Mathematical Programming Computation (2020) 12:39–68
https://doi.org/10.1007/s12532-019-00171-5

FULL LENGTH PAPER

A branch-and-price algorithm for capacitated hypergraph vertex separation

Michael Bastubbe[1] · Marco E. Lübbecke[1]

Received: 26 January 2018 / Accepted: 10 May 2019 / Published online: 9 September 2019

Abstract
We exactly solve the $\mathcal{NP}$-hard combinatorial optimization problem of finding a minimum cardinality vertex separator with k (or arbitrarily many) capacitated shores in a hypergraph. We present an exponential size integer programming formulation which we solve by branch-and-price. The pricing problem, an interesting optimization problem on its own, has a decomposable structure that we exploit in preprocessing. We perform an extensive computational study, in particular on hypergraphs coming from the application of re-arranging a matrix into single-bordered block-diagonal form. Our experimental results show that our proposal complements the previous exact approaches in terms of applicability for larger k, and significantly outperforms them in the case $k = \infty$.

Keywords Hypergraph · Balanced vertex separator · Matrix decomposition · Integer programming

Mathematics Subject Classification 90C27 · 90C09 · 49M27

1 Introduction

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with $e \subseteq \mathcal{V}$ for all $e \in \mathcal{E}$, a capacity $u \in \mathbb{N}_{>0}$, and an upper bound $k \in \mathbb{N}_{>0} \cup \{\infty\}$, the capacitated (or balanced) hypergraph vertex separator problem (CHVS) is to find a minimum cardinality subset of vertices $\mathcal{S} \subset \mathcal{V}$

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s12532-019-00171-5) contains supplementary material, which is available to authorized users.

✉ Marco E. Lübbecke
marco.luebbecke@rwth-aachen.de

Michael Bastubbe
michael.bastubbe@rwth-aachen.de

1 Lehrstuhl für Operations Research, RWTH Aachen University, Kackertstr. 7, 52072 Aachen, Germany

such that the remaining vertices decompose in at most k components (not necessarily connected) with at most u vertices each, i.e., there is no hyperedge incident to more than one component. These components are called shores. The goal is equivalent to maximizing the number of vertices in the shores.

The CHVS is not only $\mathcal{NP}$-hard, but also hard to approximate within an additive term away from the optimum, even when restricted to graphs with maximum node degree three and $k = 2$. This is an immediate consequence of Theorem 4.3 in [10].

We abbreviate $[n] := \{1, \ldots, n\}$ for any $n \in \mathbb{N}_{>0}$ and $\mathcal{E}(R) := \{e \in \mathcal{E} : R \cap e \neq \emptyset\}$ for any $R \subseteq \mathcal{V}$. For a single vertex $v \in \mathcal{V}$ we write $\mathcal{E}(v)$ instead of $\mathcal{E}(\{v\})$. Furthermore, we assume w.l.o.g. that there are no isolated vertices.

Applications and literature Our main motivation for studying the CHVS comes from a matrix decomposition problem: given a matrix $A \in \mathbb{R}^{n \times m}$, a number k of blocks, and a block capacity u, assign as many rows as possible to one of the blocks such that the number of rows assigned to each block is at most u. Two rows assigned to different blocks must not share a column having a nonzero entry in both of them. The set of unassigned rows is called the border. By re-arranging the rows and columns block-wise the matrix attains the so-called single-bordered block-diagonal form. Identifying rows with vertices and columns with nets (spanning exactly the vertices whose corresponding rows have a nonzero entry in this column), we obtain a bijection between instances (and solutions) of CHVS and the matrix decomposition problem, see Fig. 1. The single-bordered block-diagonal form has itself a vast number of applications in e.g., numerical linear algebra, see [19] for a survey. Examples of particular interest are the parallelized QR factorization [1], and determining how to apply a Dantzig-Wolfe reformulation to a mixed-integer linear program [8].

The matrix decomposition problem also motivated the only exact approach to the CHVS so far. Borndörfer et al. [9] propose a binary program which they solve by a tailored branch-and-cut algorithm. It is based on binary variables x_v^ℓ that equal 1 if and only if vertex v is part of shore ℓ. Their model reads as follows.

$$\begin{aligned} \max \quad & \sum_{\ell \in [k]} \sum_{v \in \mathcal{V}} x_v^\ell \\ \text{s.t.} \quad & \sum_{\ell \in [k]} x_v^\ell \leq 1 \quad \forall v \in \mathcal{V} \end{aligned} \tag{B.1}$$

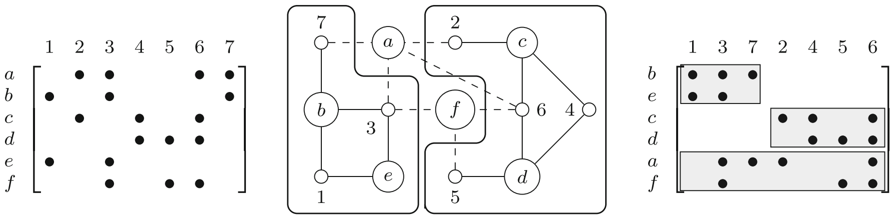


Fig. 1 Exploiting a relation to the CHVS, we are able to re-arrange a matrix (left) to single-bordered block-diagonal form (right). Black dots in the matrices represent non-zero entries. A smallest vertex separator (here: vertices a and f) corresponds to a minimum size border

$$\sum_{v \in \mathcal{V}} x_v^\ell \leq u \quad \forall \ell \in [k] \tag{B.2}$$

$$x_v^\ell + x_{v'}^{\ell'} \leq 1 \quad \forall \ell, \ell' \in [k], \quad \ell \neq \ell', \quad \forall v, v' : \mathcal{E}(v) \cap \mathcal{E}(v') \neq \emptyset, \tag{B.3}$$

$$x_v^\ell \in \{0, 1\}, \quad \forall \ell \in [k], \quad \forall v \in \mathcal{V}.$$

Every vertex can be part of at most one shore via Eq. (B.1) and each shore is capacitated, ensured by Eq. (B.2). Central to this model is the *conflict* that vertices that belong to a common net cannot be part of different shores, see Eq. (B.3). Based on these latter packing type of constraints, the model is strengthened by several classes of valid inequalities in [9]. Since formulation (B) is symmetric in index k, even the tailored approach struggles with larger k. Oosten et al. [25] suggest a non-symmetric binary program, however, they assume $k = \infty$. For the variant where the difference between component cardinalities is bounded by a parameter, Cornaz et al. [11] present a compact and an exponential-size formulation. The latter one can be seen as an intermediate step to what we propose in this paper. Yet, they can also hardly handle larger k. Most recently, Cornaz et al. [12] independently used similar ideas for the uncapacitated problem variant on graphs to cut into at least k components.

The CHVS appears in several other contexts, in particular when restricted to graphs, where network structure is of interest: removing few vertices from a graph such that a certain number of components remain is a common topic in graph clustering [25] and partitioning [15]. In communication applications such vertices have a certain criticality for the network and the cardinality of a separator is a proxy for network robustness [6].

An overview of heuristic approaches, mainly for $k = 2$, is given in [15]. Polyhedral results (other than those in [9] already mentioned) can be found in [3,13] ($k = 2$) and [25] ($k = \infty$). For $k = \infty$, the problem is polynomially solvable for several graph classes, also for the version with vertex weights [6].

Our contribution We model the CHVS as an exponential-size binary program in which the symmetry in k is eliminated. Our approach is the first to consistently solve instances with larger k and thus complements previous exact approaches that work better/only for smaller k. We design a branch-and-price algorithm, for which it is remarkable that branching on so-called aggregated original variables works well. One key component of this (theoretically incomplete) branching scheme is a repair algorithm that might solve an auxiliary Bin Packing problem to find an integer solution based on a fractional one. We discuss the complexity of the pricing problem, a variant of the Next Release Problem [2], and solve it by heuristic and exact approaches. The optimal re-arrangements of matrices into single-bordered block-diagonal form constitute a contribution on their own.

2 Branch-and-price algorithm

A slight modification of formulation (B) for the CHVS reads as follows.

$$\max \sum_{\ell \in [k]} \sum_{v \in \mathcal{V}} x_v^\ell$$

$$\text{s.t.} \quad \sum_{\ell \in [k]} y_e^\ell \leq 1 \quad \forall e \in \mathcal{E}, \tag{P.1}$$

$$\sum_{v \in \mathcal{V}} x_v^\ell \leq u \quad \forall \ell \in [k]\,, \tag{P.2}$$

$$x_v^\ell - y_e^\ell \leq 0 \quad \forall \ell \in [k]\,,\ \forall e \in \mathcal{E}, \quad \forall v \in e, \tag{P.3}$$

$$x_v^\ell,\ y_e^\ell \in \{0, 1\} \quad \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}, \quad \forall \ell \in [k]\,.$$

Binary variable x_v^ℓ equals 1 if and only if vertex v is part of shore ℓ. There is a binary variable y_e^ℓ for all $e \in \mathcal{E}$ and $\ell \in [k]$ that equals 1 if $x_v^\ell = 1$ for some $v \in e$ which is enforced by constraints (P.3). The inequalities (P.1) invoke that every hyperedge touches at most one shore. Notice that since there are no isolated vertices, every vertex is assigned to at most one shore. Furthermore constraints (P.2) accomplish that every shore includes at most u many vertices. The objective function maximizes the number of vertices assigned to some shore.

The drawbacks of this formulation are two-fold: firstly, the linear programming (LP) relaxation is weak; assigning $x_v^\ell := y_e^\ell := \min\{\frac{u}{m}, \frac{1}{k}\}$ yields a feasible solution with objective value equal to $\min\{ku, m\}$ which are trivial bounds. Secondly, the formulation is highly symmetric, as for any feasible solution of (P) every permutation of shore indices ℓ yields another feasible solution of (P).

2.1 A shore based formulation

Let $\mathcal{R} := \{R \subseteq \mathcal{V} : |R| \leq u\}$ denote the set of all possible shores, e.g., vertex subsets with cardinality at most u. We consider the following natural shore-based ILP formulation (M), which formally is a Dantzig-Wolfe reformulation of (P): for every $\ell \in [k]$ one reformulates the corresponding constraints (P.2) and (P.3) into a separate subproblem, resulting in k identical subproblems that will be aggregated into one single subproblem, thereby eliminating the symmetry of (P). The remaining constraints (P.1) form the master problem.

$$\max \quad \sum_{R \in \mathcal{R}} |R| \lambda_R$$

$$\text{s.t.} \quad \sum_{\substack{R \in \mathcal{R}: \\ e \cap R \neq \emptyset}} \lambda_R \leq 1 \quad (\beta_e) \quad \forall e \in \mathcal{E} \tag{M.1}$$

$$\sum_{R \in \mathcal{R}} \lambda_R \leq k \quad (\gamma) \tag{M.2}$$

$$\lambda_R \in \{0, 1\} \quad \forall R \in \mathcal{R}.$$

Variable λ_R takes value 1 if and only we select a shore consisting exactly of the vertices in R. The objective function maximizes the number of vertices assigned to some shore.

Constraints (M.1) ensure that for every hyperedge e there is at most one shore including a vertex incident to e. Hence there are no two shores sharing a hyperedge.

Constraint (M.2) assures that at most k shores are chosen. The LP relaxation of (M) is denoted by (MLP), in which the upper bounds on the variables need not be explicitly stated because of Eq. (M.1). The dual variables to the respective constraints are indicated in brackets.

Formulation (M) has $\sum_{i=0}^{u} \binom{m}{i} \geq 2^u$ variables, therefore we solve it by branch-and-price, i.e., the relaxation (MLP) is solved by column generation [23]. We assume the reader to be familiar with both concepts. The restricted master problem arises from (M) by only considering a subset $\bar{\mathcal{R}}$ of $\mathcal{R}$. Its LP relaxation is denoted by (RMLP). If there is no $R \in \mathcal{R}\backslash\bar{\mathcal{R}}$ such that λ_R has positive reduced cost, an optimal solution of (RMLP) is also optimal for (MLP). Otherwise, we add at least one R to $\bar{\mathcal{R}}$ with λ_R having positive reduced cost, and solve (RMLP) again, see Sect. 2.3. Note that, in principle, (RMLP) is feasible for $\bar{\mathcal{R}} = \emptyset$.

2.2 Branching

When (MLP) is solved to optimality, the optimal λ' might not be integer, i.e., $\lambda' \notin \{0, 1\}^{\mathcal{R}}$. When there is a $v \in \mathcal{V}$ with $z_v^{\lambda'} := \sum_{R \in \mathcal{R}: v \in R} \lambda'_R \in (0, 1)$ we branch on the dichotomy that v is either part of the separator or a shore. This is realized by imposing constraints $z_v^{\lambda'} = 0$ and $z_v^{\lambda'} = 1$, respectively, in the two child nodes in the branch-and-price tree. Note that this can be interpreted as branching on aggregated original x-variables of (P): $\sum_{\ell=1}^{k} x_v^{\ell} \notin \{0, 1\}$ for $v \in \mathcal{V}$. However, this branching scheme is not complete in theory, as the following example shows.

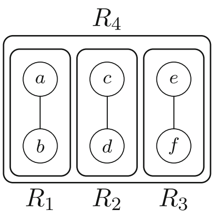
Example 1 Let $\mathcal{H} = (\{a, b, c, d, e, f\}, \{\{a, b\}, \{c, d\}, \{e, f\}\})$, $k = 2$, and $u = 6$. Then, for $\bar{\mathcal{R}} := \{\{a, b\}, \{c, d\}, \{e, f\}, \{a, b, c, d, e, f\}\}$ the solution $\lambda'_R = 0.5$ for $R \in \bar{\mathcal{R}}$ and $\lambda'_R = 0$ for $R \in \mathcal{R}\backslash\bar{\mathcal{R}}$ is basic feasible and $z_i^{\lambda'} = 1$ for $i \in [4]$. For a visualization of the solution and the relevant full rank part of the basis matrix see Fig. 2.

In the case that $\lambda' \notin \{0, 1\}^{\mathcal{R}}$ but $z_v^{\lambda'} \in \{0, 1\}$ we are able (under mild assumptions) to retrieve an integral solution of the same objective function value as λ' by solving an auxiliary Bin Packing problem. Define $V^{\lambda'} := \{v \in \mathcal{V} : z_v^{\lambda'} = 1\}$ and let $\mathcal{H}[V^{\lambda'}]$ denote the hypergraph induced by $V^{\lambda'}$.

Proposition 1 *Let λ' be a solution of (RMLP) with $z_v^{\lambda'} \in \{0, 1\}$. Then for every connected component C of $\mathcal{H}[V^{\lambda'}]$ it holds that $|C| \leq u$.*

Proof Let C be a connected component of $\mathcal{H}[V^{\lambda'}]$ and let $v \in C$ and $e \in \mathcal{E}$ with $v \in e$. Define $\mathcal{R}_w^+ := \{R \in \mathcal{R} : \lambda'_R > 0, w \in R\}$. Since $\{R \in \mathcal{R} : v \in R\} \subseteq \{R \in$

Fig. 2 Example for incomplete branching, i.e., $\lambda' \notin \{0, 1\}^{\mathcal{R}}$ but $z_v^{\lambda'} \in \{0, 1\}$



$$\begin{array}{l} \\ M.1(ab) \\ M.1(cd) \\ M.1(ef) \\ M.2 \end{array} \begin{array}{cccc} R_1 & R_2 & R_3 & R_4 \\ \end{array}$$

$$\begin{array}{l} M.1(ab) \\ M.1(cd) \\ M.1(ef) \\ M.2 \end{array} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$\mathcal{R} : e \cap R \neq \emptyset\}$ we have $1 = z_v^{\lambda'} = \sum_{R\in\mathcal{R}:v\in R} \lambda'_R \leq \sum_{R\in\mathcal{R}:e\cap R\neq\emptyset} \lambda'_R \leq 1$ which holds with equality. Hence $\mathcal{R}^+_{v_1} = \mathcal{R}^+_{v_2}$ for adjacent vertices $v_1, v_2 \in C$, and since C is connected also $\mathcal{R}^+_{w_1} = \mathcal{R}^+_{w_2}$ for arbitrary vertices $w_1, w_2 \in C$. Therefore, $C \subseteq R$ for all $R \in \mathcal{R}^+_v$ (since $R \in \mathcal{R}^+_w$ for every $w \in C$ and thus $w \in R$) and $|C| \leq u$. □

Recall that for the Bin Packing problem items of non-negative weight must be assigned to bins such that the total weight in each bin does not exceed the bin capacity and the number of used bins is minimum.

Definition 1 Let $\mathcal{H}$ be a hypergraph with connected components $C_1, \dots, C_h$, and $u \in \mathbb{N}$. The Bin Packing *instance associated to* $(\mathcal{H}, u)$ has h items of size $|C_i|$ for item i and bin capacity u.

The classical Gilmore-Gomory formulation [17] to solve such an instance reads:

$$
\begin{aligned}
w^* = \min & \sum_{P\in\mathcal{P}} \mu_P \\
\text{s.t.} & \sum_{\substack{P\in\mathcal{P}:\\ i\in P}} \mu_P = 1 \quad \forall i \in [h] \\
& \mu_P \in \{0, 1\} \quad \forall P \in \mathcal{P},
\end{aligned}
$$

where we define $\mathcal{P} = \{P \subseteq [h] : \sum_{i\in P} |C_i| \leq u\}$. Let w^*_{LP} denote the optimum of its LP relaxation.

Following the general definition from [5], an instance of the Bin Packing problem has the *integer round-up property* if $w^* = \lceil w^*_{\text{LP}} \rceil$ for that instance.

Proposition 2 *Let λ' be a solution of (RMLP) with $z_v^{\lambda'} \in \{0, 1\}$. If the* Bin Packing *instance associated to $(\mathcal{H}[V^{\lambda'}], u)$ has the integer round-up property, there exists an integer solution $\bar{\lambda}$ of (M) with $\sum_{R\in\mathcal{R}}|R|\lambda'_R = \sum_{R\in\mathcal{R}}|R|\bar{\lambda}_R$.*

Proof Let λ' be a solution of (RMLP) with $z_v^{\lambda'} \in \{0, 1\}$, $v \in \mathcal{V}$. Let $C_1, \dots, C_h$ be the connected components of $H[V^{\lambda'}]$ and let the Bin Packing instance B associated to $(\mathcal{H}[V^{\lambda'}], u)$ have the integer round-up property. We have seen in the proof of Proposition 1 that for all connected components C of $\mathcal{H}[V^{\lambda'}]$ with $R \cap C \neq \emptyset$ for R with $\lambda'_R > 0$ it holds that $C \subseteq R$. We define $P(R) := \{i \in [h] : C_i \subseteq R\}$ for $R \in \mathcal{R}$ with $\lambda'_R > 0$. Since $C_{i_1} \cap C_{i_2} = \emptyset$ for $i_1, i_2 \in [h]$ with $i_1 \neq i_2$ we have $\sum_{i\in P(R)}|C_i| \leq |R| \leq u$. We define μ such that $\mu_{P(R)} := \lambda'_R$ for R with $\lambda'_R > 0$ and $\mu_P := 0$ for all remaining $P \in \mathcal{P}$ [with $P \neq P(R)$ for all R with $\lambda'_R > 0$]. Then μ is a fractional solution of the Gilmore-Gomory formulation of B with objective value $w_{\lambda'} = \sum_{R\in\mathcal{R}} \lambda'_R$. With w^* and w^*_{LP} denoting the optimum values of the Gilmore-Gomory formulation of B and its LP relaxation, respectively, we get $k \geq \lceil w_{\lambda'} \rceil \geq \lceil w^*_{\text{LP}} \rceil = w^*$. The last equality holds since B has the integer round-up property. Every solution for B with objective function value of ℓ can be translated to a solution of CHVS using ℓ shores by assigning all vertices of components with corresponding item in the same bin to the same shore. Let $\bar{\lambda}$ be such a solution originating from an optimal solution of B. Then $\bar{\lambda}$ is also feasible for (M) with $V^{\lambda'} = V^{\bar{\lambda}}$ and thus $\sum_{R\in\mathcal{R}}|R|\lambda'_R = \sum_{R\in\mathcal{R}}|R|\bar{\lambda}_R$. □

Based on these results, we formulate Algorithm 1 to retrieve an integer solution from a fractional solution λ', showing that no branching is necessary for the current branch-and-price node when $z_v^{\lambda'} \in \{0, 1\}$. As a consequence of Proposition 1 Algorithm 1 never terminates in line 3 if the input hypergraph $\mathcal{H}$ is of the form $\mathcal{H} = \mathcal{H}[V^{\lambda'}]$ with $z_v^{\lambda'} \in \{0, 1\}$ for λ'. Hence, it terminates either (a) by finding a feasible solution with the same objective function value as an optimal solution of (RMLP) in the current node, or (b) in line 9. Case (b) happens if and only if the integer round-up property does *not* hold for the Bin Packing instance associated to $(\mathcal{H}, u)$. It is an open question whether this can actually occur, but in our numerous computational tests it never did. However, we believe that a pathological example can be crafted, so we state.

Conjecture 1 The Bin Packing instances constructed for use in Algorithm 1 do not always have the integer round-up property.

To be on the safe side, in "Appendix A.2" (online only) we present a fallback branching rule that is essentially Ryan-Foster's [26], to cover this case theoretically.

Algorithm 1 Retrieve feasible integer solution

input: Hypergraph $\mathcal{H}$, capacity u, maximal number of shores k
output: Feasible shores $(S_1, \ldots, S_k)$, encoded in λ
1 find $C = (C_1, \ldots, C_r)$ connected components of $\mathcal{H}$
2 **if** $\exists b \in [r] : |C_b| > u$ **then**
3 | state that no such solution exists and **return**
4 **if** $r > k$ **then** `// found more components than shores allowed`
5 | $R = (R_1, \ldots, R_h) \leftarrow$ solve Bin Packing problem associated to $(\mathcal{H}, u)$
6 | **if** $h \leq k$ **then** `// found assignment of components to shores`
7 | | set λ according to R
8 | **else**
9 | | state that no such solution exists and **return**
10 **else**
11 | set λ according to C
12 **return** λ

Remark 1 The first Bin Packing instance that does not have the integer round-up property was found by Marcotte [24] in 1986. More recently, Kartak et al. [18] computationally showed that there are instances with 10 items that do not have the integer round-up property and that all instances with 9 or less items have it.

2.3 Pricing

In the pricing problem we find a variable with (maximum) positive reduced cost to add to (RMLP), or prove that none exists. In the root node of the branch-and-price tree the reduced cost $\bar{c}_R$ of variable λ_R for $R \subseteq \mathcal{V}$ with $|R| \leq u$ is $|R| - \sum_{e \in \mathcal{E}(R)} \beta_e - \gamma$. Branching decisions, further down the tree, can easily be respected (this is also true for the fallback branching as described in "Appendix A.2"): if $z_v^{\lambda'} = 0$ for some $v \in \mathcal{V}$, vertex v cannot be chosen for any shore; therefore we just do not consider it. If $z_v^{\lambda'} = 1$ for some $v \in \mathcal{V}$, vertex v has to be chosen for exactly one shore. Hence we have to consider the value of the corresponding dual variable α_v of $\sum_{R\in\mathcal{R}:v\in R}\lambda_R = 1$ and the reduced costs become

$$\bar{c}_R = |R| - \left(\sum_{v\in R}\alpha_v + \sum_{e\in\mathcal{E}(R)}\beta_e + \gamma\right) = \sum_{v\in R}(1-\alpha_v) - \sum_{e\in\mathcal{E}(R)}\beta_e - \gamma.$$

Hence, the pricing problem is to find a subset of vertices $R \subseteq \mathcal{V}$ with $|R| \leq u$ such that an objective function of the form $c^*_R := \sum_{v\in R} p_v - \sum_{e\in\mathcal{E}(R)} c_e$ for $p \in \mathbb{R}^{\mathcal{V}}$, and $c \in \mathbb{R}^{\mathcal{E}}_{+}$ is maximized. We denote this problem as PR and a specific instance as PR$(\mathcal{H}, p, c, u)$. Note that we can assume w.l.o.g. that $p_v > 0$ (v could otherwise be excluded from any optimal solution) and $c_e \geq 0$ (since $\beta_e \geq 0$).

2.3.1 Applications

Problem PR can be seen as a variant of the NEXT RELEASE PROBLEM [2]: we are given a set of customers I and a set of possible software enhancements R, with profits p_i for every customer $i \in I$, programming costs c_j for every enhancement $j \in R$, and a set of demanded enhancements $R_i \subseteq R$ for every customer i. The task is to find a subset of customers S such that the total profit $\sum_{i\in S} p_i$ is maximum and the needed programming costs $\sum_{j\in\bigcup_{i\in S} R_i} c_j$ do not exceed a given value. In problem PR the programming costs are part of the objective function and the number of chosen customers is bounded by a given value.

2.3.2 Complexity

Proposition 3 *The pricing problem* PR *is $\mathcal{NP}$-hard.*

Proof We reduce the CLIQUE problem to PR. Consider an instance of the decision variant of CLIQUE, i.e., an integer ℓ and an undirected graph $G = (V, E)$. The task is to find a clique in G with at least ℓ nodes if one exists. The reduction works as follows: we take G as input for PR assigning unit costs $c_e = 1, e \in E$, to the edges and "irresistable" profits $p_v = |E| + 1, v \in V$, to the vertices. By solving PR(G, p, c, u) one gets a subset of nodes R with $|R| = u$ such that the number of edges $\mathcal{E}(R)$ having at least one end point in R is minimum. Hence $V\backslash R$ is a subset of nodes with $|V\backslash R| = |V| - u$ such that the number of edges with both end points in $V\backslash R$ is maximized. Therefore it can be checked if there exists a clique in G with exactly $|V| - u$ nodes by checking if $|E| - |\mathcal{E}(R)| = \binom{|V|-u}{2}$. Thus by solving PR$(G, p, c, u)$ for $u = 0, 1, \ldots, |V| - \ell$ one can check whether G has a clique with at least ℓ nodes. □

The following result (we give a proof in "Appendix A.1", online only) was already observed by Barahona and Jensen [4], who found bounds for a location problem with inventory cost by applying Dantzig-Wolfe decomposition. The pricing problem they solved is PR with relaxed cardinality constraint, which is polynomially solvable, and in particular an optimal solution can be calculated by finding a minimum s-t cut:

Proposition 4 *The problem* PR *is polynomially solvable for $u = |\mathcal{V}|$.*

2.3.3 Preprocessing the pricing problem

The following proposition states that unprofitable vertices can be identified by solving what we call the uncapacitated instance $\textsc{Pr}(\mathcal{H}, p, c, |\mathcal{V}|)$, i.e., with $u = |\mathcal{V}|$. We denote the set of optimal solutions to $\textsc{Pr}(\mathcal{H}, p, c, u)$ by $\mathcal{R}_u^*$.

Proposition 5 *Let $R'_\infty \in \mathcal{R}^*_{|\mathcal{V}|}$ denote an optimal solution to the uncapacitated instance $\textsc{Pr}(\mathcal{H}, p, c, |\mathcal{V}|)$. For all $u \in \mathbb{N}$ there exists an optimal solution $R \in \mathcal{R}_u^*$ with $R \subseteq R'_\infty$.*

Proof Consider an optimal solution $R'_u \in \mathcal{R}_u^*$ for the capacitated instance. We show that $R := R'_u \cap R'_\infty$ is an optimal solution for $\textsc{Pr}(\mathcal{H}, p, c, u)$. Clearly, the vertex set $R_\infty := R'_u \cup R'_\infty$ is a solution for $\textsc{Pr}(\mathcal{H}, p, c, |\mathcal{V}|)$. Since $\mathcal{E}(R) = \mathcal{E}(R'_u \cap R'_\infty) \subseteq \mathcal{E}(R'_u) \cap \mathcal{E}(R'_\infty)$, and $c_e \geq 0$ for all $e \in \mathcal{E}$, we obtain

$$
\begin{aligned}
c_R^* + c_{R_\infty}^* &= c_R^* + c_{R'_u}^* + c_{R'_\infty}^* - \sum_{v \in R'_u \cap R'_\infty} p_v + \sum_{e \in \mathcal{E}(R'_u) \cap \mathcal{E}(R'_\infty)} c_e \\
&\geq c_R^* + c_{R'_u}^* + c_{R'_\infty}^* - \sum_{v \in R} p_v + \sum_{e \in \mathcal{E}(R)} c_e \\
&= c_{R'_u}^* + c_{R'_\infty}^*.
\end{aligned}
$$

By definition, R is feasible for $\textsc{Pr}(\mathcal{H}, p, c, u)$, and because R'_u and R'_∞ are optimal for $\textsc{Pr}(\mathcal{H}, p, c, u)$ and $\textsc{Pr}(\mathcal{H}, p, c, |\mathcal{V}|)$, respectively, also R and R_∞ are optimal for them, respectively. □

Remark 2 We use Proposition 4 to preprocess the instance of the pricing problem. More precisely, we solve $\textsc{Pr}(\mathcal{H}, p, c, |\mathcal{V}|)$ and exclude (for this single pricing iteration) all vertices that are not part of an optimal solution of $\textsc{Pr}(\mathcal{H}, p, c, |\mathcal{V}|)$.

In the following, we present several algorithms that solve the pricing problem heuristically or exactly. All of them will be used in our implementation.

2.3.4 Greedy heuristic

For a subset of vertices $R \subseteq \mathcal{V}$ and a vertex $v \notin R$ we compute the change of the objective function value that would occur by including v in R by $c(R, v) := p_v - \sum_{e \in \mathcal{E}(v) \setminus \mathcal{E}(R)} c_e$. Starting with an initially empty set Algorithm 2 greedily adds a vertex that is locally most profitable until u vertices are included. For every intermediate R the corresponding variable λ_R is added if its reduced cost is positive.

Algorithm 2 Greedy pricing heuristic

input: Hypergraph $\mathcal{H}$, capacity u, maximal number of shores k
output: Feasible shores $(S_1, \dots, S_k)$

$R = \emptyset$
while $|R| < u$ **do**
  $w = \arg\max_{v \in \mathcal{V} \setminus R} c(R, v)$
  $R = R \cup \{w\}$
  if $\bar{c}_R > 0$ **then**
    add λ_R to (RMLP)
  end if
end while
return

2.3.5 Multi-start iterated local search

The following Algorithm 3 is a multi-start iterated local search with changing neighborhoods. It is started for several runs with a random initial solution and in each iteration a neighbor of largest improvement is chosen. For the number of neighborhood types we choose $\bar{\ell} = 4$. In every iteration we have a feasible solution $R \in \mathcal{R}$ for $\mathrm{PR}(\mathcal{H}, p, c, u)$ and a neighborhood level $\ell \in [\bar{\ell}]$. Then R is set to a most profitable improving neighbor in $\arg\max_{R' \in \mathcal{N}_\ell(R)} \bar{c}_{R'}$ with $\max_{R' \in \mathcal{N}_\ell(R)} \bar{c}_{R'} > \bar{c}_R$ that is found by enumeration of $\mathcal{N}_\ell(R)$ if it exists, and the neighborhood level ℓ is reset to 1. Otherwise, the neighborhood level is increased by one if $\ell \neq \bar{\ell}$. If all neighborhoods are exhausted we add λ_R to (RMLP) if $\bar{c}_R > 0$.

Algorithm 3 Iterated local search pricing heuristic

input: $A \in \mathbb{R}^{m \times n}$, $k \in \mathbb{N}$, $u \in \mathbb{N}$, $z \in \{0, 1\}^m$
output: feasible solution to the pricing problem

create a random solution R
$\ell = 1$
while $\ell \leq \bar{\ell}$ **do**
  if $\max_{R' \in \mathcal{N}_\ell(R)} \bar{c}_{R'} > \bar{c}_R$ **then**
    $R \in \arg\max_{R' \in \mathcal{N}_\ell(R)} \bar{c}_{R'}$
    $\ell = 1$
  else
    $\ell = \ell + 1$
end while
if $\bar{c}_R > 0$ **then**
  add λ_R to (RMLP)
end if
return

The neighborhood types we use are $\mathcal{N}_1 := \mathcal{N}_1^{\mathcal{V}}, \mathcal{N}_2 := \mathcal{N}_1^{\mathcal{E}}, \mathcal{N}_3 := \mathcal{N}_2^{\mathcal{E}}, \mathcal{N}_4 := \mathcal{N}_3^{\mathcal{E}}$, with $\mathcal{N}_\ell^{\mathcal{V}}(R) := \{\bar{R} \in \mathcal{R} : |R \triangle \bar{R}| = \ell\}$ and $\mathcal{N}_\ell^{\mathcal{E}}(R) := \{\bar{R} \in \mathcal{R} : |\mathcal{E}(R) \triangle \mathcal{E}(\bar{R})| = \ell\}$ with the symmetric difference $A \triangle B := (A \setminus B) \cup (B \setminus A)$. In our implementation the neighborhood $\mathcal{N}_4$ is explored at most once in every run of the algorithm.

2.3.6 Integer linear program

The following binary program (Pr) can be used to solve $\textsc{Pr}(\mathcal{H}, p, c, u)$ if $c_e \geq 0$ for all $e \in \mathcal{E}$:

$$\max \quad \sum_{v \in \mathcal{V}} p_v x_v - \sum_{e \in \mathcal{E}} c_e y_e \tag{Pr.1}$$

$$\text{s.t.} \quad \sum_{v \in \mathcal{V}} x_v \leq u \tag{Pr.2}$$

$$x_v - y_e \leq 0 \quad \forall v \in \mathcal{V}, e \in \mathcal{E} : v \in e, \tag{Pr.3}$$

$$x_v,\ y_e \in \{0, 1\}, \quad \forall v \in \mathcal{V},\ e \in \mathcal{E}. \tag{Pr.4}$$

Variable x_v equals 1 if and only if $R \ni v$ and Eq. (Pr.2) ensures that at most u vertices are chosen. Variable y_e attains value of 1 if $\mathcal{E}(R) \ni e$ which is guaranteed by Eq. (Pr.3).

2.3.7 Complementary pricing

The greedy and local search heuristics and finally the exact ILP (Pr) are run in cascade. Additionally, after any one pricing algorithm found a variable λ_R with positive reduced cost, this algorithm is restarted on $\mathcal{H}' = (\mathcal{V} \backslash R,\ \mathcal{E}[\mathcal{V} \backslash R])$. This can be applied repeatedly in rounds. If no variable with positive reduced cost is found, no restart takes place. The resulting complementary subsets are supposed to combine well to integer feasible solutions, see e.g., [16].

2.4 Preprocessing the hypergraph

We preprocess $\mathcal{H}$ in two phases: in phase 1 we only remove hyperedges that are contained in others. If there are identical hyperedges we remove all of them but one. This can be easily done by enumeration.

The idea of phase 2 is to express the conflicts between vertices (imposed by hyperedges) by a smaller set of hyperedges. Consider the clique graph of $\mathcal{H}$ defined as $G(\mathcal{H}) := (\mathcal{V}, E := \{vw \mid \exists e \in \mathcal{E} : v, w \in e\})$. Two hypergraphs $\mathcal{H}, \mathcal{H}_1$ with identical clique graph $G(\mathcal{H}) = G(\mathcal{H}_1)$ yield identical solution spaces for the CHVS. In order to find a simple such hypergraph $\mathcal{H}_1$ we search for a minimum clique edge cover in $G(\mathcal{H})$. A (minimum) clique edge cover $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ of a graph is a (minimum cardinality) set of cliques such that each $e \in \mathcal{E}$ is a subset of at least one clique, i.e., there is an $h \in [\ell]$ with $e \subseteq C_h$. Then we replace each clique $C \in \mathcal{C}$ by a hyperedge spanning exactly the vertices $q \in C$. Thus we obtain $\mathcal{H}_1$ with $G(\mathcal{H}) = G(\mathcal{H}_1)$.

Since the hyperedges of $\mathcal{H}$ represent a clique edge cover in $G(\mathcal{H})$ of cardinality m, we can assume that $\ell \leq m$ and hence the new number of hyperedges would not be increased if the clique edge cover is minimum. In practice we use the polynomial-time heuristic of Kou et al. [22] that was based on a heuristic by Kellerman [20] and replaces the original hyperedges by the found ones if their number is decreased.

2.5 Primal heuristic

Algorithm 1 can be used to verify whether a subset of vertices S disconnects $\mathcal{H}$ in at most k components of cardinality at most u. We exploit this fact by using it as a primal heuristic during the solution of the restricted master problem. In order to get a potential separator S we randomly round the possibly fractional solution values of $z_v^{\lambda'} = \sum_{R \in \mathcal{R}: v \in R} \lambda'_R$. More specifically, a vertex $v \in \mathcal{V}$ is added to S with probability $z_v^{\lambda} + \delta$ where λ is the current LP solution, and $\delta \in \{-0.001, 0, 0.05, 0.1, 0.2, 0.3\}$ fixed randomly equally distributed for this run of the heuristic. The number of runs is 200 and the heuristic is called directly after solving a branch-and-price node and after every 50 column generation iterations.

2.6 Exchange vectors

For a given subset of vertices $R \subseteq \mathcal{V}$ and a hyperedge $e \in \mathcal{E}(R)$ one can easily construct $R'(R, e) := R \setminus e$. By adding an artificial variable ν_e for every $e \in \mathcal{E}$ corresponding to removing e with all $v \in e$ from a shore one implicitly can use $\bigcup_{R \in \bar{\mathcal{R}}} \bigcup_{e \in \mathcal{E}} R'(R, e)$ pattern variables that are not explicit part of the model when solving (RMLP). Note again that in formulation (MLP) the upper bound constraints $\lambda_R \leq 1$ are already implied by Eq. (M.1) (since there are no isolated vertices). We obtain the following augmented master LP formulation (AMLP):

$$\max \quad \sum_{R \in \mathcal{R}} |R| \lambda_R - \sum_{e \in \mathcal{E}} |e| \nu_e$$

$$\text{s.t.} \quad \sum_{R \in \mathcal{R}: e \in \mathcal{E}(R)} \lambda_R - \nu_e \leq 1 \quad \forall e \in \mathcal{E} \tag{AMLP.1}$$

$$\sum_{R \in \mathcal{R}} \lambda_R \leq k \tag{AMLP.2}$$

$$\lambda_R \geq 0 \quad \forall R \in \mathcal{R}$$

$$\nu_e \geq 0 \quad \forall e \in \mathcal{E}$$

The new variables, their coefficient columns are called *exchange vectors*, translate to the following constraints in the dual (called *dual-optimal inequalities* [7]):

$$\beta_e \leq |e| \quad \forall e \in \mathcal{E}$$

The following proposition states their validity, i.e., that (MLP) and (AMLP) are equivalent.

Proposition 6 *Let z_{MLP} and z_{AMLP} denote the respective optima of (MLP) and (AMLP). Then $z_{MLP} = z_{AMLP}$.*

Proof We use the dual (DMLP) of formulation (MLP)

$$
\begin{aligned}
\min \quad & \sum_{e\in\mathcal{E}} \beta_e + k\gamma \\
\text{s.t.} \quad & \sum_{e\in\mathcal{E}(R)} \beta_e + \gamma \ge |R| \quad \forall R \in \mathcal{R} && \text{(DMLP.1)} \\
& \beta_e, \gamma \ge 0 \quad \forall e \in \mathcal{E},
\end{aligned}
$$

and show that the inequalities $\beta_e \le |e|$ for $e \in \mathcal{E}$ are fulfilled by all optimal solutions of (DMLP). Assume by contradiction that there is a an optimal solution (β^*, γ^*) to (DMLP) with $\beta^*_{e'} > |e'|$ for some $e' \in \mathcal{E}$. Then, there exists at least one subset of vertices $R' \in \mathcal{R}$ with $e' \cap R' \neq \emptyset$ and $\sum_{e\in\mathcal{E}(R')} \beta^*_e + \gamma^* = |R'|$ (otherwise $\beta^*_{e'}$ could be reduced, contradicting optimality). In particular, $|R'| \ge \beta^*_{e'} > |e'|$ and hence, $\tilde{R} := R' \backslash e'$ is nonempty and $|\tilde{R}| + |e'| \ge |R'|$. Since $\mathcal{E}(R') \supseteq \mathcal{E}(\tilde{R}) \cup \{e'\}$, we get

$$
\begin{aligned}
\sum_{e\in\mathcal{E}(\tilde{R})} \beta^*_e + \gamma^* + |e'| &\ge |\tilde{R}| + |e'| \ge |R'| \\
&= \sum_{e\in\mathcal{E}(R')} \beta^*_e + \gamma^* \ge \sum_{e\in\mathcal{E}(\tilde{R})} \beta^*_e + \beta^*_{e'} + \gamma^*,
\end{aligned}
$$

contradicting $\beta^*_{e'} > |e'|$. □

3 Computational results

We first introduce our computational environment. We then compare our algorithm with a commercial solver applied to the original formulation for different values of k. Thirdly, we investigate the influence of different algorithmic ingredients proposed in the previous section. Finally, we visualize some matrix decompositions complementing those known from the literature. This section contains mainly aggregate information; details can be found in the online appendix.

3.1 Implementation

Our algorithm denoted as *base* is a branch-and-price algorithm to solve formulation (M). Its default settings were derived in extensive experiments with the described features, see Sect. 3.7. The branching is executed as described in Sect. 2.2. The three pricing algorithms (Sects. 2.3.4–2.3.6) run in the following order: greedy heuristic, local search heuristic, and integer linear program (Pr). If a variable with positive reduced cost is found, the remaining pricing algorithm(s) will not be called. We set the maximal number of complementary pricing rounds (Sect. 2.3.7) for each algorithm to 8. The local search pricing heuristic is restarted three times for each complementary round. The preprocessing of the pricing problem (described in Sect. 2.3.3) is executed before each call of the exact pricing algorithm, i.e., solving formulation (Pr). Primal heuristic and preprocessing (of the hypergraph) are implemented as described in Sects. 2.4 and 2.5, respectively. The exchange vectors (cf. Sect. 2.6) are disabled by default.

3.2 Environment

The branch-and price algorithm is implemented in SCIP 3.2.0 with CPLEX 12.6.1 running on a single thread using default settings (with the exception that dual simplex optimizer is used) as a solver for the IP subproblems. The original formulation is also solved by CPLEX 12.6.1 under exactly the same conditions. All computations were performed on Intel Core i7-2600 CPUs with 16 GB of RAM on openSUSE 12.1 workstations running Linux kernel 3.1.10. The default time limit is 1800 s.

3.3 Instances

We consider four different groups of instances (details are in the online "Appendix A.3"):
`netlib` These 55 instances arise from basis matrices of linear programs in the NETLIB. These were also used by Borndörfer et al. [9]. We select all instances with up to 500 vertices (rows) that are not too small (more than 50 vertices). See Table 1 for more details on the instances.

`dimacs` The second group of 40 instances originates from graph coloring problems that were solved at the second DIMACS challenge. These graphs were used in [13]. We select all non-tiny (at least 20 vertices) instances with up to 500 vertices. A detailed description can be found in Table 2.

`miplib` The third group consists of 37 coefficient matrices originating from presolved mixed integer programs from MIPLIB2010 [21]. Borndörfer et al. [9] use similar instances from MIPLIB 3.0. Our instances are presolved by SCIP 3.2.0 with default settings. We report some characteristics in Table 3.

`random` Finally, we randomly constructed a test set of 50 hypergraphs based on 10 groups with different characteristics. The construction works as follows: for group $i \in \{1, \ldots, 5\}$ or group $j \in \{6, \ldots, 10\}$, the number of vertices is i.i.d. in $[25(i+1), 25(i+2)]$ or $[25(j-4), 25(j-3)]$, respectively. The number of hyperedges is i.i.d. in [50, 75] for groups 1 through 5 and from [75, 100] for groups 6 through 10. The cardinality of each hyperedge is i.i.d. in [2, 4] and the spanning vertices are chosen randomly as well. Details of the resulting instances can be found in Table 4.

Summarized instance information At the end of this paragraph we display some aggregated instance information for the testsets. Columns headed $|\mathcal{V}_i|$ list the arithmetic mean of the number of vertices before ($i = 0$) and after ($i = 1$) presolving. Columns headed $|\mathcal{E}_i|$ contain the arithmetic mean of the number of hyperedges in the original graph ($i = 0$), and in the presolved hypergraph after phase 1 ($i = 1$) and phase 2 ($i = 2$), see Sect. 2.4. The columns that are indicated by $D_i^{\mathcal{E}}$, $i \in \{0, 1, 2\}$ contain the average cardinality of a hyperedge in the respective hypergraph. The last column shows the density of the clique graph $G(\mathcal{H})$.

Table 1 Instance information of `netlib` testset

Name	$\|\mathcal{V}_0\|$	$\|\mathcal{V}_1\|$	$\|\mathcal{E}_0\|$	$\|\mathcal{E}_1\|$	$\|\mathcal{E}_2\|$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
vtpbase	51	50	51	38	19	3.8	4.7	8.6	0.277
bore3d	52	52	52	29	26	5.9	9	9.5	0.463
adlittle	53	51	53	37	34	3.8	4.5	4.7	0.173
blend	54	52	54	30	23	5.7	9.2	9.6	0.382
recipe	55	55	55	24	24	1.8	2.8	2.8	0.086
scagr7	58	58	58	34	33	4.1	6.1	6.2	0.399
sc105	59	57	59	50	50	3.7	4.1	4.1	0.208
stocfor1	62	53	62	34	34	2.9	4.4	4.4	0.143
scsd1	77	76	77	71	67	2.7	2.9	2.9	0.069
beaconfd	90	90	90	48	48	6.8	12	12	0.299
share2b	93	93	93	37	37	5.1	9.1	9.1	0.144
share1b	102	100	102	58	57	4.7	6.3	6.4	0.095
forplan	104	102	104	72	71	5.5	7.4	7.5	0.215
scorpion	105	94	105	57	45	3.6	5.4	5.3	0.091
brandy	113	113	113	83	78	7.7	9.3	9.7	0.254
sc205	113	113	113	104	104	6.1	6.5	6.5	0.246
boeing2	122	112	122	80	80	3.5	4.9	4.9	0.1
lotfi	122	112	122	70	70	2.8	3.9	3.9	0.071
tuff	137	131	137	90	79	5.9	8.3	8.7	0.157
grow7	140	140	140	51	51	11.8	4.7	4.7	0.14
scsd6	147	145	147	140	140	2.6	2.6	2.6	0.031
e226	148	147	148	90	79	6.4	9.1	9.3	0.141
israel	163	162	163	38	26	8.1	26.7	38.2	0.804
agg	164	159	164	58	58	4	9.7	9.7	0.126
capri	166	159	166	110	108	4.9	6.6	6.8	0.195
wood1p	171	165	171	61	52	13.9	22	22.3	0.227
bandm	180	177	180	93	81	5.9	8.7	9.2	0.147
scrs8	181	168	181	123	112	4.9	5.5	6	0.112
ship04s	213	197	213	192	191	2.6	2.8	2.8	0.017
scagr25	221	221	221	91	89	7.3	16.1	16.4	0.331
scfxm1	242	236	242	160	145	4.3	5.8	6.1	0.07
stair	246	246	246	217	217	13.8	14.6	14.6	0.374
shell	252	238	252	249	249	1.9	1.9	1.9	0.007
standata	258	211	258	156	156	1.9	2.6	2.6	0.012
sctap1	269	202	269	144	144	2.3	3	3	0.019
agg2	280	266	280	123	123	5.2	10.4	10.4	0.102
agg3	282	265	282	116	116	5.1	10.1	10.1	0.103
boeing1	284	284	284	174	174	4.8	7.3	7.3	0.068
ship08s	284	233	284	252	252	2.4	2.6	2.6	0.011
grow15	300	300	300	102	102	12.2	4.7	4.7	0.065

Table 1 continued

Name	$\|\mathcal{V}_0\|$	$\|\mathcal{V}_1\|$	$\|\mathcal{E}_0\|$	$\|\mathcal{E}_1\|$	$\|\mathcal{E}_2\|$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
fffff800	306	274	306	214	170	4.5	5.9	7.1	0.083
etamacro	307	302	307	220	220	3.2	4.1	4.1	0.031
ship04l	313	305	313	282	282	2.7	2.9	2.9	0.012
gfrdpnc	322	278	322	320	320	1.9	1.9	1.9	0.006
ship12s	344	304	344	287	286	2.4	2.7	2.7	0.01
finnis	350	279	350	249	248	2.3	2.9	2.9	0.015
pilot4	352	349	352	268	262	8.9	11.1	11.3	0.092
standmps	360	282	360	295	295	2.3	2.6	2.6	0.009
degen2	382	376	382	268	268	6.3	8.3	8.3	0.078
scsd8	397	397	397	394	394	2.8	2.8	2.8	0.013
grow22	440	440	440	161	161	11.9	4.5	4.5	0.044
bnl1	448	438	448	317	317	3.6	4.7	4.7	0.02
czprob	475	464	475	475	475	1.9	1.9	1.9	0.004
scfxm2	485	474	485	325	298	4.4	5.9	6.1	0.036
perold	500	499	500	425	413	6.5	7.2	7.3	0.046

Table 2 Instance information of `dimacs` testset

Name	$\|\mathcal{V}_0\|$	$\|\mathcal{V}_1\|$	$\|\mathcal{E}_0\|$	$\|\mathcal{E}_1\|$	$\|\mathcal{E}_2\|$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
myciel4	23	23	71	71	71	2	2	2	0.28
queen5_5	25	25	160	160	34	2	2	3.9	0.533
queen6_6	36	36	290	290	65	2	2	3.8	0.46
myciel5	47	47	236	236	236	2	2	2	0.218
queen7_7	49	49	476	476	91	2	2	4	0.404
queen8_8	64	64	728	728	126	2	2	4.1	0.361
huck	74	74	301	301	34	2	2	4.5	0.111
jean	80	77	254	254	55	2	2	3.5	0.08
queen9_9	81	81	1056	1056	175	2	2	4.2	0.325
david	87	87	406	406	65	2	2	4.3	0.108
myciel6	95	95	755	755	755	2	2	2	0.169
queen8_12	96	96	1368	1368	218	2	2	4.2	0.3
queen10_10	100	100	1470	1470	218	1.9	1.9	4.3	0.296
games120	120	120	638	638	202	2	2	2.9	0.089
queen11_11	121	121	1980	1980	255	2	2	4.6	0.272
miles1000	128	128	3216	3216	96	2	2	14.6	0.395
miles1500	128	128	5198	5198	60	2	2	25.4	0.639
miles250	128	125	387	387	89	2	2	3.6	0.047
miles500	128	128	1170	1170	104	2	2	6.4	0.143
miles750	128	128	2113	2113	112	2	2	9.3	0.259

Table 2 continued

Name	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{E}_0\rvert$	$\lvert\mathcal{E}_1\rvert$	$\lvert\mathcal{E}_2\rvert$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
anna	138	138	493	493	115	2	2	3.6	0.052
queen12_12	144	144	2596	2596	315	2	2	4.6	0.252
queen13_13	169	169	3328	3328	345	2	2	4.9	0.234
mulsol.i.3	184	174	3916	3916	177	2	2	13.3	0.232
mulsol.i.4	185	175	3946	3946	177	2	2	13.3	0.231
mulsol.i.5	186	176	3973	3973	181	2	2	13.4	0.23
mulsol.i.2	188	173	3885	3885	175	1.9	1.9	13.3	0.221
myciel7	191	191	2360	2360	2360	2	2	2	0.13
queen14_14	196	196	4186	4186	419	2	2	4.9	0.219
mulsol.i.1	197	138	3925	3925	109	1.9	1.9	17.3	0.203
zeroin.i.3	206	157	3540	3540	173	2	2	13	0.167
zeroin.i.1	211	126	4100	4100	99	2	2	21.5	0.185
zeroin.i.2	211	157	3541	3541	172	2	2	12.9	0.159
queen15_15	225	225	5180	5180	433	2	2	5.3	0.205
queen16_16	256	256	6320	6320	450	2	2	5.7	0.193
school1_nsh	352	352	14612	14612	1204	1.9	1.9	8	0.236
school1	385	385	19095	19095	1485	2	2	8.6	0.258
fpsol2.i.3	425	363	8688	8688	404	2	2	13.4	0.096
fpsol2.i.2	451	363	8691	8691	398	2	2	13.4	0.085
fpsol2.i.1	496	269	11654	11654	205	2	2	25.2	0.094

Table 3 Instance information of `miplib` testset

Name	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{E}_0\rvert$	$\lvert\mathcal{E}_1\rvert$	$\lvert\mathcal{E}_2\rvert$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
b-ball	19	19	89	89	8	2.1	2.1	12	0.836
eil33.2	32	32	4484	2558	1	9.8	10.4	32	1
neos-911880	83	83	888	840	840	2.8	3	3	0.5
harp2	92	92	2967	999	999	0.6	2	2	0.238
eilB101	100	100	2718	2284	1391	8.8	9.2	9.6	0.792
m100n500k4r1	100	100	500	500	498	4	4	4	0.454
mik.250-1-100.1	100	100	251	1	1	21.1	100	100	1
ns1766074	110	110	100	100	100	4.5	4.5	4.5	0.292
neos858960	128	128	160	80	80	17.3	17.3	17.3	0.298
pg	135	135	2690	2500	2500	2	2.1	2.1	0.305
dfn-gwin-UUM	156	156	937	469	469	2.8	3	3	0.116
noswot	172	172	121	50	50	5.6	8.9	8.9	0.098
pg5_34	225	225	2600	2500	2500	2.9	3	3	0.202
50v-10	233	233	2013	183	183	1.3	3	3	0.02
neos-1228986	241	241	245	160	160	5	7.7	7.7	0.1
k16x240	256	256	480	240	240	2	3	3	0.018

Table 3 continued

Name	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{E}_0\rvert$	$\lvert\mathcal{E}_1\rvert$	$\lvert\mathcal{E}_2\rvert$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
csched007	271	271	1656	1653	1565	3.4	3.4	3.5	0.148
csched008	271	271	1480	1479	1397	3.4	3.4	3.5	0.135
csched010	272	272	1654	1654	1505	3.4	3.4	3.6	0.144
ran14x18	284	284	504	252	252	2	3	3	0.018
ran16x16	288	288	512	256	256	2	3	3	0.018
probportfolio	302	302	320	301	2	20.6	2.9	301	0.999
neos-1440225	328	328	1285	1277	512	10.9	11	14.1	0.16
timtab1	332	332	214	53	50	5.9	18.1	19	0.228
gmu-35-40	357	357	1202	265	239	3.4	8	7.9	0.054
gmu-35-50	358	358	1917	373	276	3.8	9.7	10	0.069
go19	361	361	441	361	361	3.9	4.7	4.7	0.03
glass4	392	392	322	317	91	5.5	5.6	17.6	0.323
neos788725	433	433	352	352	352	13.9	13.9	13.9	0.074
ran14x18.disj-8	447	447	504	502	502	20.3	20.4	20.4	0.159
p80x400b	474	474	798	396	396	1.9	3	3	0.008
neos-777800	475	475	6400	6400	6400	4.9	4.9	4.9	0.345
swath	482	482	6804	6260	6239	3.7	4	4	0.19
neos-1426635	486	486	510	320	320	5	7.9	7.9	0.052
30n20b8	490	490	18375	1092	208	2.6	12.8	19.4	0.235
neos15	492	492	677	443	438	2.4	3.2	3.2	0.013
ger50_17_trans	498	498	22414	8240	4459	7.6	7.2	6.7	0.113

Table 4 Instance information of `random` testset

Name	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{E}_0\rvert$	$\lvert\mathcal{E}_1\rvert$	$\lvert\mathcal{E}_2\rvert$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
grp1_1	68	67	55	54	54	3.1	3.1	3.1	0.083
grp1_2	68	64	60	59	59	2.7	2.7	2.7	0.074
grp1_3	58	57	73	71	71	2.9	2.9	2.9	0.131
grp1_4	60	58	58	58	56	3	3	3	0.105
grp1_5	75	67	52	51	48	3.1	3.1	3.2	0.066
grp2_1	75	70	62	62	62	3.2	3.2	3.2	0.082
grp2_2	95	78	50	48	48	3.1	3.2	3.2	0.04
grp2_3	87	76	63	62	62	3	3	3	0.058
grp2_4	98	85	68	66	66	2.9	2.9	2.9	0.042
grp2_5	93	77	50	49	49	2.9	2.9	2.9	0.037
grp3_1	102	90	65	64	64	3.1	3.1	3.1	0.045
grp3_2	108	96	71	70	70	2.9	2.9	2.9	0.037
grp3_3	122	101	69	67	67	2.9	2.9	2.9	0.028
grp3_4	104	89	61	61	60	3	3	3.1	0.04
grp3_5	107	89	67	67	67	2.9	2.9	2.9	0.038

Table 4 continued

Name	$\|\mathcal{V}_0\|$	$\|\mathcal{V}_1\|$	$\|\mathcal{E}_0\|$	$\|\mathcal{E}_1\|$	$\|\mathcal{E}_2\|$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
grp4_1	142	103	66	66	65	3	3	3	0.022
grp4_2	125	106	72	72	72	3	3	3	0.031
grp4_3	135	96	55	54	54	3	3	3	0.02
grp4_4	128	96	59	59	58	2.7	2.7	2.8	0.019
grp4_5	126	100	63	63	61	2.9	2.9	2.9	0.024
grp5_1	173	125	75	75	74	2.8	2.8	2.8	0.014
grp5_2	161	99	50	50	50	2.8	2.8	2.8	0.012
grp5_3	158	105	64	63	63	2.9	2.9	2.9	0.015
grp5_4	159	114	59	59	59	2.9	2.9	2.9	0.015
grp5_5	158	110	59	58	58	3	3	3	0.016
grp6_1	69	68	91	88	85	3	3	3.1	0.124
grp6_2	74	71	79	78	76	3.1	3.1	3.1	0.098
grp6_3	50	50	96	89	81	2.8	2.9	3.1	0.204
grp6_4	52	52	89	85	84	3	3	3	0.207
grp6_5	63	63	95	89	86	3	3.1	3.1	0.152
grp7_1	96	85	77	75	75	2.8	2.9	2.9	0.048
grp7_2	77	74	95	89	87	2.8	2.9	2.9	0.092
grp7_3	77	75	98	97	92	3.1	3.1	3.3	0.119
grp7_4	87	86	98	95	94	2.9	3	3	0.084
grp7_5	78	77	90	90	87	3	3	3	0.096
grp8_1	115	108	94	93	93	3	3	3	0.049
grp8_2	121	112	98	95	95	2.9	2.9	2.9	0.041
grp8_3	118	106	95	94	94	3	3	3	0.043
grp8_4	122	108	86	86	86	3	3	3	0.04
grp8_5	108	101	86	85	85	3.1	3.1	3.1	0.052
grp9_1	136	123	98	96	95	3	3	3.1	0.037
grp9_2	143	110	78	77	77	2.8	2.9	2.9	0.023
grp9_3	146	129	98	98	97	3	3	3.1	0.032
grp9_4	139	128	89	89	89	3.1	3.1	3.1	0.034
grp9_5	138	118	94	93	93	2.9	2.9	2.9	0.031
grp10_1	168	139	98	94	94	3	3	3	0.022
grp10_2	169	141	100	98	97	3	3.1	3.1	0.024
grp10_3	161	134	90	90	90	3	3	3	0.022
grp10_4	157	126	79	78	78	3.1	3.1	3.1	0.022
grp10_5	164	123	79	79	79	3	3	3	0.019

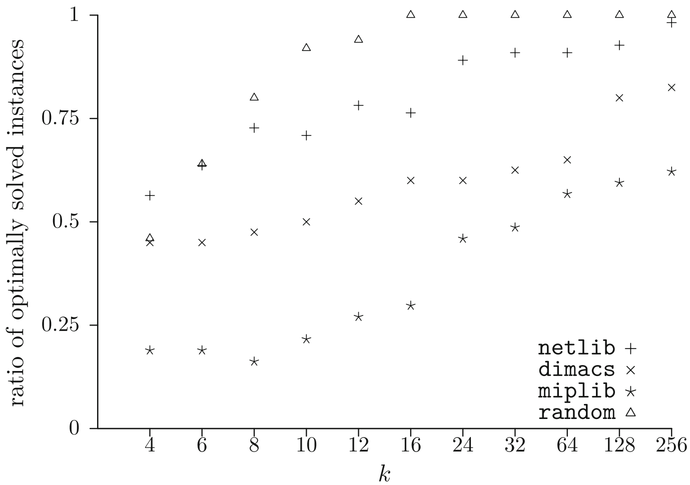


Fig. 3 Ratio of optimally solved instances in each testset for different values of k

Testset	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{E}_0\rvert$	$\lvert\mathcal{E}_1\rvert$	$\lvert\mathcal{E}_2\rvert$	$D_0^{\mathcal{E}}$	$D_1^{\mathcal{E}}$	$D_2^{\mathcal{E}}$	d
miplib	277.7	277.7	2421.2	1237.8	968.6	6.0	9.1	18.6	0.264
netlib	218.1	206.3	218.1	150.7	146.4	5.1	6.7	7.1	0.136
random	112.3	94.5	76.3	75.0	74.1	3.0	3.0	3.0	0.056
dimacs	168.4	151.5	3507.6	3507.6	311.4	2.0	2.0	8.3	0.236

3.4 Fixed maximum number of shores

When reporting on a fixed maximum number k of shores we set capacity $u := \lceil \frac{|\mathcal{V}|}{k} \rceil$, see Fig. 3. With increasing k the ratio of optimally solved instances also increases for every testset. Furthermore the order of the testsets according to the ratio of optimally solved instances is the same for every k. Considering this as measure for difficulty (according to *base*) we get in increasing order of difficulty: `random`, `netlib`, `dimacs`, `miplib`. Note that for $k \leq 12$ there are instances in every testset that could not be solved optimally.

Figure 4 gives more detail on solution quality. We plot the ratio of instances that were solved with optimality gap worse than σ for different values of k. On the one hand, the ratio of instances with optimality gap $\sigma \geq 50\%$ is varying not much but on the other hand, the ratio of instances with gap $\sigma \leq 25\%$ decreases with increasing k. This shows that finding some solution is generally not so hard, but closing the gap is easier for larger k. About 5% of the instances could not be solved with a gap better than 200% for every k.

In Fig. 5 we display the dependence of the clique graph density d. The instances are grouped according to their density. The performance for instances with $d \leq 0.05$ seems to be most k-sensitive in the sense that these instances are the hardest to solve

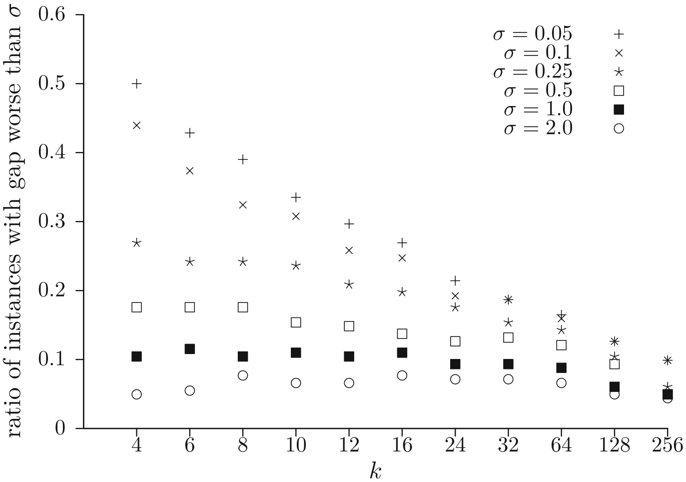


Fig. 4 Ratio of instances solved with optimality gap worse than σ for different values of σ and k

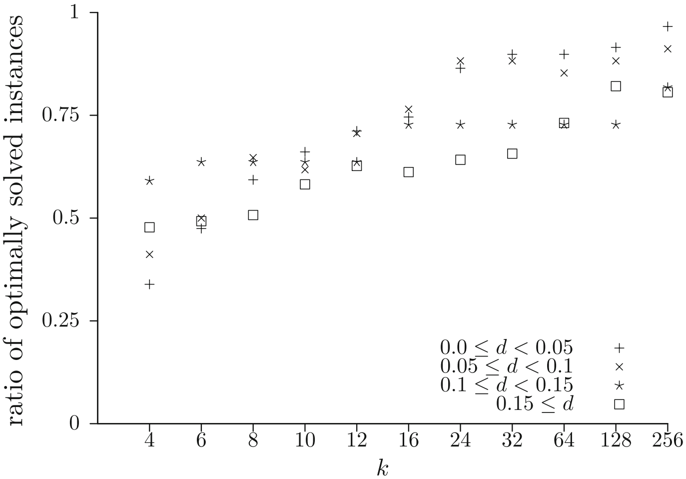


Fig. 5 Ratio of optimally solved instances for different density d and k

for $k = 4$ and the easiest to solve for $k = 256$. In contrast the difficulty of instances with $d > 0.15$ seems to be least k-sensitive in the sense that the ratio of optimally solved instances is changing the least for increasing k.

Aggregated report of results for all instances In the following we want to compare the performance of *base* with the performance of CPLEX 12.6 working on the original formulation (P). We call this algorithm *cplex*. In order to compare the performance on all instances for every k we use performance profiles [14], displayed in Fig. 6. Algorithms *cplex* and *base* are represented by the dashed and the solid line, respectively. We realize that for $k < 8$ algorithm *cplex* performs better while for $k = 8$ both algorithms have a similar performance. For $k > 8$, however, algorithm *base* outperforms

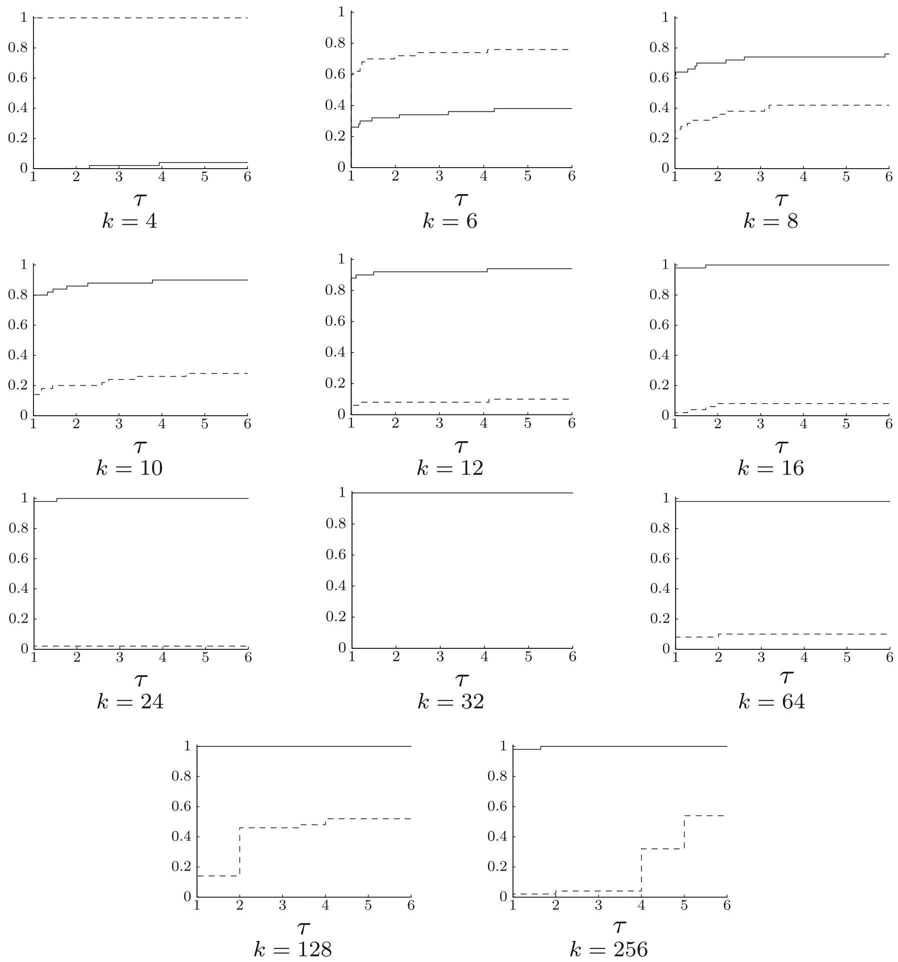


Fig. 6 Performance difference between branch-and-price and branch-and-cut, over all instances: *base* (solid) on model (M) versus *cplex* (dashed) on model (P)

algorithm *cplex*. Furthermore we observe for increasing k that instances overall get easier to solve for *base*. On the contrary for algorithm *cplex* for increasing k instances get harder to solve (up to $k = 32$ then easier again). The results for each testset are similar to the aggregated ones. Performance profiles for each testset can be found in "Appendix A.4" (online only). We want to point out that Borndörfer et al. [9] also solved instances from the `netlib` testset for $k = 4$ but since *base* ist outperformed by *cplex* for $k = 4$, a comparison between *base* and the algorithm in [9] is obsolete.

3.5 Arbitrarily many shores

In the next experiment we do not bound k and adapt u accordingly, but consider the reverse setting: set a specific capacity and allow arbitrarily many shores. We report on

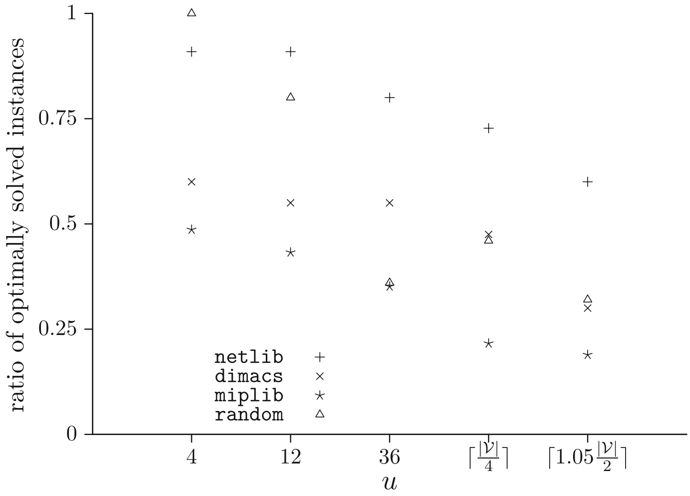


Fig. 7 Ratio of optimally solved instances by algorithm *base* for $k = \infty$ and varying capacity u

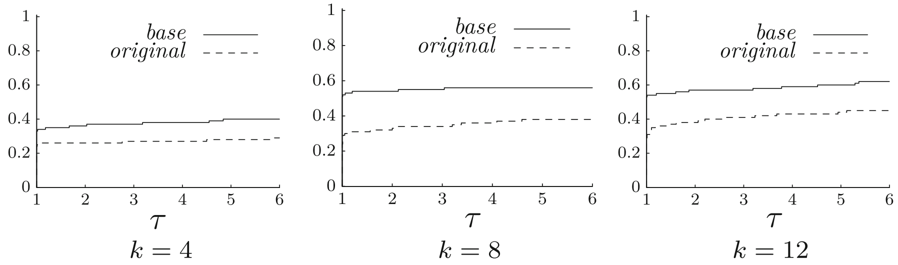


Fig. 8 Impact of hypergraph preprocessing as described in Sect. 2.4: algorithm *base* (with phase 2 preprocessing) versus *original* (only phase 1 preprocessing) on all instances for $k \in \{4, 8, 12\}$

$u \in \{4, 12, 36, \lceil\frac{|\mathcal{V}|}{4}\rceil, \lceil 1.05\frac{|\mathcal{V}|}{2}\rceil\}$. In Fig. 7 the success rate for the different testsets and values of u is displayed. For $u \in \{4, 12\}$ all `random` and a large share of `netlib` instances can be solved optimally. We further find that for all testsets with increasing u less instances can be solved optimally. This is to be expected as pricing problems get combinatorially richer. One might expect the results for $k = 4$ and $k = \infty$ with $u = \lceil\frac{|\mathcal{V}|}{4}\rceil$ to be similar, but in fact the latter one is much better. A possible reason is the absence of constraint (M.2) for $k = \infty$.

Oosten et al. [25] tested their approach for a subset of the `netlib` instances for $u = \lceil\frac{|\mathcal{V}|}{4}\rceil$. Algorithm *base* solves all instances they tested within 25 s in total (including the three instances *scsd1*, *beaconfd*, and *share2b* that could not be solved within the time limit of 60 min in [25]).

3.6 Strength of formulations

Phase 2 of hypergraph preprocessing may find an alternative hypergraph with the same clique graph as the input hypergraph (Sect. 2.4). The corresponding formulations (M)

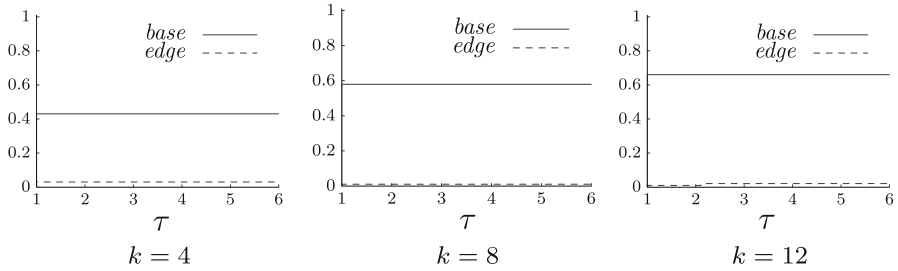


Fig. 9 Impact of working with a hypergraph based formulation (algorithm *base*) versus an edge based formulation using clique graphs (algorithm *edge*) on all instances for $k \in \{4, 8, 12\}$

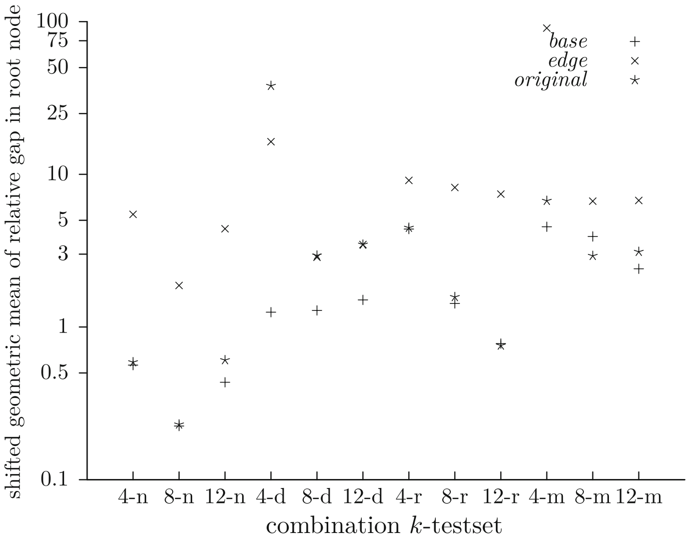


Fig. 10 Shifted (by 1) geometric mean of the integrality gap in the root node (in percent, for instances with solved root node) for combinations of k and testset (abbreviated by their first letter)

have the same integer solutions but the respective LP relaxations may differ in strength. To evaluate the impact of that reformulation we compare against the case with only phase 1 preprocessing enabled (called algorithm *original*). As a third reference we also compare against using the corresponding clique graph in formulation (M) (called algorithm *edge*).

For a fixed maximum number of shores the particularly interesting (since difficult) shore numbers are $k \in \{4, 8, 12\}$. The performance profiles in Figs. 8 and 9 reveal that for these k algorithm *base* outperforms *original* and massively outclasses *edge*. More specifically, algorithm *base* solves between 10 and 20 percent more instances for $k \in \{4, 8, 12\}$ than algorithm *original*.

In order to get an idea of the strength of the formulations we look at the integrality gap at the root node (if it is solved for this instance). Figure 10 shows the shifted (by 1) geometric mean for $k \in \{4, 8, 12\}$ and each instance set. Algorithm *base* delivers the smallest shifted geometric mean of the integrality gap in almost every case.

3.7 Feature impact

Presolving of pricing problem The pricing problem is preprocessed as described in Sect. 2.3.3. The influence of the preprocessing is displayed in the three performance profiles in Fig. 11. As one can see the performance is slightly improved in particular for $k = 8$.

Exchange vectors We tested the model modifications discussed in Sect. 2.6. Unfortunately, it turns out that the described exchange vectors (which in fact are rather removal vectors) are not involved enough and lead to a slightly worse performance. Therefore we decided to disable them by default, see Fig. 12.

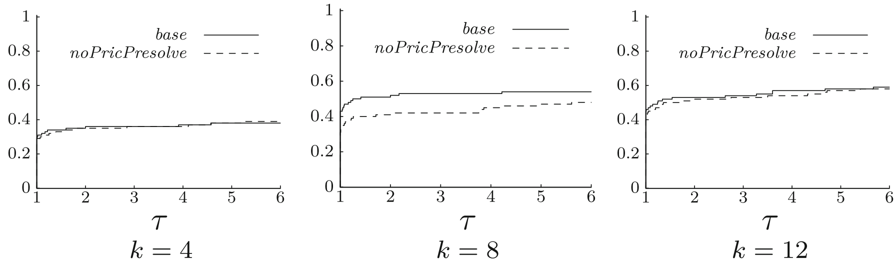


Fig. 11 Performance profiles *base* versus *base* with unpresolved pricing problem on all instances for $k \in \{4, 8, 12\}$

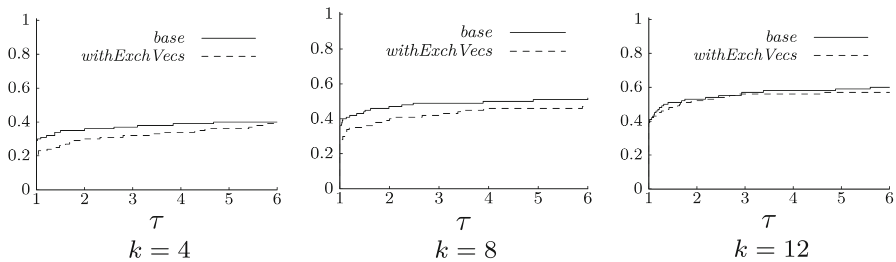


Fig. 12 Performance profiles *base* versus *base* with exchange vectors on all instances for $k \in \{4, 8, 12\}$

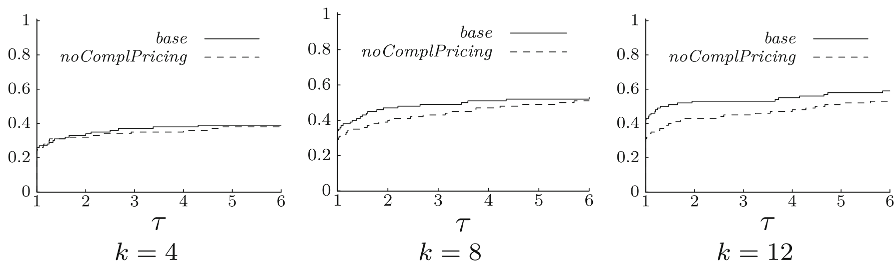


Fig. 13 Performance profiles *base* versus *base* without complementary pricing on all instances for $k \in \{4, 8, 12\}$

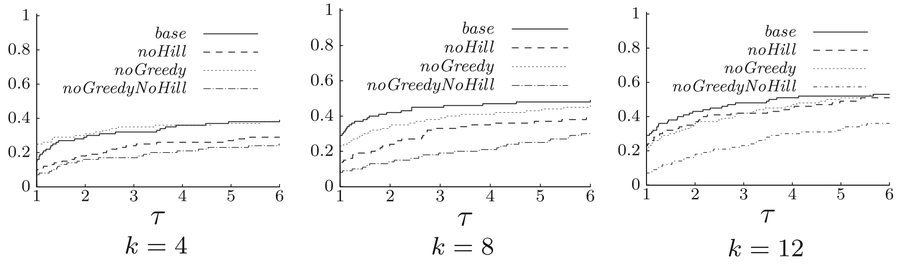


Fig. 14 Performance profiles *base* versus *base* without some heuristic pricing algorithm(s) on all instances for $k \in \{4, 8, 12\}$

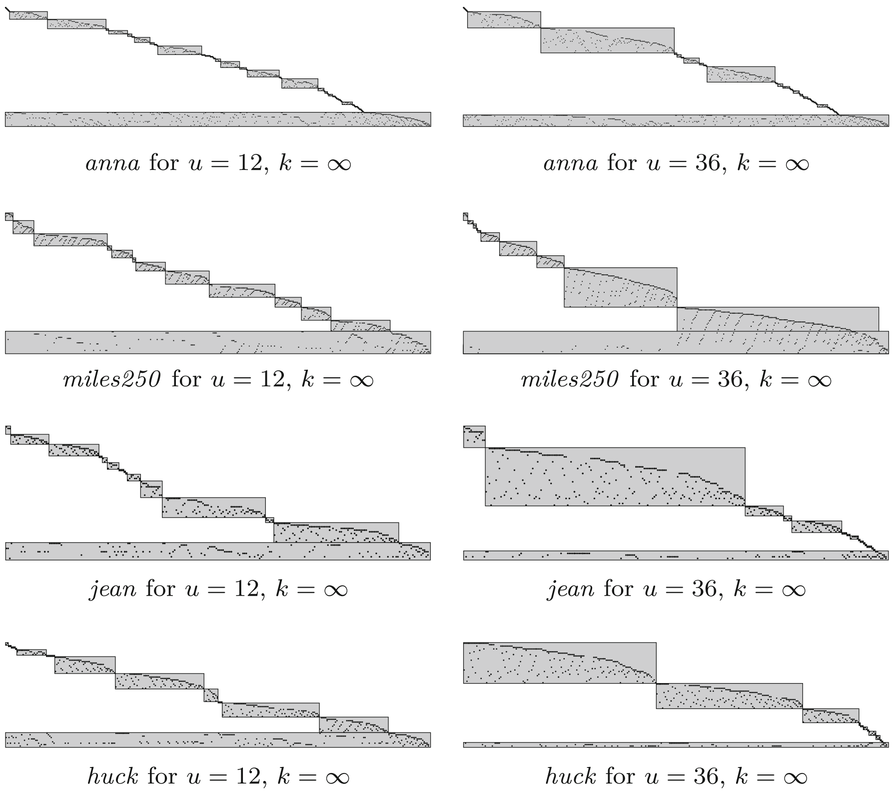


Fig. 15 Examples of (optimally) decomposed matrices corresponding to graphs of `dimacs` instances

Complementary pricing In the performance profiles in Fig. 13 one can see the influence of the complementary pricing described in Sect. 2.3.7. For $k \in \{4, 8, 12\}$ *base* outperforms the variant with disabled complementary pricing.

Pricing algorithms The performance profiles in Fig. 14 give an overview over the performance when disabling one or both of the heuristic pricing approaches for $k \in \{4, 8, 12\}$ over all instances. It turns out that enabling both heuristic pricing algorithms overall outperforms the other approaches.

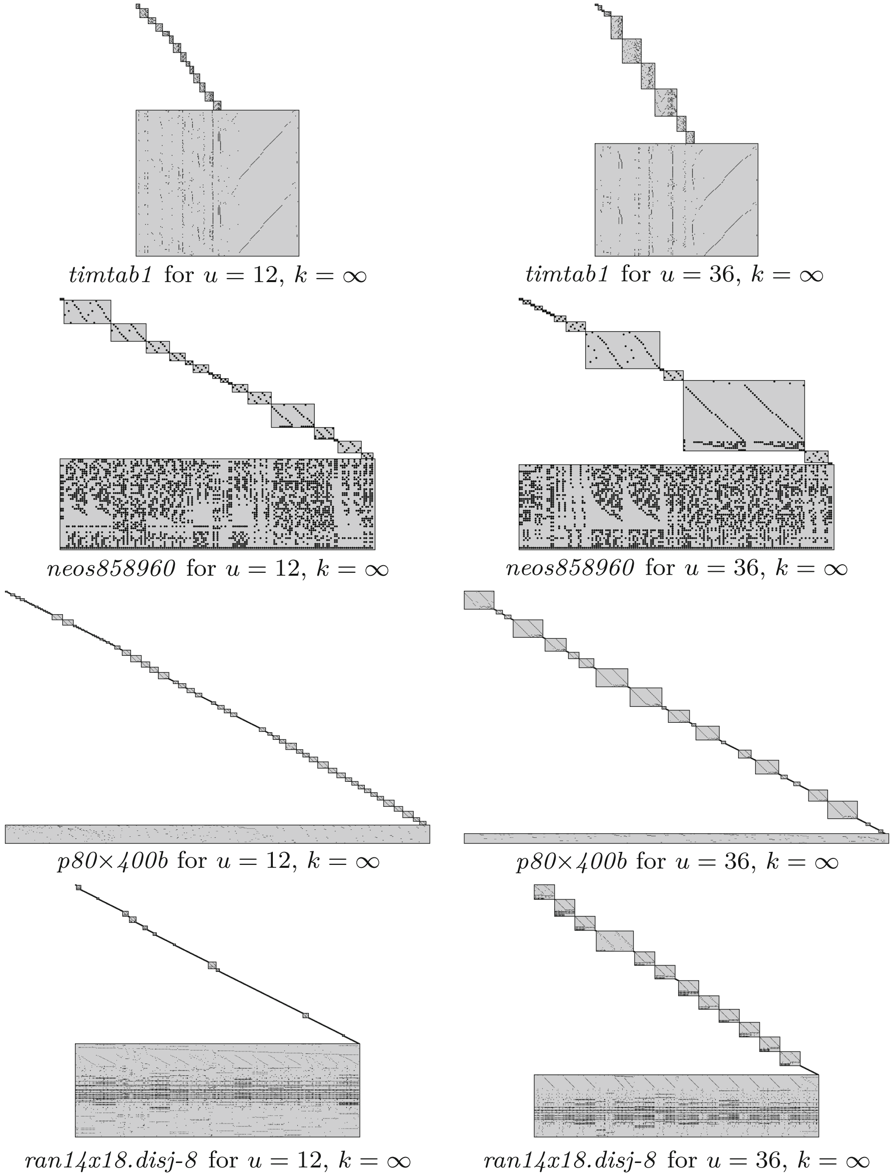


Fig. 16 Examples of (optimally) decomposed coefficent matrices of presolved MIPLIB2010 mixed integer programs

3.8 Optimal decompositions of matrices

Examples for coefficient matrices of `dimacs`, `miplib`, and `netlib` instances into single-bordered block-diagonal form with minimum cardinality border are shown in Figs. 15, 16 and 17.

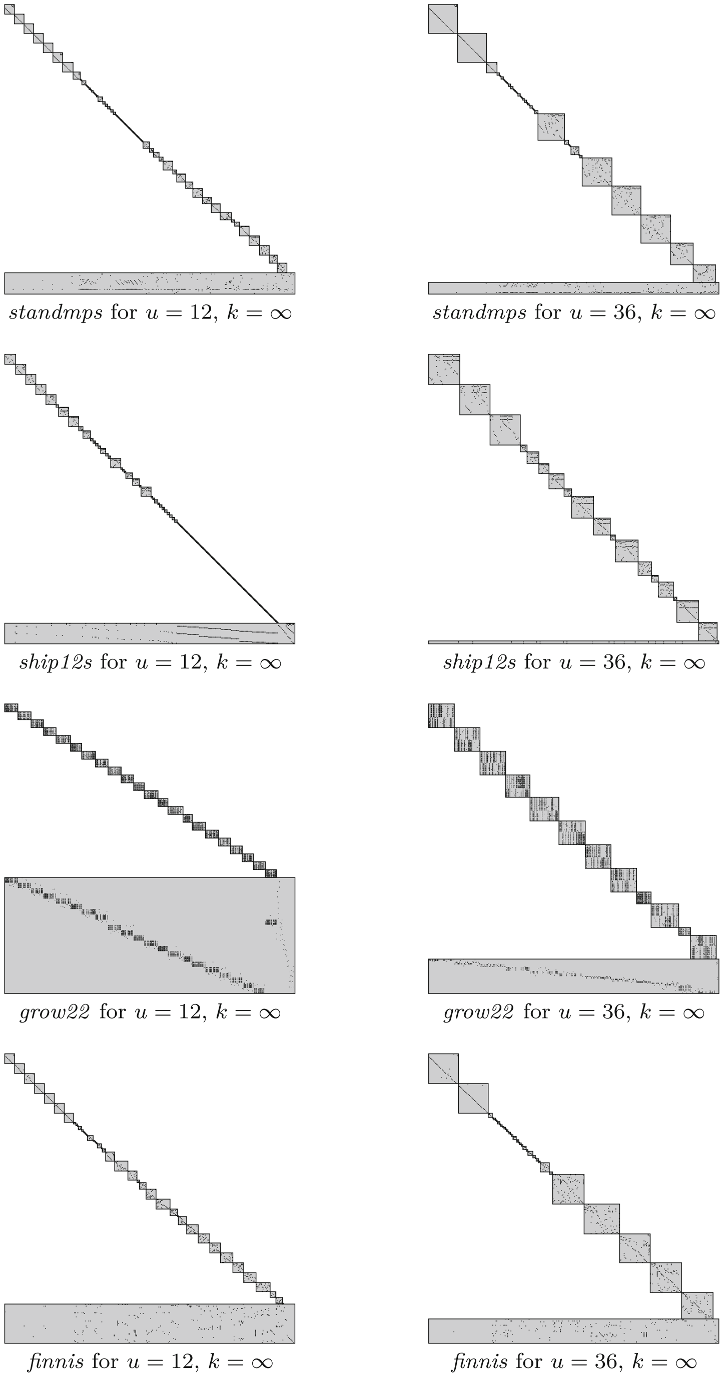


Fig. 17 Examples of (optimally) decomposed coefficent matrices of `netlib`instances

4 Conclusion

In this paper we studied the capacitated vertex separator problem on hypergraphs (CHVS). We presented a branch-and-price approach for fixed and arbitrary number of shores, and reported and compared our results to the existing approaches whenever possible. It is the first successful algorithm for the CHVS for a large number of shores $k > 8$ that is especially interesting for the matrix decomposition application. It uses state-of-art methods that highlight the impact of exploiting problem structure, e.g., in preprocessing. We tested on a large set of instances from several applications. The complexity of the pricing problem, which has an interesting application on its own, is studied and we give furthermore three approaches to solve it. The non-trivial branching scheme uses results on the integer round-up property for Bin Packing.

More than 20 years have passed since the first presentation of an exact algorithm for the CHVS [9]. At the time, elaborate valid inequalities were needed to strengthen the LP relaxation. Such cutting planes are part of generic solvers nowadays and make them successful tools as can be seen in our experiments for fixed $k \leq 12$. For larger k, our exponential-size reformulation, and the resulting branch-and-price algorithm, still significantly outperform the standard state-of-the-art solver. We may hope that 20 years from now, such reformulations be part of generic solvers as well.

References

1. Aykanat, C., Pinar, A., Çatalyürek, Ü.V.: Permuting sparse rectangular matrices into block-diagonal form. SIAM J. Sci. Comput. **25**(6), 1860–1879 (2004)
2. Bagnall, A., Rayward-Smith, V., Whittley, I.: The next release problem. Inf. Softw. Technol. **43**(14), 883–890 (2001). https://doi.org/10.1016/S0950-5849(01)00194-X
3. Balas, E., de Souza, C.C.: The vertex separator problem: a polyhedral investigation. Math. Program. **103**(3), 583–608 (2005). https://doi.org/10.1007/s10107-005-0574-7
4. Barahona, F., Jensen, D.: Plant location with minimum inventory. Math. Program. **83**(1), 101–111 (1998). https://doi.org/10.1007/BF02680552
5. Baum, S., Trotter Jr., L.: Integer rounding for polymatroid and branching optimization problems. SIAM J Algebra Discrete Methods **2**(4), 416–425 (1981)
6. Ben-Ameur, W., Mohamed-Sidi, M.A., Neto, J.: The k-separator problem: polyhedra, complexity and approximation results. J. Combinatorial Optim. **29**, 1–32 (2015)
7. Ben Amor, H., Desrosiers, J., Valério de Carvalho, J.: Dual-optimal inequalities for stabilized column generation. Oper. Res. **54**(3), 454–463 (2006). https://doi.org/10.1287/opre.1060.0278
8. Bergner, M., Caprara, A., Ceselli, A., Furini, F., Lübbecke, M., Malaguti, E., Traversi, E.: Automatic Dantzig-Wolfe reformulation of mixed integer programs. Math. Prog. **149**(1–2), 391–424 (2015). https://doi.org/10.1007/s10107-014-0761-5
9. Borndörfer, R., Ferreira, C.E., Martin, A.: Decomposing matrices into blocks. SIAM J. Optim. **9**(1), 236–269 (1998)
10. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Inf. Process. Lett. **42**(3), 153–159 (1992)
11. Cornaz, D., Furini, F., Lacroix, M., Malaguti, E., Mahjoub, A.R., Martin, S.: Mathematical formulations for the balanced vertex k-separator problem. In: Control, Decision and Information Technologies (CoDIT), 2014 International Conference on, pp. 176–181. IEEE (2014)
12. Cornaz, D., Furini, F., Lacroix, M., Malaguti, E., Mahjoub, A.R., Martin, S.: The vertex k-cut problem. Tech. rep., Optimization Online (2017)
13. de Souza, C., Balas, E.: The vertex separator problem: algorithms and computations. Math. Program. **103**(3), 609–631 (2005). https://doi.org/10.1007/s10107-005-0573-8

14. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
15. Evrendilek, C.: Vertex separators for partitioning a graph. Sensors **8**(2), 635–657 (2008)
16. Ghoniem, A., Sherali, H.D.: Complementary column generation and bounding approaches for set partitioning formulations. Optim. Lett. **3**(1), 123–136 (2009)
17. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. Oper. Res. **9**(6), 849–859 (1961)
18. Kartak, V.M., Ripatti, A.V., Scheithauer, G., Kurz, S.: Minimal proper non-irup instances of the one-dimensional cutting stock problem. Discrete Appl. Math. **187**, 120–129 (2015)
19. Kayaaslan, E., Pinar, A., Çatalyürek, Ümit, Aykanat, C.: Partitioning hypergraphs in scientific computing applications through vertex separators on graphs. SIAM J. Sci. Comput. **34**(2), A970–A992 (2012). https://doi.org/10.1137/100810022
20. Kellerman, E.: Determination of keyword conflict. IBM Tech. Discl. Bull. **16**(2), 544–546 (1973)
21. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Math. Program. Comput. **3**(2), 103–163 (2011). https://doi.org/10.1007/s12532-011-0025-9
22. Kou, L.T., Stockmeyer, L.J., Wong, C.K.: Covering edges by cliques with regard to keyword conflicts and intersection graphs. Commun. ACM **21**(2), 135–139 (1978). https://doi.org/10.1145/359340.359346
23. Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Oper. Res. **53**(6), 1007–1023 (2005)
24. Marcotte, O.: An instance of the cutting stock problem for which the rounding property does not hold. Oper. Res. Lett. **4**(5), 239–243 (1986)
25. Oosten, M., Rutten, J.H.G.C., Spieksma, F.C.R.: Disconnecting graphs by removing vertices: a polyhedral approach. Stat. Neerl. **61**(1), 35–60 (2007). https://doi.org/10.1111/j.1467-9574.2007.00350.x
26. Ryan, D.M., Foster, B.A.: An integer programming approach to scheduling. Comput. Sched. Public Transp. Urban Passeng. Veh. Crew Sched. 269–280 (1981)