force of demography. Working Papers.
Derval, G., Branders, V., Dupont, P., & Schaus, P. (2018). Synthetic dataset used in "the maximum weighted submatrix coverage problem: A CP approach". Nov.10.5281/zenodo.3549866
Derval, G., Branders, V., Dupont, P., & Schaus, P. (2019a). The maximum weighted submatrix coverage problem: A CP approach. In *International conference on integration of constraint programming, artificial intelligence, and operations research* (pp. 258–274). Springer.
Derval, G., Branders, V., Dupont, P., & Schaus, P. (2019b). "The maximum weighted submatrix coverage problem: A CP approach": Experiment results. Nov.10.5281/zenodo.3549851
Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*(2), 109–133.
IOC Research and Reference Service The guardian: Olympic sports and medals, 1896–2014. URL https://www.kaggle.com/the-guardian/olympic-games.
Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., & Glance, N. (2007). Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 420–429).
Letchford, A. N., & Sørensen, M. M. (2014). A new separation algorithm for the boolean quadric and cut polytopes. *Discrete Optimization, 14*, 61–71.
Madeira, S. C., & Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 1*(1), 24–45.
Padberg, M. (1989). The boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming, 45*(1–3), 139–172.
Resende, M. G., & Ribeiro, C. C. (2003). Greedy randomized adaptive search procedures. In *Handbook of metaheuristics* (pp. 219–249). Springer.
de Souto, M. C., Costa, I. G., de Araujo, D. S., Ludermir, T. B., & Schliep, A. (2008). Clustering cancer gene expression data: A comparative study. *BMC Bioinformatics, 9*(1), 497.
Tanay, A., Sharan, R., & Shamir, R. (2005). Biclustering algorithms: A survey. *Handbook of Computational Molecular Biology, 9*(1–20), 122–124.
Wolsey, L. A., & Nemhauser, G. L. (1999). *Integer and combinatorial optimization*: 55. John Wiley & Sons.