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

Martin Bergner*, Marco E. Lübbecke, and Jonas T. Witt

Operations Research, RWTH Aachen University,
Kackertstr. 7, 52072 Aachen, Germany
{martin.bergner,marco.luebbecke,jonas.witt}@rwth-aachen.de

Abstract. 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 interested theorists and practitioners alike. 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 improving the algorithm's performance.

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 non-trivial subset, thus $\delta(S) \neq \emptyset$. Two cuts $\delta_1 \subseteq E$ and $\delta_2 \subseteq E$ are *disjoint*, if $\delta_1 \cap \delta_2 = \emptyset$. The *cut packing problem* is to find a largest cardinality set of pairwise disjoint cuts. The maximum is called the cut packing number $\gamma(G)$.

In combinatorial optimization, cut packing is known for its role in duality theorems [12,22]. It is NP-hard in general [6] and even in planar graphs [4], but polynomial time solvable in chordal or bipartite graphs [6] or when packing s-t cuts [7,22]. Interestingly, packing directed cuts in digraphs is polynomial time solvable as well [19]. Cut packing is closely related to other combinatorial optimization problems like cycle packing [4] and independent set [6]: An independent set $I = \{v_1, \ldots, v_k\} \subseteq V$ immediately translates to a (particular) cut packing $\{\delta(\{v_i\}) : v_i \in I\}$. The latter provides a strong inapproximability result [5]: 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 [5]. The parameterized version of the cut packing problem is W[1]-hard by a reduction from parameterized independent set [10], which is performed in the full version of this paper. Cut packing and variants have applications in bioinformatics [5] and network reliability [6]. Colbourn [7] mentions that the NP-hardness of several edge packing problems "limits their

* Supported by the German Research Foundation (DFG) as part of the Priority Program "Algorithm Engineering" under grants no. LU770/4-1 and LU770/4-2.

J. Gudmundsson and J. Katajainen (Eds.): SEA 2014, LNCS 8504, pp. 34–45, 2014.

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 one (minimum) cut in a graph. As trivially $\gamma(G) \leq m$, we replicate the constraints of such a formulation, one set for each *potential* cut in a packing, indexed by $c = 1, \ldots, m$. 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 $1 \in S_c$, $c = 1, \ldots, m$, 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 iff $ij \in \delta_c$, and the binary variables x^c are used to count the cuts in the packing. The cut packing problem can be formulated as

$$
\begin{aligned}
\max \quad & \sum_{c=1}^{m} x^c & & & \\
\text{s.t.} \quad & u_i^c - u_j^c + y_{ij}^c \geq 0 & & \forall ij \in E, \forall c \in \{1, \ldots, m\} & (1)\\
& u_j^c - u_i^c + y_{ij}^c \geq 0 & & \forall ij \in E, \forall c \in \{1, \ldots, m\} & (2)\\
& y_{ij}^c - u_i^c - u_j^c \leq 0 & & \forall ij \in E, \forall c \in \{1, \ldots, m\} & (3)\\
& u_i^c + u_j^c + y_{ij}^c \leq 2 & & \forall ij \in E, \forall c \in \{1, \ldots, m\} & (4)\\
& u_1^c = 0 & & \forall c \in \{1, \ldots, m\} & (5)\\
& x^c \leq \sum_{ij \in E} y_{ij}^c & & \forall c \in \{1, \ldots, m\} & (6)\\
& \sum_{c=1}^{m} y_{ij}^c \leq 1 & & \forall ij \in E & (7)\\
& x^c,\, u_i^c,\, y_{ij}^c \in \{0, 1\} & \quad \forall ij \in E, \forall c \in \{1, \ldots, m\}, \forall i \in V & .
\end{aligned}
$$

The metric inequalities (1)–(4) ensure compatibility between u and y variables for the c-th potential cut. Constraints (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, constraints (3) and (4) ensure that the edge between vertices on the same shore will not be in the cut. Constraint (5) puts vertex 1 in S_c and constraint (6) states that a cut is counted only iff there are edges assigned to that cut. Together with constraint (3), at least one other vertex is in $V \backslash S_c$. The packing constraint (7) ensures that each edge is contained in at most one cut.

2.2 Extended Formulation

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

$$\begin{aligned} \max \quad & \sum_{\delta \in D} x_\delta & & \\ \text{s.t.} \quad & \sum_{\delta \ni ij} x_\delta + \bar{x}_{ij} = 1 & \forall ij \in E & \qquad (8) \\ & x_\delta,\, \bar{x}_{ij} \in \{0,1\} & \forall ij \in E,\, \forall \delta \in D \ . & \end{aligned}$$

The binary slack variable $\bar{x}_{ij}$ for edge $ij \in E$ attains value 1 iff ij is not contained in any cut in the packing. This model (in packing form) was presented in [5]. We remark that it formally results from a Dantzig-Wolfe reformulation of our compact formulation by keeping constraint (7) in the master problem and reformulating constraints (1)–(6) into m identical subproblems which are then aggregated into a single subproblem, thereby completely eliminating the symmetry from the compact formulation.

3 Algorithmic Ingredients

In this section we describe the components of a full branch-price-and-cut algorithm to solve the extended formulation to integer optimality. As D grows exponentially, the linear programming (LP) relaxation of formulation (8) needs to be solved by column generation. That is, we include a small set of variables in a restricted master problem and price positive reduced cost variables using an auxiliary optimization problem, the pricing problem. If no more positive reduced variables can be found, the relaxation is proven to be optimal. Variables are also referred to as columns in this context. See [9] for a thorough introduction to the topic.

3.1 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. This pricing problem amounts to solving $\max_{\delta \in D}(1 - \pi^T y^\delta) = 1 - \min_{\delta \in D}(\pi^T y^\delta)$, where $\pi = (\pi_{ij})_{ij \in E}$ denotes the current dual solution corresponding to equation (8) and

$y^\delta = (y^\delta_{ij})_{ij \in E}$ is a binary vector indicating whether a given edge $ij \in E$ belongs to cut $\delta \in D$.

In the resulting *minimum* cut problem, the dual variables π_{ij} are free, we thus face a min-cut problem with arbitrary edge weights. We remark however, that all dual variables at the root node are non-negative as the variables $\bar{x}_{ij} \geq 0$ can be interpreted as slack variables. 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 [25] and then generalized to hypergraphs in [17]. In case some dual variables are negative (which happens because of branching, see below), we need to solve the pricing problem as a binary program, as this results in a *maximum* cut problem, which is NP-hard. The binary program arises from the compact formulation by leaving out constraints (7), omitting the index c, and replacing the objective function by the reduced cost function for the potential cut.

Desrochers et al. remark that pricing disjoint columns helps with arriving at integral solutions [8]. 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 in the tree resembles a max-cut problem, we employ further heuristics to price out favorable columns based on max-cut heuristics [24] inspired by the formulation of the problem. Columns that are not immediately added to the restricted master problem were collected in a column pool which was searched for positive reduced cost columns before calling any pricing algorithm.

3.2 Branching

As the restricted master problem is the linear programming relaxation of the formulation, we need to use branching schemes in order to find optimal integral solutions. We employ two different branching rules: First we try to branch on original y_{ij} variables and if that is not possible, we will use Ryan-Foster branching [23] to branch on pairs of edges.

In the first case, given an LP solution $x^\star$ of the master problem, we can calculate the value of variables y_{ij} for each edge $ij \in E$ as $y_{ij} = \sum_{\delta : ij \in \delta} x^\star_\delta = 1 - \bar{x}^\star_{ij}$, we can 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 contains edge ij or exactly one cut going through this edge ij. The first case can be respected directly in the pricing by setting $y_{ij} = 0$ or by solving a combinatorial algorithm on a restricted graph where ij is contracted. The remaining dual variables stay nonnegative in this case which is beneficial since we can still use a combinatorial algorithm. In the second case, the dual variables corresponding to edge ij can be negative which we cannot respect in a 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 $y_{ij} = 1 - \bar{x}_{ij}$ is integral for all edges $ij \in E$, we can branch on pairs of edges ij and kl, analogous to Ryan-Foster [23] branching. The branching decisions are: Either $\bar{x}_{ij} = \bar{x}_{kl}$, (this is called the *same* branch) or $\bar{x}_{ij} + \bar{x}_{kl} \geq 1$ (the *diff* branch). In the *same* branch, the two edges either must appear together in a cut or neither of the edges must be part of a cut. In the *diff* branch, not both edges are allowed to be in the same cut. These conditions can easily be respected in the binary program. 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 jl (and il and jk, resp.). Unfortunately, this is only feasible for a limited number 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. This follows by a reduction from MONOTONE 1-in-3 SAT [11], which is performed in the full version of this paper.

After enforcing the branching decisions, the set $\emptyset \neq S \subsetneq V$ to 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 can exchange the roles of S and $V \backslash S$). While adding this cut to the restricted master problem can potentially slow down the solution process as it will not be part of an optimal solution, this causes no further harm. In some cases, a special handling of these cuts can improve the performance which we cover in detail in the full version of this paper.

3.3 Cutting Planes

The optimality gap between the LP relaxation of the extended formulation and an optimal integral solution can be arbitrarily bad: The cardinality of an optimal cut packing for a clique $K = (V_K, E_K)$ with n nodes is 1 whereas the optimal LP solution value will be $\frac{n}{2}$, with cuts separating each vertex from the remaining of the clique. The associated master variables all take the value $\frac{1}{2}$. To improve the LP bound, we thus separate clique inequalities using the maximum weighted clique heuristic based on an algorithm by Borndörfer and Kormos [3]. This is readily available in the solver framework we use, which is `SCIP`.

For the column generation algorithm, we need to respect the dual variables introduced when separating the cutting planes in the master problem. Given a graph $G = (V, E)$ and a set of cliques $\mathcal{K}$, we can construct a hypergraph $H = (V, F)$ where original edges are copied and all cliques are converted to hyperedges. Formally, $F = \{e \in E : e \notin E_K \, \forall K \in \mathcal{K}\} \cup \{V_K : K \in \mathcal{K}\}$ and the dual values from the clique cuts can be transferred to the hyperedges.

In the root node, we can use the hypergraph extension of the Stoer-Wagner algorithm to solve the minimal cut problem. In the binary programming pricing case, the transformation is straightforward by adding new variables and constraints to the pricing problem. This is particularly described in the full version of this paper.

3.4 Combinatorial Dual Bounds

In order to further strengthen the dual bound, we also consider combinatorial upper bounds. An *edge clique cover* $\mathcal{Q}$ is a set of cliques with $\bigcup_{K \in \mathcal{Q}} E_K = E$. Finding an edge clique cover of minimum cardinality is NP-hard [18]. In an integral cut packing, at most one cut can be active per clique. Thus, the cut packing number $\gamma(G)$ is bounded from above by the number of cliques in an edge clique cover of minimum cardinality. We solve the edge clique covering problem approximately using the algorithm in [13] in order to obtain a valid upper bound. We calculate an edge clique cover at the beginning and add all induced clique cuts to the master problem before starting the column generation algorithm.

3.5 Primal Heuristics

Besides using the generic heuristics included in `SCIP`, we try to find good cut packings by using a (noncrossing) maximum s-t path cut packing heuristic [6] which we will call *maxpath*. Furthermore, we use an approximation algorithm for the independent set [16] for degree bounded graphs and transfer the resulting solution $I = \{v_1, \ldots, v_k\}$ to a solution $S = \{\delta(\{v_i\}) : v_i \in I\}$ for the cut packing problem by sorting the nodes according to non-decreasing degree and sequentially adding the next available node to the independent set. In order to assess the quality of the heuristic we calculate a maximum independent set exactly by solving the textbook formulation $\max\{\sum_{i \in V} x_i : x_i + x_j \leq 1 \, \forall ij \in E, \, x_i \in \{0,1\} \, \forall i \in V\}$.

4 Computational Setup and Results

We implemented the branch-price-and-cut algorithm in `SCIP` 3.0.1 [1] with `CPLEX` 12.4.0.1 as LP-solver. 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 3600 seconds unless stated otherwise.

We applied our approach to instances from the 10th DIMACS implementation challenge [2]. 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 [26] and investigate 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.

We further generated smaller random graphs using several graph generators such as `Rudy` by Rinaldi [21], `Randgraph` by Pettie and Ramachandran [20] and `NetworkX` 1.7 by Hagberg et al. [15]. Moreover, we used the `MUSKETEER` graph generator [14] to obtain a set of graphs which 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 vs. Extended Formulation

In Fig. 1, we compare the integrality gaps and the running times of the compact and the extended formulation on a test set generated with the `Randgraph` graph generator. A generated graph *graph-n-d* consists of a random tree on n nodes plus $\max\{0, d\% \cdot \frac{n \cdot (n-1)}{2} - n + 1\}$ additional randomly selected edges. In Fig. 1, the *all* column comprises the implementation with all presented features, in *basic*, only the bare column generation implementation is visualized.

When looking at the time needed to solve the instances to optimality, the extended formulation outperforms the compact one on almost every instance. The difference in solution time increases further if all presented features are added. In particular, the compact formulation was only able to solve 2 out of 12 instances to optimality. The basic implementation of the reformulation however solved 8 out of 12 instances and with all features turned on, all instances can be solved to optimality within the time limit.

If we look at the integrality gaps at the root node, we see 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.

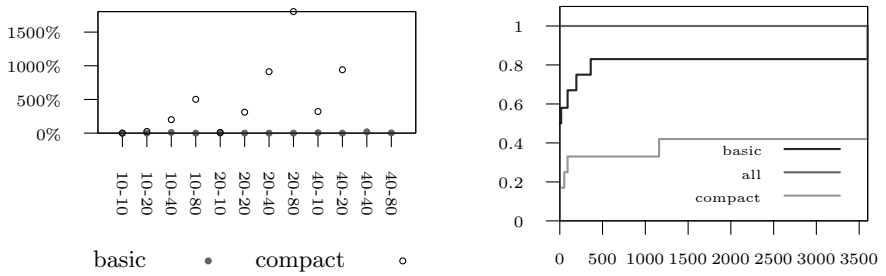

Fig. 1. Relative gap between primal and dual bounds at the root node for the compact and extended formulations for random graphs of type *graph-n-m* (left) and a performance profile for all the different formulations and settings (right) showing in how many instances (y-axis in percent) an algorithm is at most x times slower (factor on the x-axis).

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) 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 nodes, 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 nodes and edge densities of 10, 20, 40, and 80 percent. We only generated chordal graphs up to 160 nodes as using `NetworkX` to find all chordless cycles in larger graphs was too expensive. The results are summarized in Table 1.

Table 1. 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.

Type	Nodes												mean time
	10		20		40		80		160		320		
	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	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

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 nodes, resp. 160 nodes, in contrast to general random graphs where we fail to solve instances larger than 40 nodes.

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 nodes or less and all coloring instances from [26] with less than 5000 edges. We also tried to solve the problem on instances with hidden optimal solutions for classical graph problems [27], 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. These graphs have the same number of nodes and edges. The random graphs were generated with `Randgraph`, a connected graph *rand-n-m* consists of n nodes and m edges. In order to create similar graphs, we used `MUSKETEER` where we edited 7.5% of the edges at level 2, 15% of the edges at level 1, and 30% of the level 0 of the coarsening phases. The new graphs, forced to be connected, are denoted by suffix *-m*. The results on these graphs are compared in Table 2.

We observe that our algorithm solved 5 out of 9 real-world instances to optimality, but only 1 out of 9 of the corresponding random graphs. In contrast, the graphs edited by `MUSKETEER` are more similar to the original graphs and indeed our algorithm performs better on those than on arbitrary random graphs.

Table 2. 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%

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 the 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 Fig. 2.

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 cliques during run time, 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 bare column generation procedure. In our case, the combinatorial pricing is a nice add-on but has, nonetheless not that much influence on the solution time as expected.

4.4 Relation to Independent Set

Motivated by the close connection to maximum independent set (MIS) we want to study the relationship between an MIS and an optimal cut packing on selected instances. For a given graph, we therefore compare the solution values of a greedy algorithm to approximate the cardinality of an MIS, the cardinality $\alpha(G)$ of an optimal MIS by using an exact integer program based algorithm 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 Fig. 3.

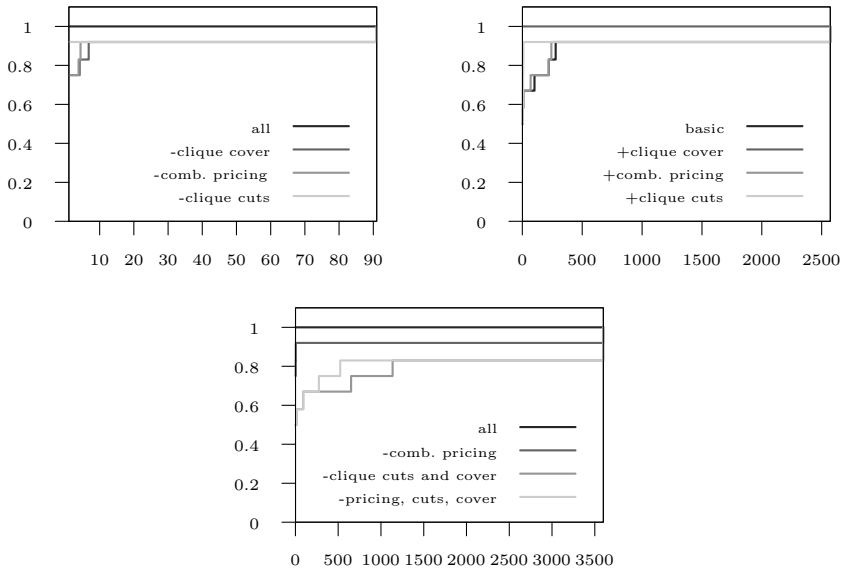

Fig. 2. Performance profiles for clique cover, clique separator, and combinatorial algorithm showing in how many instances (y-axis in percent) a given setting is at most x times slower (factor on the x-axis). The plot on the upper left states the influence of disabling each part of the algorithm, on 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.

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 the 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 and assessed its performance on about 100 random and benchmark graphs from the 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 5000 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 $2\alpha(G) - 1$.

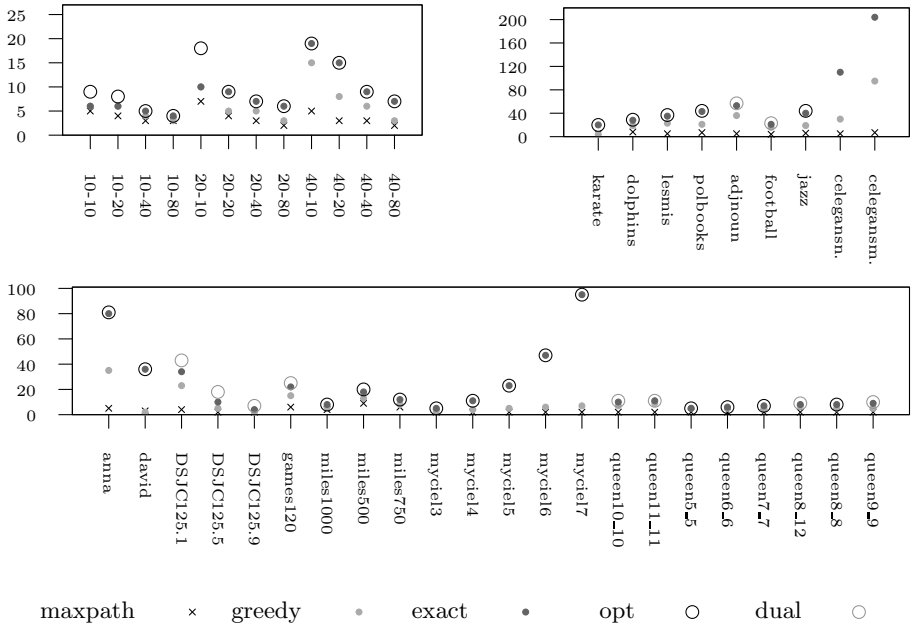

Fig. 3. Objective function values of max path heuristic, greedy and exact MIS, optimal cut packing or dual bound for random graphs (top left), DIMACS partitioning benchmark graphs (top right), and coloring graphs (bottom)

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 have exploited the problem's close relation to the independent set problem e.g., for designing primal heuristics. Future computational research on the problem should clarify whether it pays to work on (integer programming formulations for) better clique edge covers for stronger dual bounds; and whether different formulations for min-cut can speed up solving the pricing problem. In addition, one should fully exploit the known branch-and-price machinery like dual variable stabilization.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Programming Comp. 1(1), 1–41 (2009)
2. Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D. (eds.): Graph Partitioning and Graph Clustering. 10th DIMACS Implementation Challenge Workshop, February 13-14, 2012. Contemp. Mathematics, vol. 588. American Mathematical Society (2013)
3. Borndörfer, R., Kormos, Z.: An algorithm for maximum cliques. unpublished working paper, Konrad-Zuse-Zentrum für Informationstechnik Berlin (1997)

4. Caprara, A., Panconesi, A., Rizzi, R.: Packing cycles in undirected graphs. J. Algorithms 48, 239–256 (2003)
5. Caprara, A., Panconesi, A., Rizzi, R.: Packing cuts in undirected graphs. Networks 44(1), 1–11 (2004)
6. Colbourn, C.J.: The Combinatorics of Network Reliability. Oxford University Press, New York (1987)
7. Colbourn, C.: Edge-packing of graphs and network reliability. Discrete Math 72(1-3), 49–61 (1988)
8. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. 40(2), 342–354 (1992)
9. Desrosiers, J., Lübbecke, M.: Branch-price-and-cut algorithms. In: Cochran, J. (ed.) Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Chichester (2011)
10. Downey, R., Fellows, M.: Fixed-parameter tractability and completeness II: On completeness for W(1). Theoretical Computer Science 141(12), 109–131 (1995), http://www.sciencedirect.com/science/article/pii/0304397594000973
11. Fox-Epstein, E.: Forbidden Pairs Make Problems Hard. Bachelor's thesis. Wesleyan University (2011)
12. Fulkerson, D.: Blocking and anti-blocking pairs of polyhedra. Math. Programming 1, 168–194 (1971)
13. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Data reduction, exact, and heuristic algorithms for clique cover. In: Proc. 8th ALENEX, pp. 86–94 (2006)
14. Gutfraind, A., Meyers, L.A., Safro, I.: Multiscale network generation, arXiv:1207.4266 (2012)
15. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using NetworkX. In: Proc. of the 7th Python in Science Conference (SciPy 2008), Pasadena, pp. 11–15 (2008)
16. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: Approximating independent sets in sparse and bounded-degree graphs. Algorithmica 18(1), 145–163 (1997)
17. Klimmek, R., Wagner, F.: A simple hypergraph min cut algorithm. Tech. Rep. B 96-02. FU Berlin (1996)
18. 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)
19. Lucchesi, C., Younger, D.: A minimax theorem for directed graphs. J. Lond. Math. Soc. 17, 369–374 (1978)
20. Pettie, S., Ramachandran, V.: Randgraph graph generator (2006), http://www.dis.uniroma1.it/challenge9/download.shtml
21. Rinaldi, G.: Rudy, a graph generator (1998), http://www-user.tu-chemnitz.de/~helmberg/sdp_software.html
22. Robacker, J.T.: Min-max theorems on shortest chains and disjunct cuts of a network. Tech. Rep. RM-1660-PR. Rand Corporation (1956)
23. Ryan, D.M., Foster, B.A.: An integer programming approach to scheduling. Opt. Res. Q. 27(2), 367–384 (1976)
24. Sahni, S., Gonzalez, T.: P-complete approximation problems. J. ACM 23(3), 555–565 (1976)
25. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM 44(4), 585–591 (1997)
26. Trick, M.: Coloring instances (1993), http://mat.gsia.cmu.edu/COLOR/instances.html
27. Xu, K.: Bhoslib: Benchmarks with hidden optimum solutions for graph problems (2010), http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm