A Branch-Price-and-Cut Algorithm for Packing Cuts in Undirected Graphs

MARTIN BERGNER, MARCO E. LÜBBECKE, and JONAS T. WITT, RWTH Aachen University

The cut packing problem in an undirected graph is to find a largest cardinality collection of pairwise edge-disjoint cuts. We provide the first experimental study of this NP-hard problem that is interesting from a pure theorist's viewpoint as well as from the standpoint of scientific applications (e.g., in bioinformatics and network reliability). So far it could not be solved exactly. We propose a branch-price-and-cut algorithm to optimally solve instances from various graph classes, random and from the literature, with up to several hundred vertices. In particular, we investigate how complexity results match computational experience and how combinatorial properties help improve the algorithm's performance.

CCS Concepts: • **Mathematics of computing** → **Integer programming**; *Combinatorial optimization*; *Graph algorithms;*

Additional Key Words and Phrases: Column generation, branch-price-and-cut, packing problem

ACM Reference Format:
Martin Bergner, Marco E. Lübbecke, and Jonas T. Witt. 2016. A branch-price-and-cut algorithm for packing cuts in undirected graphs. ACM J. Exp. Algor. 21, 1, Article 1.2 (January 2016), 16 pages.
DOI: http://dx.doi.org/10.1145/2851492

1. INTRODUCTION

Given an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, every $S \subseteq V$ induces a *cut* $\delta(S) = \{ij \in E \mid i \in S, j \notin S\}$. We call S and $V \backslash S$ the *shores* of cut $\delta(S)$. We assume G to be connected and S a nontrivial subset; thus, $\delta(S) \neq \emptyset$. Two cuts $\delta_1 \subseteq E$ and $\delta_2 \subseteq E$ are *disjoint* if $\delta_1 \cap \delta_2 = \emptyset$. A *cut packing* is a set of pairwise disjoint cuts and the *cut packing problem* is to find a cut packing of largest cardinality. The maximum is called the cut packing number $\gamma(G)$.

In combinatorial optimization, cut packing is known for its role in duality theorems [Fulkerson 1971; Robacker 1956]. It is NP-hard in general [Colbourn 1987] and even in planar graphs [Caprara et al. 2003]. However, it is polynomial time solvable in chordal or bipartite graphs [Colbourn 1987] or when packing *s-t* cuts [Colbourn 1988; Robacker 1956]. Interestingly, packing directed cuts in digraphs is polynomial time solvable as well [Lucchesi and Younger 1978]. Cut packing is closely related to other combinatorial optimization problems like cycle packing [Caprara et al. 2003] and independent set [Colbourn 1987]: an independent set $I \subseteq V$ immediately translates to a (particular) cut packing $\{\delta(\{v\}) : v \in I\}$. The latter provides a strong

A preliminary version of this work appeared as Bergner et al. [2014].
Supported by the German Research Foundation (DFG) as part of the Priority Program "Algorithm Engineering" under grant no. LU770/4-1 and LU770/4-2.
Authors' addresses: M. Bergner, M. E. Lübbecke, and J. T. Witt, RWTH Aachen University, Operations Research, Kackertstr. 7, 52072 Aachen, Germany; emails: {martin.bergner, marco.luebbecke, jonas.witt}@rwth-aachen.de.

DOI: http://dx.doi.org/10.1145/2851492

inapproximability result [Caprara et al. 2004]: for the stability number $\alpha(G)$ (the cardinality of a maximum independent set in G), it holds that $\alpha(G) \leq \gamma(G) \leq 2\alpha(G)-1$. Again, special graph classes allow better approximation guarantees [Caprara et al. 2004]. The parameterized version of the cut packing problem is W[1]-hard by a reduction from a parameterized independent set; see Section 2.3 of this article. Cut packing and variants have applications in bioinformatics [Caprara et al. 2004] and network reliability [Colbourn 1987]. Colbourn [1988] mentions that the NP-hardness of several edge packing problems "limits their applicability" for obtaining bounds for network reliability. We mitigate this argument by presenting an exact integer-programming-based approach to solve the cut packing problem for arbitrary graphs optimally.

Our Contribution. We are not aware of any experimental results, neither exact nor approximate, for cut packing; thus, we provide the first computational study of the problem. We propose a branch-price-and-cut algorithm to optimally solve instances from various graph classes, random and from the literature, with up to several hundred vertices. This success mainly builds on an easily computable combinatorial upper bound. In particular, we investigate how theoretical complexity and approximability results match with computational experience.

2. FORMULATIONS AND PROPERTIES

2.1. Compact Formulation

There are several known integer linear programming formulations for finding a single cut (of minimum capacity) in a graph, where the integer feasible solutions correspond to cuts in a graph. As trivially $\gamma(G) \leq n-1$, we replicate the constraints of such a formulation, one set for each *potential* cut in a packing, indexed by $c = 1, \ldots, n-1$. A cut $\delta_c = \{ij \in E \mid i \in S_c, j \notin S_c\}$ induced by S_c is represented by binary variables u_i^c, $i \in V$, taking value 0 when $i \in S_c$ and value 1 when $i \notin S_c$. With $V = \{1, \ldots, n\}$, we normalize vertex $1 \in S_c$, $c = 1, \ldots, n-1$, mildly reducing symmetry. Note that $\delta(S_c) = \delta(V \backslash S_c)$. The binary variables y_{ij}^c, $ij \in E$, take value 1 if and only if $ij \in \delta_c$, and the binary variables z^c are used to count the cuts in the packing. The cut packing problem can be formulated as

$$\max \quad \sum_{c=1}^{n-1} z^c$$

$$\text{s.t.} \quad u_i^c - u_j^c + y_{ij}^c \geq 0 \qquad \forall ij \in E,\ \forall c \in \{1, \ldots, n-1\} \qquad (1)$$

$$u_j^c - u_i^c + y_{ij}^c \geq 0 \qquad \forall ij \in E,\ \forall c \in \{1, \ldots, n-1\} \qquad (2)$$

$$y_{ij}^c - u_i^c - u_j^c \leq 0 \qquad \forall ij \in E,\ \forall c \in \{1, \ldots, n-1\} \qquad (3)$$

$$u_i^c + u_j^c + y_{ij}^c \leq 2 \qquad \forall ij \in E,\ \forall c \in \{1, \ldots, n-1\} \qquad (4)$$

$$u_1^c = 0 \qquad \forall c \in \{1, \ldots, n-1\} \qquad (5)$$

$$z^c \leq \sum_{ij \in E} y_{ij}^c \qquad \forall c \in \{1, \ldots, n-1\} \qquad (6)$$

$$\sum_{c=1}^{n-1} y_{ij}^c \leq 1 \qquad \forall ij \in E \qquad (7)$$

$$z^c,\ u_i^c,\ y_{ij}^c \in \{0, 1\} \qquad \forall ij \in E,\ \forall c \in \{1, \ldots, n-1\},\ \forall i \in V.$$

The metric inequalities in Equations (1) through (4) ensure compatibility between u and y variables for the cth potential cut. Equations (1) and (2) guarantee that if two vertices are on different shores, the edge between them has to be in the cut. On the other hand, Equations (3) and (4) ensure that the edge between vertices on the same

shore will not be in the cut. Equation (5) puts vertex 1 in S_c. Equation (6) states that a cut can be counted only if there are edges assigned to it. Together with Equation (3), we have that $V \setminus S_c \neq \emptyset$. The packing constraint in Equation (7) ensures that each edge is contained in at most one cut.

2.2. Extended Formulation

Let D denote the set of *all* nonempty cuts in G. A natural formulation is based on variables $x_\delta \in \{0, 1\}$, representing whether $\delta \in D$ is part of a cut packing or not.

$$\begin{aligned} \max \quad & \sum_{\delta \in D} x_\delta & & \\ \text{s.t.} \quad & \sum_{\substack{\delta \in D: \\ ij \in \delta}} x_\delta \leq 1 & & \forall ij \in E \qquad (8) \\ & x_\delta \in \{0, 1\} & & \forall ij \in E,\ \forall \delta \in D. \end{aligned}$$

This model was presented already in Caprara et al. [2004]. We remark that it formally results from a Dantzig-Wolfe reformulation of our compact formulation by keeping Equation (7) in the master problem and reformulating Equations (1) through (6) into m identical subproblems, which are then aggregated into a single subproblem, thereby completely eliminating the symmetry from the compact formulation.

2.3. Complexity

As previously mentioned, the cut packing problem is NP-hard in general [Colbourn 1987]. Additionally, we will specify the parameterized complexity of the parameterized version k-CUT PACKING of the cut packing problem. Given a graph $G = (V, E)$ and a parameter $k \in \mathbb{N}$, the k-CUT PACKING problem is to determine whether there exists a cut packing in G of cardinality greater than or equal to k. The parameterized complexity reveals whether it is likely that there exists a fixed-parameter tractable algorithm for k-CUT PACKING, that is, an algorithm that solves k-CUT PACKING with running time $\mathcal{O}(f(k) \cdot p(|V| + |E|))$, where $f : \mathbb{N} \to \mathbb{N}$ is an arbitrary computable function and $p : \mathbb{N} \to \mathbb{N}$ a polynomial function. The parameterized complexity class FPT consists of all fixed-parameter tractable problems. Additionally, Downey and Fellows [1995] introduced a hierarchy of complexity classes for parameterized problems, called *W-hierarchy*. We will prove that k-CUT PACKING is W[1]-hard by a reduction from the parameterized version k-INDEPENDENT SET of the independent set problem, which is known to be W [1]-hard [Downey and Fellows 1995].

COROLLARY 2.1. *The parameterized version k-CUT PACKING of the cut packing problem is W[1]-hard.*

PROOF. This follows immediately from Colbourn [1987] using the same construction as in the NP-hardness proof of the cut packing problem. □

It is widely believed that W[1]-hard problems are not fixed-parameter tractable [Downey and Fellows 1999; Flum and Grohe 2006]. Hence, the W[1]-hardness of the parameterized version of the cut packing problem motivates even further the study of practical algorithms for the cut packing problem.

3. ALGORITHMIC INGREDIENTS

3.1. Big Picture

In this section, we describe the components of a full branch-price-and-cut algorithm to solve the extended formulation to integer optimality. The general scheme is that of linear programming (LP) based branch-and-bound. In each node of the

branch-and-bound tree, the LP relaxation needs to be solved first. As the number of cuts in a graph is exponential in the size of the graph, we have to solve the LP relaxation of the formulation in Equation (8) by column generation. In this context, the LP relaxation of the formulation in Equation (8) is called the *master problem (MP)*. In column generation, one starts with a small set of variables of the master problem, yielding a *restricted master problem (RMP)*. We alternately solve the RMP (a linear program!) to optimality and add variables having positive reduced cost. Computing the reduced cost of a variable involves a dual (optimal) solution to the RMP, which we simultaneously obtain when solving the RMP. As a coefficient column of Equation (8) represents a cut in G, we interchangeably speak of adding variables, or columns, or cuts to the RMP. In order to find a positive reduced cost variable (or to prove that none exists), we solve an auxiliary optimization problem called the *pricing problem* (Section 3.2). If no more positive reduced cost variables are found, the master problem and hence the LP relaxation of the formulation in Equation (8) is optimally solved.

A potential strengthening of the LP relaxation is to separate cutting planes after the column generation process (Section 3.4). If a cutting plane cuts off the current solution of the RMP, we add it. The enlarged RMP (by one row) is resolved by column generation again. Since each cutting plane (after all, an additional constraint) introduces a new dual variable, there is an impact on the calculation of the reduced cost. We adapt the pricing problem to accommodate the new dual information. We repeat this process until we find no more cutting planes or some upper bound on the number of added cutting planes is reached. The column generation procedure combined with the separation of cutting planes is depicted in Figure 1.

In order to solve the extended formulation to integer optimality, we embed this procedure in a branch-and-bound tree, where we use specialized branching rules (Section 3.3). The obtained algorithm is called *branch-price-and-cut*. Note that in branch-price-and-cut algorithms, the LP relaxation in every node of the branch-and-bound tree is solved with column generation. Since only one RMP is maintained, columns generated in one node are available in all nodes. In many cases, cutting planes are separated only in the root node. We refer to Desrosiers and Lübbecke [2011] for a thorough introduction to branch-price-and-cut algorithms.

3.2. Solving the Pricing Problem

In order to find a variable/column/cut of positive reduced cost, or to conclude that none exists, we seek a maximum reduced cost cut. Let $\pi = (\pi_{ij})_{ij \in E}$ denote the current dual solution corresponding to Equation (8) in the RMP. Note that we have a constraint per edge in the extended formulation and hence, each dual variable π_{ij} corresponds to an edge $ij \in E$. The reduced cost of a variable x_δ is computed as $1 - \sum_{ij \in \delta} \pi_{ij}$ and is nonpositive for all $\delta \in D$ at optimality of the master problem. The pricing problem thus amounts to solving $\max_{\delta \in D}(1 - \pi^T y^\delta) = 1 - \min_{\delta \in D}(\pi^T y^\delta)$, where $y^\delta = (y^\delta_{ij})_{ij \in E}$ is a binary vector indicating whether a given edge $ij \in E$ belongs to cut $\delta \in D$.

At the root node, dual variables π_{ij} are nonnegative, and solving the pricing problem amounts to solving a *minimum* cut problem (with nonnegative edge weights). This enables us to benefit from the broad range of state-of-the-art minimum cut algorithms. We use the Stoer-Wagner algorithm introduced in Stoer and Wagner [1997] and then generalized to hypergraphs in Klimmek and Wagner [1996]. After having taken branching decisions (see later), dual variables π_{ij} are no longer restricted in sign, and we face a min-cut problem with arbitrary edge weights. This problem is equivalent to a *maximum* cut problem with arbitrary edge weights, which is obtained by multiplying the edge weights by -1. The maximum cut problem with arbitrary edge weights is NP-hard and thus we model and solve the min-cut problem with arbitrary edge weights as a binary program. We have seen such a model already when stating the compact

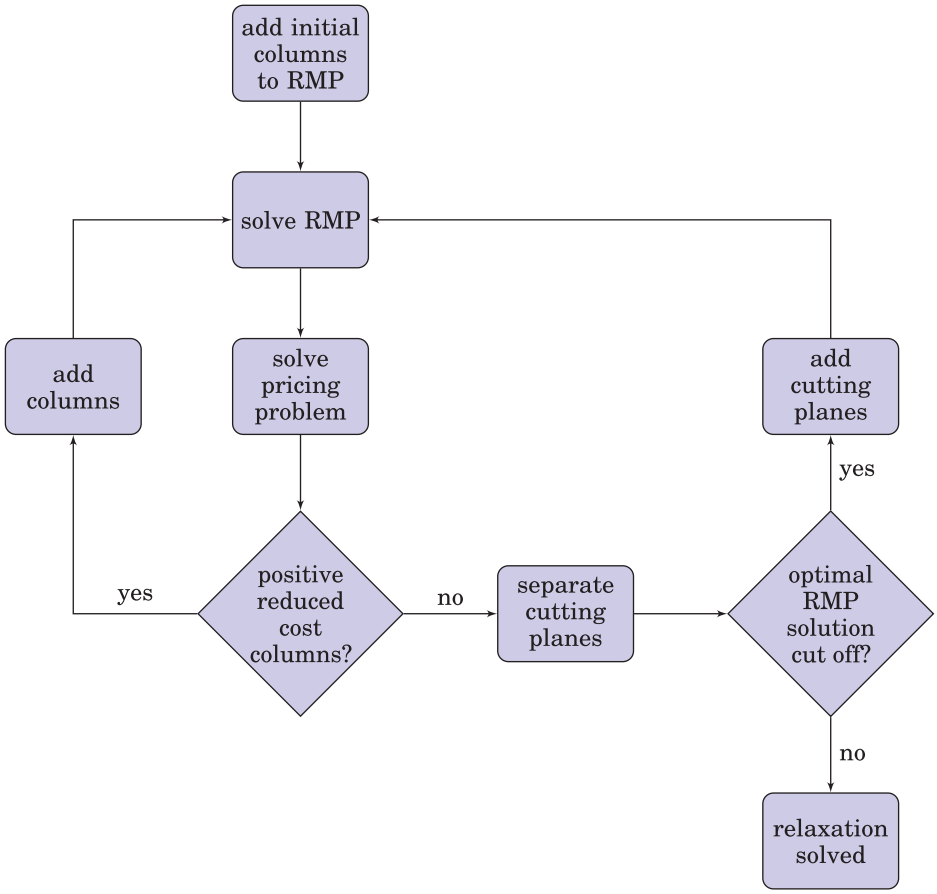

Fig. 1. Flow chart of column generation combined with separation of cutting planes.

formulation in Equations (1) through (7) for cut packing. When computing just one cut, there is no need for Equation (7) or for an index c; moreover, we replace the objective function by the reduced cost function for the potential cut. Hence, the pricing problem can be written as

$$
\begin{aligned}
1 - \min \quad & \sum_{ij \in E} \pi_{ij} y_{ij} & & \\
\text{s.t.} \quad & u_i - u_j + y_{ij} \geq 0 & \forall ij \in E \quad & (9)\\
& u_j - u_i + y_{ij} \geq 0 & \forall ij \in E \quad & (10)\\
& y_{ij} - u_i - u_j \leq 0 & \forall ij \in E \quad & (11)\\
& u_i + u_j + y_{ij} \leq 2 & \forall ij \in E \quad & (12)\\
& u_1 = 0 & & (13)\\
& 1 \leq \sum_{ij \in E} y_{ij} & & (14)\\
& u_i, y_{ij} \in \{0, 1\} & \forall ij \in E,\ \forall i \in V, &
\end{aligned}
$$

where $\pi = (\pi_{ij})_{ij \in E}$ denotes the current dual solution corresponding to Equation (8).

We will call a shore S of a cut $\delta(S)$ *connected* if the subgraph of G induced by S is connected. Suppose that one of the shores S and $V \setminus S$ of a cut $\delta(S)$ with positive reduced cost resulting from the pricing problem is not connected. Without loss of generality, we can assume that S is not connected and let $S_1, \ldots, S_K$ with $K \geq 2$ be the connected components of S. The induced cuts $\delta(S_1), \ldots, \delta(S_K)$ are pairwise disjoint and satisfy $\delta(S) = \delta(S_1) \cup \ldots \cup \delta(S_K)$. Because at least one cut $\delta(S_k)$ with $k \in \{1, \ldots, K\}$ will have positive reduced cost, we add cuts $\delta(S_k)$, $k \in \{1, \ldots, K\}$ with positive reduced cost instead of cut $\delta(S)$ to the restricted master problem.

Furthermore, the cut $\delta(S)$ will not be part of any optimal solution to the LP relaxation of the formulation in Equation (8), since we could replace $\delta(S)$ by the cuts $\delta(S_1), \ldots, \delta(S_K)$, which leads to a solution with greater objective value.

Desrochers et al. [1992] remark that pricing disjoint columns helps with arriving at integral solutions. In order to find such a set of disjoint columns, we use a greedy heuristic during pricing, where we compute a cut δ_1 of maximum reduced cost, fix all variables corresponding to edges $ij \in \delta_1$ to 0, and resolve the pricing problem, creating a new cut δ_2, disjoint to δ_1. We again fix the edges from δ_2 and iterate until no cut δ_{k+1} can be found. The heuristic cut packing with objective function value k is $\{\delta_1, \ldots, \delta_k\}$.

As the pricing problem further down in the branch-and-bound tree resembles a max-cut problem, we employ further heuristics to price out favorable columns based on max-cut heuristics [Sahni and Gonzalez 1976] inspired by the formulation of the problem. Columns that are not immediately added to the restricted master problem are collected in a column pool, which is searched for positive reduced cost columns before calling any pricing algorithm.

3.3. Branching

In case the obtained optimal solution to the restricted master problem (which is a linear program) is fractional, we need to employ branching schemes in order to find an optimal integral solution. For the purposes of this section, we modify our extended formulation to become a set partitioning problem by introducing a binary slack variable $\bar{x}_{ij}$ for edge $ij \in E$ in Equation (8): $\sum_{\delta \ni ij} x_\delta + \bar{x}_{ij} = 1$. An interpretation of this variable is that it attains value 1 if and only if edge ij is not contained in any cut in the cut packing. We employ two different branching rules: first we try to branch on *aggregated* original y^0_{ij} variables, which are defined as $y^0_{ij} = \sum_{c=1}^{m} y^c_{ij}$, and if that is not possible, we will use Ryan-Foster branching [Ryan and Foster 1976] to branch on pairs of edges.

In the first case, given an LP solution $x^\star$ of the master problem, we calculate the value of variables y^0_{ij} for each edge $ij \in E$ as $y^0_{ij} = \sum_{\delta : ij \in \delta} x^\star_\delta = 1 - \bar{x}^\star_{ij}$. If y^0_{ij} is fractional for some edge $ij \in E$, we create two branches by setting the slack variables $\bar{x}_{ij} = 1$ in one and $\bar{x}_{ij} = 0$ in the other branch, enforcing that either no cut or exactly one cut contains edge ij. The first case can be respected directly in the pricing by setting $y_{ij} = 0$.

In the second case, the dual variables π_{ij} corresponding to edge ij can be negative, which we cannot respect in a standard min-cut algorithm. This justifies solving the pricing problem using a binary program instead of using a classical combinatorial min-cut algorithm throughout the branch-and-price tree because of mixed negative and positive edge weights.

In case $y^0_{ij} = 1 - \bar{x}_{ij}$ is integral for all edges $ij \in E$, we branch on pairs of edges ij and $k\ell$, analogous to Ryan-Foster branching [Ryan and Foster 1976]. The branching decisions are either $\bar{x}_{ij} = \bar{x}_{k\ell}$ (this is called the *same* branch) or $\bar{x}_{ij} + \bar{x}_{k\ell} \geq 1$ (the *diff* branch). In the *same* branch, either the two edges must appear together in a cut or neither of the edges must be part of a cut. In the *diff* branch, both edges are not allowed to be in the same cut. These conditions can easily be respected in the binary programming

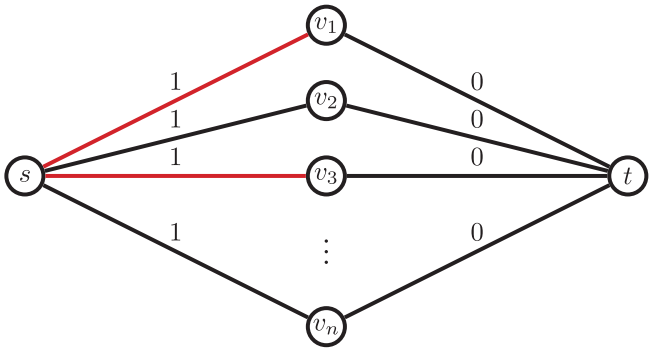

Fig. 2. Sketch of graph $G' = (V', E')$ that is obtained from Graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$. Besides the edges on a path from s to t, edges $\{s, u\} \in E'$ and $\{s, w\} \in E'$ with $\{u, w\} \in E$ are forbidden edge pairs. This is exemplarily depicted for the edge $\{v_1, v_3\} \in E$ of graph G.

formulation of the pricing problem. Theoretically, we could respect these branching decisions in a combinatorial algorithm by constructing two reduced graphs for the *same* branching decision by contracting edges ik and $j\ell$ (and $i\ell$ and jk, respectively). Unfortunately, this is only feasible for a rather small number of pairs and does not scale in the case of a large number of consecutive branching decisions.

Solving the pricing problem with an arbitrary number of *diff* constraints is an NP-hard problem on its own: given a graph G, a weight function $c : E \to \mathbb{Q}^+$, and a set of edge pairs $P \subseteq E \times E$, we want to find a minimum cut with the constraint that at most one edge from each pair $p \in P$ is active in the cut. We will prove that the decision version MINCUT-CONFLICTS of this problem is NP-complete by a reduction from the decision version INDEPENDENT SET of the independent set problem.

THEOREM 3.1. *The decision problem MINCUT-CONFLICTS is NP-complete.*

PROOF. It is easy to see that MINCUT-CONFLICTS is in NP.

Let $G = (V, E)$ and $k \in \mathbb{N}$ be an instance of the decision problem INDEPENDENT SET, where V is the vertex set and E the edge set of the graph G. We construct an instance of MINCUT-CONFLICTS with graph $G' = (V', E')$, parameter $k' \in \mathbb{N}$, weight function c, and set of forbidden edge pairs P as follows: let $V' = V \cup \{s, t\}$ be the vertex set, where s and t are new vertices, and let $E' = \{\{s, v\} : v \in V\} \cup \{\{v, t\} : v \in V\}$ be the edge set of the graph $G' = (V', E')$. Furthermore, we define the edge weights $c(\{s, v\}) = 1$ and $c(\{v, t\}) = 0$ for all initial vertices $v \in V$ as well as the set of forbidden edge pairs as $P = \{(\{s, v\}, \{v, t\}) : v \in V\} \cup \{(\{u, t\}, \{v, t\}) : \{u, v\} \in E\}$. The parameter k' is defined as $k' = |V| - k$. A sketch of the graph G' is depicted in Figure 2.

We will prove that there exists a cut with weight lower than or equal to k' in G' if and only if there exists an independent set of cardinality greater than or equal to k in G.

Suppose $I \subseteq V$ is an independent in G with $|I| \geq k$. Then, we define the set $S = I \cup \{s\}$, which induces the cut $\delta(S) = \{\{v, t\} : v \in I\} \cup \{\{s, v\} : v \notin I\}$ in G' with weight $c(\delta(S)) = |V| - |I| \leq |V| - k = k'$.

Now suppose $\emptyset \subsetneq S \subsetneq V$ induces a cut $\delta(S) \subseteq E$ in G' with weight $c(\delta(S)) \leq k'$. We define the independent set $I = \{v : \{v, t\} \in \delta(S)\}$ in G with cardinality $|I| = |V| - c(\delta(S)) \geq k$. □

As we have seen previously, the set $\emptyset \subsetneq S \subsetneq V$ inducing a cut $\delta(S)$ with positive reduced cost $1 - \sum_{ij \in \delta} \pi_{ij}$ might not be connected (if S is connected but $V \backslash S$ is not, we exchange the roles of S and $V \backslash S$). If *same* branching decisions have been added, the special handling of these cuts is not applicable. Assume that the branching decision

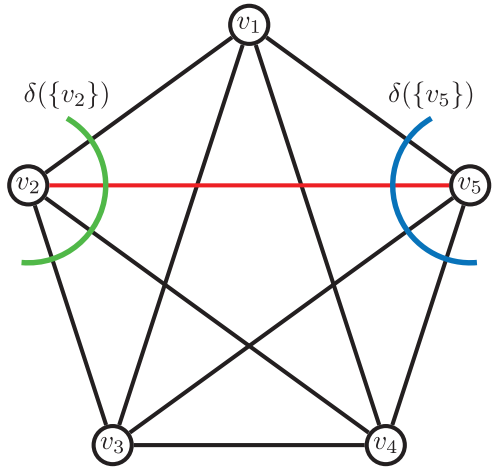

Fig. 3. A clique $K = (V_K, E_K)$ with $V_K = \{v_1, \ldots, v_5\}$ together with cuts $\delta(\{v_2\})$ and $\delta(\{v_5\})$. The edge $\{v_2, v_5\} \in E_K$ belongs to both cuts $\delta(\{v_2\})$ and $\delta(\{v_5\})$.

forces edges $e = ij$ and $f = k\ell$ to be either in one cut or in no cut. Then the pricing problem could create a cut $\delta(S)$ with positive reduced cost and both edges in the cut. Suppose S_1 and S_2 are the connected components of S and $i \in S_1$, $k \in S_2$ holds. In this case, we cannot split up the cut $\delta(S)$ into individual cuts $\delta(S_1)$ and $\delta(S_2)$ as $e \in \delta(S_1)$ and $f \notin \delta(S_1)$ as well as $f \in \delta(S_1)$ and $e \notin \delta(S_2)$ hold. Hence, both cuts $\delta(S_1)$ and $\delta(S_2)$ individually do not respect the branching decision, whereas the cut $\delta(S)$ does. If we encounter this case, we will add the cut $\delta(S)$ to the master problem, knowing that it will not be part of any optimal integral solution. Besides potentially slowing down the solution process, this will cause no further harm.

3.4. Cutting Planes

The optimality gap between an optimal solution to the LP relaxation of the extended formulation and an optimal integral solution can be arbitrarily bad: suppose we solve the cut packing problem on a clique $K = (V_K, E_K)$ with n vertices, that is, $E_K = \{\{u, v\} \subseteq V : u \neq v\}$. The cardinality of an optimal cut packing for a clique K is 1, because any two cuts in K are never disjoint, which is exemplarily depicted in Figure 3. However, the optimal LP solution value will be $\frac{n}{2}$, with cuts separating each vertex from the remainder of the clique. The master variables corresponding to cuts $\delta(\{v\})$ for $v \in V_K$ all take the value $\frac{1}{2}$. To improve the LP bound, we thus separate clique inequalities of the form

$$\sum_{\delta : \delta \cap E_K \neq \emptyset} x_\delta \leq 1 \tag{15}$$

for a clique $K = (V_K, E_K)$ in the root node. Cliques are found using a maximum weighted clique heuristic based on an algorithm by Borndörfer and Kormos [1997].

For the column generation algorithm, when adding cutting planes to the master problem, new dual variables are introduced that need to be respected in the pricing problem. Let $\mathcal{K}$ be the set of corresponding cliques and let the nonnegative dual variable of these cutting planes be $\mu_K \geq 0$, one for each clique $K \in \mathcal{K}$. Given a graph $G = (V, E)$ and a set of cliques $\mathcal{K}$, we construct a hypergraph $H = (V, F)$ where original edges

are copied and all cliques are converted to hyperedges. Let $V(K)$ denote the vertices of clique $K = (V_K, E_K)$, that is, $V(K) = V_K$. Formally, $F = E \cup \{V(K) : K \in \mathcal{K}\}$ and the dual values from the clique inequalities are transferred to the corresponding hyperedges.

In the root node, we use the hypergraph extension of the Stoer-Wagner algorithm to solve the minimum cut problem. When using the binary program to solve the pricing problem, we add variables z_K for each clique $K \in \mathcal{K}$ along with constraints

$$y_{ij} \leq z_K \qquad \forall ij \in E_K,\ \forall K \in \mathcal{K}.$$

The reduced cost for a cut changes to

$$1 - \left(\sum_{ij \in E} \pi_{ij} y_{ij} + \sum_{K \in \mathcal{K}} \mu_K z_K \right),$$

and the pricing problem's objective function becomes

$$1 - \min \sum_{ij \in E} \pi_{ij} y_{ij} + \sum_{K \in \mathcal{K}} \mu_K z_K.$$

Let us demonstrate that it is as easy to draw from the wealth of other classes of valid inequalities and incorporate them in our approach. As an example, we consider the well-known odd-cycle inequalities: for a given cycle $C = (V_C, E_C)$ with an odd number of vertices $|V_C|$ (equivalently odd $|E_C|$), the odd-cycle inequality takes the form

$$\sum_{\delta : \delta \cap E_C \neq \emptyset} \frac{|\delta \cap E_C|}{2} \cdot x_\delta \leq \frac{|E_C| - 1}{2}.$$

Notice that the value $\frac{|\delta \cap E_C|}{2}$ is an integer for all cuts δ and cycles C.

As before, we need to respect the dual variables introduced when separating odd-cycle inequalities. Let $\mathcal{C}$ be the set of corresponding odd cycles and let $\mu_C \geq 0$ be the nonnegative dual variables of these cutting planes, one for each odd cycle $C \in \mathcal{C}$. In order to respect the dual variables for the added odd-cycle inequalities, the reduced cost for a cut changes to

$$1 - \left(\sum_{ij \in E} \pi_{ij} y_{ij} + \sum_{C \in \mathcal{C}} \sum_{ij \in E_C} \frac{1}{2} \mu_C y_{ij} \right),$$

and the pricing problem objective becomes

$$\begin{aligned} &1 - \min \sum_{ij \in E} \pi_{ij} y_{ij} + \sum_{C \in \mathcal{C}} \sum_{ij \in E_C} \frac{1}{2} \mu_C y_{ij} \\ &\quad = 1 - \min \sum_{ij \in E} \left(\pi_{ij} + \sum_{\substack{C \in \mathcal{C}: \\ ij \in E_C}} \frac{1}{2} \mu_C \right) y_{ij}. \end{aligned}$$

Notice that only the objective function of the pricing problem changes and no additional variables or constraints are needed. Therefore, we can still apply a combinatorial algorithm (at the root node) to solve the pricing problem to optimality.

Rebennack et al. [2011] separated different types of cutting planes for the independent set problem including clique and odd-cycle inequalities. In their computational tests, they observed that most of the separated odd-cycle inequalities correspond to

cycles of length three or, in other words, cliques of size three (that were then lifted to maximal cliques). This is one of the reasons the impact of odd-cycle inequalities in reducing the LP bound is not that important in comparison to clique inequalities. Because the cut packing problem is closely related to the independent set problem, we expect similar results concerning the odd-cycle inequalities for the cut packing problem and decided to not separate odd-cycle inequalities in our branch-price-and-cut algorithm at all.

3.5. Combinatorial Dual Bounds

In order to further strengthen the dual bound, we make use of a combinatorial upper bound that is similar to a combinatorial upper bound proposed by Caprara et al. [2004]. An *edge clique cover* $\mathcal{Q}$ is a set of cliques with $\bigcup_{K \in \mathcal{Q}} E(K) = E$, where $E(K)$ is the set of edges of clique K. We observe that the edges of a clique can be crossed by at most one cut in any cut packing. Thus, the cut packing number $\gamma(G)$ is bounded from above by the number of cliques in an edge clique cover of minimum cardinality. Determining this minimum number is itself an NP-hard problem [Kou et al. 1978]. We solve the minimum edge clique covering problem approximately using the algorithm of Gramm et al. [2006] in order to obtain a valid upper bound. In addition, we compute an edge clique cover at the outset of the overall algorithm and add all induced clique inequalities to the master problem before starting column generation.

3.6. Primal Heuristics

Besides using the generic heuristics included in branch-price-and-cut frameworks, we try to find good cut packings by using a (noncrossing) maximum *s-t* path cut packing heuristic [Colbourn 1987], which we call *maxpath*. Furthermore, we use an approximation algorithm for the independent set problem [Halldórsson and Radhakrishnan 1997] for degree bounded graphs by sorting the vertices according to nondecreasing degree and sequentially adding the next available vertex to the independent set. The resulting independent set $I \subseteq V$ is transferred to the cut packing $S = \{\delta(\{v\}) : v \in I\}$. In order to assess the quality of the heuristic, we calculate a maximum independent set exactly by solving the textbook binary program formulation

$$\max \left\{ \sum_{i \in V} x_i : x_i + x_j \leq 1 \, \forall ij \in E, \; x_i \in \{0, 1\} \, \forall i \in V \right\}.$$

4. COMPUTATIONAL SETUP AND RESULTS

We implemented our branch-price-and-cut algorithm in SCIP 3.0.1 [Achterberg 2009] with CPLEX 12.4.0.1 as the LP solver. The maximum weighted clique heuristic [Borndörfer and Kormos 1997] that we use to separate clique inequalities is readily available in SCIP. All computations were performed on Intel Core i7-2600 CPUs with 16GB of RAM on openSUSE 12.1 workstations running Linux kernel 3.1.10. The default time limit is 3,600 seconds unless stated otherwise.

We applied our approach to instances from the 10th DIMACS implementation challenge [Bader et al. 2013]. We expect difficult graph partitioning problems to be hard for the cut packing problem too, as in both settings the vertex set is partitioned in some way. In addition, we collected coloring instances from Trick [1993] and investigate the the performance of our algorithm on these instances, because each color class in a coloring constitutes an independent set, to which a cut packing is intimately related.

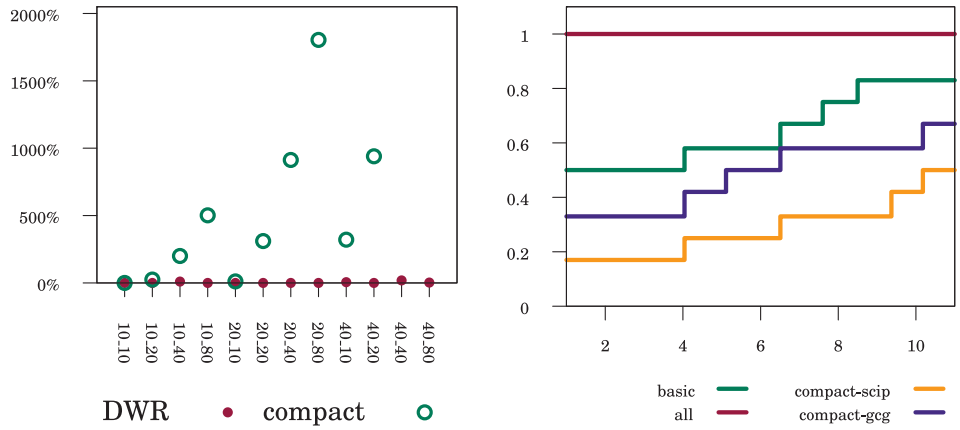

Fig. 4. Relative gap between primal and dual bounds at the root node for the compact and extended formulations for random graphs of type *graph-n-m* (left) and a performance profile for all the different formulations and settings (right) showing in how many instances (y-axis in percent) an algorithm is at most $\log_2(x)$ times slower (factor on the x-axis).

We further generated smaller random graphs using several graph generators such as `Rudy` by Rinaldi [1998], `Randgraph` by Pettie and Ramachandran [2006], and `NetworkX` 1.7 by Hagberg et al. [2008]. Moreover, we used the `MUSKETEER` graph generator [Gutfraind et al. 2012] to obtain a set of graphs that are similar to the graphs from the literature. We used the random seed value 1 for the generation of random graphs unless stated otherwise. In total, our test set comprises around 100 instances.

4.1. Compact Versus Extended Formulation

In Figure 4, we compare the integrality gaps and the running times of the compact and the extended formulations on a small test set generated with the `Randgraph` graph generator. A generated graph *graph-n-d* consists of a random tree on n vertices plus $\max\{0, d\% \cdot \frac{n\cdot(n-1)}{2} - n + 1\}$ additional randomly selected edges. In Figure 4, the *all* plot reflects the implementation with all presented features; in *basic*, only the bare column generation implementation is visualized. The bare column generation implementation uses the binary programming formulation for the pricing problem and the presented branching scheme. Furthermore, we solved the compact formulation with the branch-and-cut solver `SCIP` 3.0.1 [Achterberg 2009] (*compact-scip* plot) and the generic branch-price-and-cut solver `GCG` 2.0.1 [Gamrath and Lübbecke 2010] (*compact-gcg* plot). Note that `GCG` derives the extended formulation by reformulating the compact formulation using the Dantzig-Wolfe reformulation and solves the extended formulation with branch-price-and-cut.

When looking at the time needed to solve the instances to optimality, the extended formulation outperforms the compact one on almost every instance, which is depicted in the performance profile in Figure 4. Furthermore, our basic implementation performs better than the generic branch-price-and-cut solver `GCG`. We remark that `GCG` solves the same pricing problem and uses additional features tailored for the branch-price-and-cut algorithm, but applies other branching rules. The difference in solution time increases further if all presented features (except for odd-cycle cuts, which are not implemented) are added. In particular, `SCIP` was only able to solve three out of 12 instances to optimality, whereas `GCG` solved five of 12 instances to optimality. The basic implementation solving the extended formulation, however, solved eight out of

Table I. Number of Solved Instances and Mean of Solution Times for the Extended Formulation for Cut Packing on Random General, Bipartite, and Chordal Graphs (All Reported Times in *s* Are Shifted Geometrical Means)

	Nodes												
	10		20		40		80		160		320		
Type	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Mean Time
random	4/4	0.0	4/4	0.0	4/4	25.2	0/4	3600	0/4	3600	0/4	3600	224.3
bipartite	4/4	0.0	4/4	0.0	4/4	0.1	4/4	1.0	4/4	20.8	4/4	527.8	13.9
chordal	4/4	0.0	4/4	0.0	4/4	0.1	4/4	1.3	4/4	9.3			1.7

12 instances, and with all features enabled, all instances can be solved to optimality within the time limit.

If we look at the integrality gaps at the root node, we see in Figure 4 that the gap obtained by the extended formulation is remarkably better than the gap obtained by the compact formulation. The bound improvement translates to a better pruning of branch-and-price nodes, and the removal of the symmetry by aggregation leads to fewer branching decisions, which explains the huge difference in solution time.

4.2. Different Graph Classes

It is known that the cut packing problem can be solved in polynomial time on bipartite and chordal graphs. In order to investigate whether this complexity improvement translates to faster solution times also for our (exponential time) algorithm, we evaluate the performance of the extended formulation on these instance types using random instances generated with `NetworkX` 1.7. We generated connected bipartite graphs and connected chordal graphs. The latter were generated from a random graph on n vertices, in which every edge occurs with probability $d\%$. To this graph, chords were iteratively added until the graph became chordal by using `NetworkX` to search for chordless cycles of size larger than 3. The chord was added randomly. We generated graphs with 10, 20, 40, 80, 160, and 320 vertices and edge densities of 10%, 20%, 40%, and 80%. We only generated chordal graphs with up to 160 vertices as using `NetworkX` to find all chordless cycles in larger graphs was too expensive. The results are summarized in Table I.

We notice that the column generation procedure runs significantly faster on the graph classes where the problem is solvable in polynomial time. We are able to solve the cut packing problem to optimality in all generated bipartite and chordal graphs with up to 320 vertices (160 vertices, respectively), in contrast to general random graphs where we fail to solve instances larger than 40 vertices.

To evaluate the performance of our implementation on real-world data, we selected those real-world instances from the the 10th DIMACS implementation challenge with 500 vertices or fewer and all coloring instances from Trick [1993] with fewer than 5,000 edges. We also tried to solve the problem on instances with hidden optimal solutions for classical graph problems [Xu 2010], but we were not able to successfully solve the root node of any of the instances within the 1-hour time limit.

To compare the performance in these graphs to random graphs, we generated both a random graph and a similar graph for each real-world graph from the the 10th DIMACS implementation challenge. These graphs have the same number of vertices and edges. The random graphs were generated with `Randgraph`; a connected graph *rand-n-m* consists of n vertices and m edges. In order to create similar graphs, we used `MUSKETEER`. The new graphs, forced to be connected, are denoted by suffix *-m*. The results on these graphs are compared in Table II.

We observe that our algorithm solved five out of nine real-world instances to optimality, but only one out of nine of the corresponding random graphs. In contrast, the

Table II. Branch-and-Bound Nodes and Solution Time (or Relative Gap Achieved Within Time Limit) for Real-World DIMACS Partitioning Benchmark Graphs

Name	Nodes	Time/Gap	Name	Nodes	Time/Gap	Name	Nodes	Time/Gap
karate	1	0.1	karate-m	1	0.1	rand-34-78	25	7.5
dolphins	51	317.4	dolphins-m	1	1.5	rand-62-159	>188	8.0%
lesmis	3	2.3	lesmis-m	8	21.4	rand-77-254	>77	19.2%
polbooks	4	95.9	polbooks-m	1	36.3	rand-105-441	>74	17.6%
adjnoun	>102	5.6%	adjnoun-m	>46	8.5%	rand-112-225	>304	5.2%
football	>29	9.5%	football-m	9	2327.9	rand-115-613	>57	32.3%
jazz	1	380.4	jazz-m	>1	27.25%	rand-198-2742	>1	102.9%
celegansneural	>1	4195.0%	celegansneural-m	>1	1161.3%	rand-297-2148	>1	3612.2%
celegans_meta	>1	866.7%	celegans_meta-m	>1	1646.8%	rand-453-2025	>1	4038.6%

graphs edited by MUSKETEER are more similar to the original graphs, and indeed, our algorithm performs better on those than on arbitrary random graphs.

Overall, it seems that our algorithm is effective on small real-world problems and on those graph classes where the problem is easy. On the other hand, random graphs seem to be a challenge, even when they are reasonably sparse (10% of overall edges).

4.3. Influence of Particular Implementation Parts

In addition to the basic column generation algorithm, we presented a few enhancements with the purpose of saving computation time. From our experiments, the clique cover, clique separator, and combinatorial pricing algorithm seem to have the largest impact on the performance and we will restrict attention to those only. We plot performance profiles in Figure 5.

It becomes evident that disabling the initial clique cover heuristic including the resulting clique inequalities causes the largest performance drop. The clique separator can partially catch this drop by separating the clique inequalities at runtime, but if both the clique separator and the clique cover heuristic are disabled, the performance decrease is significant. A similar improvement can be noticed if these features are added to a vanilla column generation procedure. In our case, the combinatorial pricing is a nice add-on but nonetheless has not that much influence on the solution time as expected.

4.4. Relation to Independent Set

Motivated by the close connection to the maximum independent set (MIS) problem, we want to study the relationship between an MIS and an optimal cut packing on selected instances. For a given graph, we compare the solution values of a greedy algorithm to approximate the cardinality of an MIS, the cardinality $\alpha(G)$ of an optimal MIS computed via an integer program, to the values of the maximum s-t path heuristic and $\gamma(G)$. If an instance has not been solved to optimality, only the best-known dual bound on the cut packing number $\gamma(G)$ is shown. This can be seen in Figure 6.

We notice that the value of the solution found by the s-t maximal path heuristic is often much worse than any of the other algorithms. The solution found by the MIS heuristic is better but is usually much worse than an optimal solution. In contrast, $\alpha(G)$ is mostly identical to $\gamma(G)$. This comes as a surprise as there is a theoretical gap of a factor of 2 that we do not observe in the instances in our experiments. We are aware of the fact that the gap will be closer to 2 when considering sparser graphs and in particular paths and trees.

5. SUMMARY AND CONCLUSIONS

We presented an exact algorithm for the cut packing problem in undirected graphs and assessed its performance on about 100 random and benchmark graphs from the

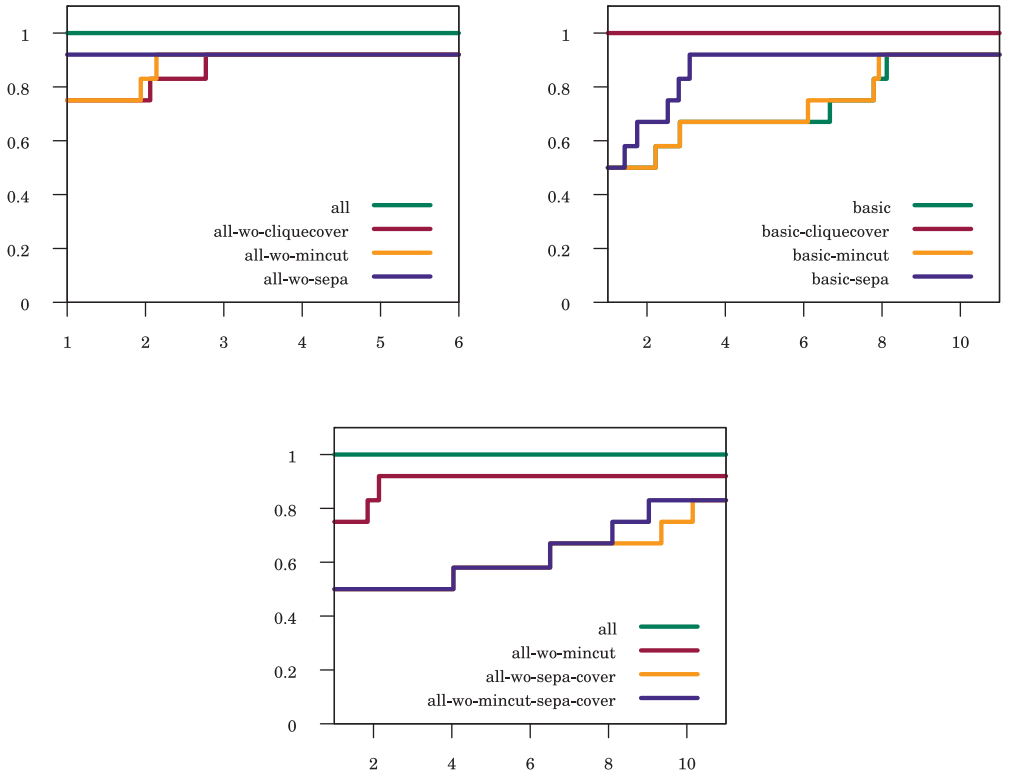

Fig. 5. Performance profiles for clique cover, clique separator, and combinatorial algorithm showing in how many randomly generated instances (y-axis in percent) a given setting is at most $\log_2(x)$ times slower (factor on the x-axis). The plot on the upper left states the influence of disabling each part of the algorithm; on the upper right the influence of enabling each part on a basic implementation is shown. The plot on the bottom presents the performance of a concurrent deactivation.

literature. The current size limits of our approach are up to 80 vertices on random graphs, up to 500 vertices on graph partitioning benchmark graphs, and up to 5,000 edges on vertex coloring benchmark graphs. Our algorithm performs much better on instances where the cut packing problem is solvable in polynomial time. Our experiments revealed that $\gamma(G)$ is typically much closer (and often even identical) to $\alpha(G)$ than to the theoretically possible $2\alpha(G) - 1$.

We have made algorithmic use of the combinatorial structure of the problem: we used min-cut algorithms for the pricing problem; we computed a combinatorial dual bound for tightening the relaxation; and we exploited the problem's close relation to the independent set problem, for example, for designing primal heuristics. We compared our efforts to using a generic approach via the branch-price-and-cut solver GCG. We conclude that at the current state of the art, it still considerably pays to consider a problem-specific implementation, even though generic solvers may become competitive soon. Future computational research on cut packing should clarify whether it pays to work on (integer programming formulations for) better clique edge covers for stronger dual bounds, and whether different integer linear or semidefinite programming formulations for min-cut can speed up solving the pricing problem.

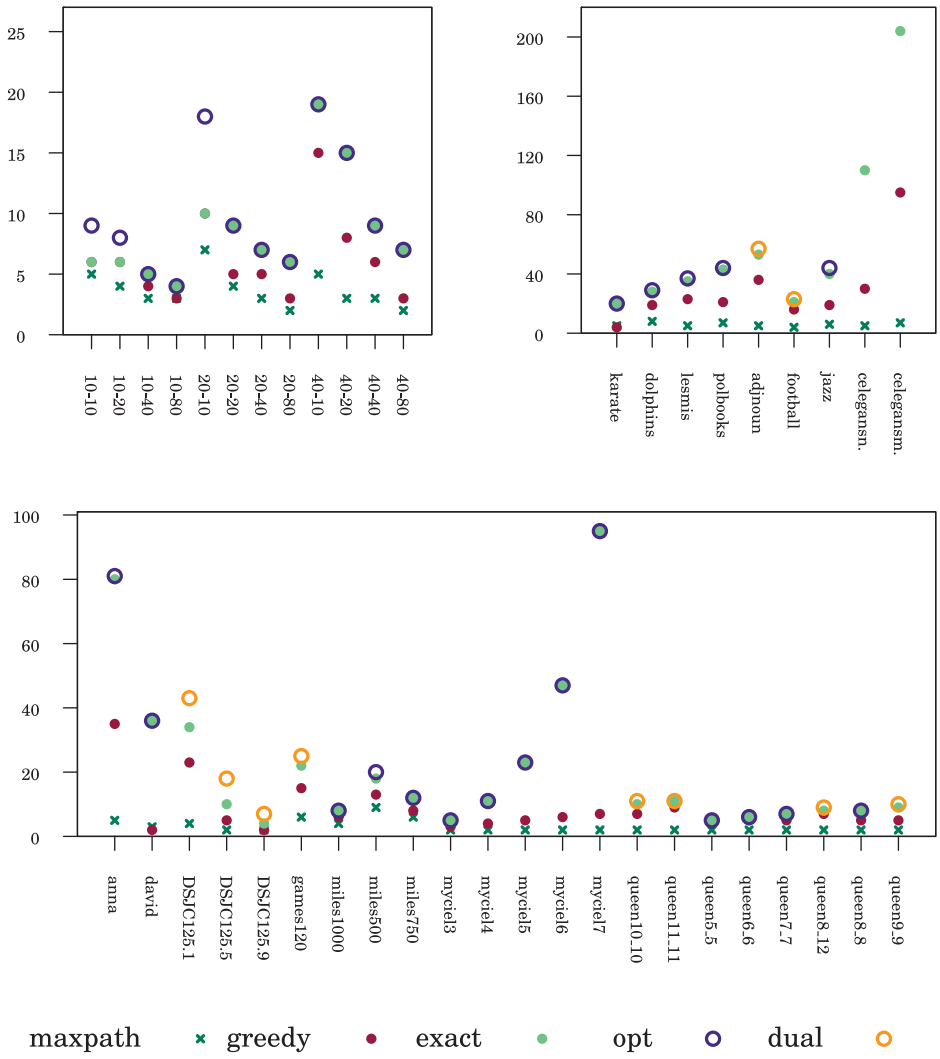

Fig. 6. Objective function values of maxpath heuristic, greedy and exact MIS, optimal cut packing or dual bound for random graphs (top left); DIMACS partitioning benchmark graphs (top right); and coloring graphs (bottom).

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments that helped to improve the presentation of the article and made it more accessible to nonexperts in branch-price-and-cut algorithms.

REFERENCES

T. Achterberg. 2009. SCIP: solving constraint integer programs. *Math. Programming Comp.* 1, 1 (2009), 1–41.

D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner (Eds.). 2013. *Graph Partitioning and Graph Clustering. 10th DIMACS Implementation Challenge Workshop.* Contemp. Mathematics, Vol. 588. American Mathematical Society.

M. Bergner, M. E. Lübbecke, and J. T. Witt. 2014. A branch-price-and-cut algorithm for packing cuts in undirected graphs. In *Experimental Algorithms*, J. Gudmundsson and J. Katajainen (Eds.), Vol. 8504. Springer International Publishing, Switzerland, 34–45.

R. Borndörfer and Z. Kormos. 1997. An algorithm for maximum cliques. Unpublished working paper, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

A. Caprara, A. Panconesi, and R. Rizzi. 2003. Packing cycles in undirected graphs. *J. Algorithms* 48 (2003), 239–256.

A. Caprara, A. Panconesi, and R. Rizzi. 2004. Packing cuts in undirected graphs. *Networks* 44, 1 (2004), 1–11.

C. J. Colbourn. 1987. *The Combinatorics of Network Reliability*. Oxford University Press, New York, NY.

C. J. Colbourn. 1988. Edge-packing of graphs and network reliability. *Discrete Math.* 72, 1–3 (1988), 49–61.

M. Desrochers, J. Desrosiers, and M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* 40, 2 (1992), 342–354. DOI: http://dx.doi.org/10.1287/opre.40.2.342

J. Desrosiers and M. E. Lübbecke. 2011. Branch-price-and-cut algorithms. In *Encyclopedia of Operations Research and Management Science*, J. J. Cochran (Ed.). John Wiley & Sons, Chichester. DOI: http://dx.doi.org/10.1002/9780470400531.eorms0118

R. G. Downey and M. R. Fellows. 1995. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theor. Comput. Sci.* 141, 12 (1995), 109–131. DOI: http://dx.doi.org/10.1016/0304-3975(94)00097-3

R. G. Downey and M. R. Fellows. 1999. *Parameterized Complexity*. Vol. 3. Springer Heidelberg.

J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory*. Vol. 3. Springer.

D. R. Fulkerson. 1971. Blocking and anti-blocking pairs of polyhedra. *Math. Programming* 1, 1 (1971), 168–194. DOI: http://dx.doi.org/10.1007/BF01584085

G. Gamrath and M. E. Lübbecke. 2010. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In *Experimental Algorithms*, P. Festa (Ed.), Vol. 6049. Springer-Verlag, Berlin, 239–252.

J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. 2006. Data reduction, exact, and heuristic algorithms for clique cover. In *Proc. 8th ALENEX*. 86–94.

A. Gutfraind, L. A. Meyers, and I. Safro. 2012. Multiscale Network Generation. (2012). arXiv:1207.4266.

A. A. Hagberg, D. A. Schult, and P. J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA, 11–15.

M. M. Halldórsson and J. Radhakrishnan. 1997. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica* 18, 1 (1997), 145–163.

R. Klimmek and F. Wagner. 1996. *A Simple Hypergraph Min Cut Algorithm*. Technical Report B 96-02. FU Berlin.

L. T. Kou, L. J. Stockmeyer, and C.-K. Wong. 1978. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM* 21, 2 (1978), 135–139.

C. L. Lucchesi and D. H. Younger. 1978. A minimax theorem for directed graphs. *J. Lond. Math. Soc.* 17 (1978), 369–374.

S. Pettie and V. Ramachandran. 2006. Randgraph graph generator. Retrieved from http://www.dis.uniroma1.it/challenge9/download.shtml.

S. Rebennack, M. Oswald, D.O. Theis, H. Seitz, G. Reinelt, and P. M. Pardalos. 2011. A branch and cut solver for the maximum stable set problem. *J. Combinatorial Optimization* 21, 4 (2011), 434–457.

G. Rinaldi. 1998. Rudy, a graph generator. Retrieved from http://www-user.tu-chemnitz.de/~helmberg/sdp_software.html.

J. T. Robacker. 1956. *Min-Max Theorems on Shortest Chains and Disjunct Cuts of a Network*. Technical Report RM-1660-PR. Rand Corporation.

D. M. Ryan and B. A. Foster. 1976. An integer programming approach to scheduling. *Opt. Res. Q.* 27, 2 (1976), 367–384.

S. Sahni and T. Gonzalez. 1976. P-complete approximation problems. *J. ACM* 23, 3 (1976), 555–565.

M. Stoer and F. Wagner. 1997. A simple min-cut algorithm. *J. ACM* 44, 4 (1997), 585–591.

M. Trick. 1993. Coloring instances. Retreived from http://mat.gsia.cmu.edu/COLOR/instances.html.

K. Xu. 2010. BHOSLIB: Benchmarks with hidden optimum solutions for graph problems. Retrieved from http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

Received October 2014; revised July 2015; accepted November 2015