A Branch-and-Bound Algorithm for the Knapsack Problem with Conflict Graph

Andrea Bettinelli, Valentina Cacchiani, Enrico Malaguti

DEI, Università di Bologna, Viale Risorgimento 2,
40136 Bologna, Italy
{andrea.bettinelli, valentina.cacchiani, enrico.malaguti}@unibo.it

Abstract

We study the Knapsack Problem with Conflict Graph (KPCG), an extension of the 0-1 Knapsack Problem, in which a conflict graph describing incompatibilities between items is given. The goal of the KPCG is to select the maximum profit set of compatible items while satisfying the knapsack capacity constraint. We present a new Branch-and-Bound approach to derive optimal solutions to the KPCG in short computing times. Extensive computational experiments are reported, showing that the proposed method outperforms a state-of-the-art approach and Mixed Integer Programming formulations tackled through a general purpose solver.

Keywords: Knapsack Problem, Maximum Weight Stable Set Problem, Branch-and-Bound, Combinatorial Optimization, Computational Experiments.

1 Introduction

The *Knapsack Problem with Conflict Graph* (KPCG) is an extension of the NP-hard 0-1 Knapsack Problem (0-1 KP, see Martello and Toth [17]) where incompatibilities between pairs of items are defined. A feasible KPCG solution cannot include pairs of incompatible items; in particular, a conflict graph is given, which has one vertex for each item and one edge for each pair of items that are incompatible. The KPCG is also referred to as *Disjunctively Constrained Knapsack Problem*.

Formally, in the KPCG, we are given a knapsack with capacity c and a set of n items, each one characterized by a positive profit p_i and a positive weight w_i $(i = 1, \dots, n)$. In addition, we are given an undirected *conflict graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each vertex $i \in \mathcal{V}$ corresponds to an item (i.e., $n = |\mathcal{V}|$) and an edge $(i, j) \in \mathcal{E}$ denotes that items i and j cannot be packed together. The goal is to select the maximum profit subset of items to be packed into the knapsack, while satisfying the capacity and the incompatibility constraints. The KPCG can also be seen as an extension of the NP-hard Maximum Weight Stable Set Problem (MWSSP), in which one has to find a maximum profit stable set of $\mathcal{G}$. The KPCG generalizes the MWSSP by defining weights for the vertices of $\mathcal{G}$ and by imposing a capacity constraint on the set of selected vertices.

Without loss of generality, we assume that $\sum_{i=1,\dots,n} w_i > c$ and that $w_i \leq c$ $(i = 1, \dots, n)$. We also assume that items are sorted by non increasing profit-over-weight ratio, i.e.

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}.$$

Some definitions will be useful in the following. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a clique $C \subseteq \mathcal{V}$ is a subset of the vertices that induces a complete subgraph. A stable set $S \subseteq \mathcal{V}$ is a subset of pairwise non-adjacent vertices. The density of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as the ratio between $|\mathcal{E}|$ and the cardinality of the edge set of the complete graph having the same number of vertices.

In this paper, we propose a Branch-and-Bound algorithm to derive optimal solutions of the KPCG. The algorithm consists of a new upper bounding procedure, which takes into account both the capacity constraint and the incompatibilities between items, and a new branching strategy, based on optimally presolving the 0-1 KP by dynamic programming while neglecting the conflicts between items. The proposed Branch-and-Bound algorithm is tested on a large set of randomly generated instances, with correlated and uncorrelated profits and weights, having conflict graph densities between 0.1 and 0.9. Its performance is evaluated by comparison with a state-of-the-art approach from the literature, and with a Mixed Integer Programming formulation (MIP) solved through a general purpose solver.

The paper is organized as follows. Section 1.1 reviews exact and heuristic algorithms proposed for the problem. In Section 1.2, we present two standard MIP formulations of the problem. Sections 2.1 and 2.2 are devoted to the description of the proposed branching rule and upper bound. In Section 3, we present the computational results obtained by the proposed Branch-and-Bound algorithm, and its comparison with other solution methods.

1.1 Literature review

The KPCG has been introduced by Yamada et al. [20]. In their paper a greedy algorithm, enhanced with a 2-opt neighborhood search, as well as a Branch-and-Bound algorithm, which exploits an upper bound obtained through a Lagrangian relaxation of the constraints representing incompatibilities between items, are proposed. The algorithms are tested on randomly generated instances with up-to 1000 items and very sparse conflict graphs (densities range between 0.001 and 0.02). The profits and the weights are assumed uncorrelated (random and independent, ranging between 1 and 100). Hifi and Michrafy [12] present several versions of an exact algorithm for the KPCG which solves a MIP model. A starting lower bound is obtained through a heuristic algorithm, then, reduction strategies are applied, which fix some decision variables to their optimum values, based on the current lower bound value. Finally, the reduced problem is solved by a Branch-and-Bound algorithm. Improved versions apply dichotomous search combined with the reduction strategies and different ways of modeling the conflict constraints. The three versions of the algorithm are tested on instances with 1000 items and very sparse conflict graphs (densities range between 0.007 and 0.016).

Several heuristic methods have been proposed in the literature. Hifi and Michrafy [11] present a reactive local search based algorithm. It consists in determining a feasible solution by applying a greedy algorithm, and improving it

by a swapping procedure and a diversification strategy. The algorithm is tested on instances randomly generated by the authors by following the generation method of [20]. In particular, they consider 20 instances with 500 items and capacity equal to 1800, with conflict graph densities between 0.1 and 0.4. In addition they consider 30 larger correlated instances with 1000 items and capacity equal to 1800 or 2000, with conflict graph densities between 0.05 and 0.1. Akeb et al. [1] investigates the use of local branching techniques (see Fischetti and Lodi [7]) for approximately solving instances of the KPCG. The algorithm is tested on the instances proposed in [11]. Hifi and Otmani [13], propose a scatter search metaheuristic algorithm. The algorithm is tested on the instances proposed in [11] and compared with the algorithm therein presented, showing that it is able to improve many of the best known solutions from the literature. Recently, an iterative rounding search based algorithm has been proposed in Hifi [10]. The algorithm is tested on the instances from [11] and compared with the algorithm proposed in [13]. The results show that it is able to improve many solutions from the literature.

Pferschy and Schauer [18] present algorithms with pseudo-polynomial time and space complexity for two special classes of conflict graphs: graphs with bounded treewidth and chordal graphs. Fully polynomial-time approximation schemes are derived from these algorithms. In addition, it is shown that the KPCG remains strongly NP-hard for perfect conflict graphs.

The interest in the KPCG is not only as a stand-alone problem, but also as a subproblem arising in the solution of other complex problems, such as the Bin Packing Problem with Conflicts (BPPC) (see e.g., Gendreau et al. [8], Fernandes-Muritiba et al. [6], Elhedhli et al. [4] and Sadykov and Vanderbeck [19]). In this case, the KPCG corresponds to the pricing subproblem of a column generation based algorithm and is solved several times. In this context, it is therefore fundamental to develop a computationally fast algorithm for the KPCG. In Fernandes-Muritiba et al. [6], the KPCG is solved by a greedy heuristic and, if the latter fails to produce a negative reduced cost column, the general purpose solver CPLEX is used to solve to optimality the KPCG formulated as a MIP. Elhedhli et al. [4] solve the MIP formulation of the KPCG by CPLEX, after strengthening it by means of clique inequalities. In Sadykov and Vanderbeck [19], an exact approach for the KPCG is proposed. They distinguish between interval conflict graphs, for which they develop a pseudo-polynomial time algorithm based on dynamic programming, and arbitrary conflict graphs, for which they propose a depth-first Branch-and-Bound algorithm. The latter algorithm is taken as reference in our computational experiments (Section 3).

1.2 MIP models

This section presents two standard MIP models for the KPCG, where binary variables x_i $(i = 1, \ldots, n)$ are used to denote the selection of items. These models are solved by a general purpose MIP solver and compared to the Branch-and-Bound algorithm we proposed (see Section 3).

$$\text{Maximize} \sum_{i=1,\dots,n} p_i x_i \tag{1a}$$

$$\text{s.t.} \sum_{i=1,\dots,n} w_i x_i \leq c \tag{1b}$$

$$x_i + x_j \leq 1 \qquad (i,j) \in \mathcal{E} \tag{1c}$$

$$x_i \in \{0,1\} \qquad i = 1,\dots,n. \tag{1d}$$

The objective (1a) is to maximize the sum of the profits of the selected items. Constraint (1b) requires not to exceed the capacity of the knapsack. Constraints (1c) impose to choose at most one item for each conflicting pair, represented by edges of the conflict graph. Finally, constraints (1d) require the variables to be binary.

Let $\mathcal{C}$ be a family of cliques on $\mathcal{G}$, such that, for each edge $(i,j) \in \mathcal{E}$, vertices (items) i and j belong to some clique $C \in \mathcal{C}$. A MIP model, equivalent to (1a)-(1d) but having a stronger (i.e., smaller) LP-relaxation bound, is the following (see, e.g., Malaguti et al. [16] for a discussion on the heuristic strengthening of edge constraints to clique constraints):

$$\text{Maximize} \sum_{i=1,\dots,n} p_i x_i \tag{2a}$$

$$\text{s.t.} \sum_{i=1,\dots,n} w_i x_i \leq c \tag{2b}$$

$$\sum_{i \in C} x_i \leq 1 \qquad C \in \mathcal{C} \tag{2c}$$

$$x_i \in \{0,1\} \qquad i = 1,\dots,n. \tag{2d}$$

Models (1a)-(1d) and (2a)-(2d) will be taken as reference in our computational experiments (Section 3). In our computational experiments, we generated $\mathcal{C}$ using the following heuristic procedure: we iteratively select a random uncovered edge (i,j) and build a maximal clique containing it, until all edges are covered by at least a clique.

2 Branch-and-Bound algorithm

A Branch-and-Bound algorithm is based on two main operations: *branching*, that is, dividing the problem to be solved in smaller subproblems, in such a way that no feasible solution is lost; and *bounding*, that is, computing an upper bound (for a maximization problem) on the optimal solution value of the current subproblem, so that eventually the subproblem can be fathomed. In the Branch-and-Bound algorithm that we propose, a subproblem (associated with a node in a Branch-and-Bound exploration tree) is defined by:

- a partial feasible solution to the KPCG, i.e., a subset $S \in \mathcal{V}$ of items inserted in the knapsack such that S is a stable set and the sum of the weights of the items in S does not exceed the capacity c;

- a set of available items F, i.e., items that can enter the knapsack and for which a decision has not been taken yet.

In Sections 2.1 and 2.2, we present, respectively, the branching rule and the upper bound procedure embedded in the proposed Branch-and-Bound algorithm. In Figure 1 we introduce the notation we need for the description of these procedures.

- S: set of items inserted in the knapsack in a given subproblem, i.e., node of the Branch-and-Bound tree;
- F: set of *free* items in a given subproblem, i.e., items that can enter the knapsack and for which a decision has not been taken yet;
- $p(V)$: sum of the profits of the items in the set $V \subseteq \mathcal{V}$;
- $w(V)$: sum of the weights of the items in the set $V \subseteq \mathcal{V}$;
- $KP(V, \bar{c})$: value of the integer optimal solution of the 0-1 KP with capacity $\bar{c}$ on the subset of items V;
- $\alpha_p(G(V))$: value of the integer optimal solution of the MWSSP with weight vector p on the subgraph induced on the conflict graph G by the subset of vertices (items) V;
- LB: current global lower bound;
- $N(i)$: the set of vertices in the neighborhood of vertex i.

Figure 1: Summary of the used notation.

2.1 Branching scheme

The branching rule we propose consists of an improvement of the one used in Sadykov and Vanderbeck [19], which is derived from the scheme proposed by Carraghan and Pardalos [2] for the MWSSP. For sake of clarity, we briefly review the branching scheme used in [2, 19] and then describe the improvement we propose. In addition, we discuss a possible adaptation to the KPCG of the branching rule used by Held et al. [9] for the MWSSP.

Given S and F, at a node of the Branch-and-Bound tree, let $p(S)$ and $w(S)$ be the sum of the profits and of the weights, respectively, of the items in the set S. The idea of the branching in [2, 19] is that a new subproblem has to be generated for each item $i \in F$, in which the corresponding item i is inserted in the knapsack, unless this choice would produce an upper bound not better than the current best known solution value LB. More in detail, items $i \in F$ are sorted by non increasing profit-over-weight ratio (p_i/w_i). Iteratively, the next item $i \in F$ in the ordering is considered, and an upper bound UB_i on the maximum profit which can be obtained by inserting i in the knapsack is computed as $UB_i = p(S) + (c - w(S))(p_i/w_i)$. UB_i is the profit which is

obtained by using the residual capacity of the knapsack for inserting multiple (fractional) copies of item i. If $UB_i > LB$, a new child node is created with $\bar{S} = S \cup \{i\}$, $\bar{F} = F \setminus (\{j \in F, j \leq i\} \cup \{j \in F, j \in N(i)\})$, where the condition $j \leq i$ is evaluated according to the ordering. As soon as $UB_i \leq LB$, the iteration loop is stopped since the next items in F would not generate a solution improving on LB, once inserted in the knapsack.

The idea we propose is to enhance this branching scheme by improving the value of the upper bound UB_i, at the cost of limited additional computational effort. As it will be seen in Section 3, this improvement helps to significantly reduce the number of explored nodes and, consequently, the total computing time. Let $KP(V, \bar{c})$ be the value of the optimal integer solution of the 0-1 KP with capacity $\bar{c}$ on the subset of items V. Recalling that items are ordered by non increasing profit-over-weight ratio, in a preprocessing phase we compute $KP(V, \bar{c})$ for all $i = 1, \ldots, n$, $V = \{i, \ldots, n\}$, $\bar{c} = 0, \ldots, c$. This can be done in $O(nc)$ by using a Dynamic Programming algorithm.

When branching, we iterate on the items $i \in F$ sorted by non increasing p_i/w_i ratio. The next item $i \in F$ in the ordering is considered and F is updated to $F = F \setminus \{i\}$. An upper bound on the maximum profit which can be obtained by considering items in $F \cap \{i, \ldots, n\}$ is computed as $\tilde{UB}_i = p(S) + KP(\{i, \ldots, n\}, c - w(S))$. $KP(\{i, \ldots, n\}, c - w(S))$ is the profit which is obtained by optimally solving the knapsack problem with the items in $\{i, \ldots, n\}$ and the residual capacity, and by disregarding the conflicts.

It is easy to see that, given F, $\tilde{UB}_i \leq UB_i$ for each $i \in F$. Thus the branching rule based on optimally presolving the KP for all the residual capacities would possibly fathom nodes that are instead generated by the rule proposed in [19]. We denote the proposed branching rule as ${}_{pre}KP$.

In Figure 2 we report the pseudocode for a the generic Branch-and-Bound algorithm BB based on the scheme from [19] or on the improvement described in this section. The algorithm receives in input sets S, F and the value of the incumbent solution LB, and includes a call to a generic upper bounding procedure ($UpperBound()$) for evaluating the profit which can be obtained from items in F and the residual capacity $c - w(S)$. Several upper bounding procedures can be used within this scheme. In Section 2.2, we present the upper bounding procedure we propose. The BB algorithm can be started by invoking $BB(\emptyset, \{1, \ldots, n\}, 0)$.

Stable set-based branching. We conclude this section by briefly discussing the branching rules used in the Branch-and-Bound algorithm presented in Held et al. [9] for the MWSSP. These rules can be extended to deal with the KPCG. Let $F' \subseteq F$ be a subset of the free items such that $\alpha_p(G(F')) + p(S) \leq LB$, where $\alpha_p(G(F'))$ denotes the value of the integer optimal solution of the MWSSP with weight vector p on the subgraph induced on the conflict graph G by the subset F' of vertices (items). The idea is that, to improve on the current LB, at least one of the items in $F \setminus F'$ must be used in a solution of the Branch-and-Bound tree descending from a node associated with a set S. Therefore, one can branch by generating from the node one child node for each item in $F \setminus F'$. In Held et al. [9] interesting ideas on how to compute F' given the conflict graph G are presented, e.g., one possibility is to compute an upper bound on $\alpha_p(G(F'))$ by

```
//items are ordered by non increasing profit-over-weight ratio
BB(S, F, LB)
if LB < p(S) then
    LB = p(S)
end
compute UpperBound(F, c − w(S))
if UpperBound(F, c − w(S)) + p(S) ≤ LB then
    return
end
for i ∈ F do
    compute UB_i
    if UB_i > LB then
        F = F \ {i}
        BB(S ∪ {i}, F \ N(i), LB)
    end
    else
        break
    end
end
```

Figure 2: Generic Branch-and-Bound scheme for the KPCG.

exploiting the weighted clique cover bound (see Section 2.2). Similarly, pruning rules presented in Held et al. [9] can be extended to the KPCG. We adapted and tested these ideas for the KPCG. Even though they are very effective for the MWSSP, our preliminary computational experiments showed that this extension is not effective for the KPCG.

2.2 Upper bound

In this section we describe the upper bounding procedure we propose for the KPCG. It is based on considering the KPCG as an extension of the MWSSP. In the MWSSP, we have a conflict graph $\mathcal{G}$ and assign to each vertex $i \in \mathcal{V}$ a weight equal to the profit p_i of the corresponding item i. In addition, each vertex is assigned a second weight, called *load* in the following, which corresponds to the item weight w_i. The goal is to determine a stable set of $\mathcal{G}$ of maximum weight (profit), while satisfying the capacity constraint (i.e. the sum of the loads of the vertices in the stable set is smaller or equal to the available capacity). By neglecting the capacity constraint, upper bounds for the MWSSP are valid upper bounds for the KPCG as well.

Held et al. [9] proposed the *weighted clique cover bound* for MWSSP. The upper bounding procedure we propose extends the weighted clique cover bound by taking into account constraints on the item loads. As it will be evident from the computational experiments (see Section 3), it is fundamental to consider these constraints in order to derive good upper bounds for the KPCG. In the following, we briefly describe the weighted clique cover bound, and then explain how we extend it.

$p'_i := p_i \quad \forall i \in \mathcal{V}$
$r := 0$
while $\exists i \in \mathcal{V} : \ p'_i > 0$ **do**
  $\bar{i} := \text{argmin}\{p'_i : p'_i > 0, i \in \mathcal{V}\}$
  $r := r + 1$
  Find a clique $K_r \subseteq \{j \in N(\bar{i}) : p'_j > 0\}$
  $K_r := K_r \cup \{\bar{i}\}$
  $\Pi_r := p'_{\bar{i}}$
  $p'_j := p'_j - p'_{\bar{i}} \quad \forall j \in K_r$
end

Figure 3: Weighted clique cover algorithm of Held et al. [9].

Weighted clique cover bound [9] A *weighted clique cover* is a set of cliques $K_1, K_2, \ldots, K_R$ of $\mathcal{G}$, each with an associated positive weight Π_r ($r = 1, \ldots, R$), such that, for each vertex $i \in \mathcal{V}$, $\sum_{r=1,\ldots,R:i\in K_r} \Pi_r \geq p_i$. The weight of the clique cover is defined as $\sum_{r=1}^{R} \Pi_r$. The *weighted stability number* (i.e. the optimal solution value of MWSSP without capacity constraint) is less or equal to the weight of any clique cover of $\mathcal{G}$ (see [9]). Therefore, the weight of any clique cover is an upper bound for the MWSSP and consequently it is an upper bound for the KPCG.

Determining the minimum weight clique cover is a NP-hard problem (see Karp [14]). Held et al. [9] propose a greedy algorithm to compute a weighted clique cover, summarized here for sake of clarity. Recall that $N(i)$ identifies the set of vertices in the neighborhood of vertex i. Initially the residual weight p'_i of each vertex i is set equal to its weight p_i (profit). Iteratively, the algorithm determines the vertex $\bar{i}$ with the smallest residual weight $p'_{\bar{i}}$, heuristically finds a maximal clique K_r containing $\bar{i}$, assigns weight $\Pi_r = p_{\bar{i}}$ to K_r, and subtracts Π_r from the residual weight of every vertex in K_r. The algorithm terminates when no vertex has positive residual weight. A sketch of the algorithm is reported in Figure 3. We denote the upper bound obtained by applying the weighted clique cover algorithm as CC.

Capacitated weighted clique cover bound The weighted clique cover bound previously described neglects the capacity constraint. This can lead to a weak upper bound for the KPCG when the capacity constraint is tight.

We propose a new upper bound, called *capacitated weighted clique cover bound*, which extends the weighted clique cover bound by taking into account the capacity constraint. The idea is to either find a weighted clique cover (in this case we obtain exactly the weighted clique cover bound) or to saturate the knapsack capacity *in the best possible way*. In the second case, we say that the weighted clique cover is *partial* (*complete* otherwise). To this aim, we associate with each clique K_r a load

$$W_r \leq \Pi_r \min_{j \in K_r} \left\{ \frac{w_j}{p_j} \right\}.$$

$p'_i := p_i \quad \forall i \in \mathcal{V}$
$r := 0$
while $p' \neq 0 \wedge \sum_{h=0}^{r} W_h < c$ **do**
  $\bar{i} := \text{argmin}\{\frac{w_i}{p_i} : p'_i > 0, i \in \mathcal{V}\}$
  $r := r + 1$
  Find a clique $K_r \subseteq \{j \in N(\bar{i}) : p'_j > 0\}$
  $K_r := K_r \cup \{\bar{i}\}$
  $t := \text{argmin}\{p'_t : p'_t > 0, t \in K_r\}$
  $W_r := \min\{p'_t \frac{w_{\bar{i}}}{p_{\bar{i}}}, c - \sum_{h=0}^{r} W_h\}$
  $\Pi_r := \min\{p'_t, \frac{W_r}{w_{\bar{i}}/p_{\bar{i}}}\}$
  $p'_j := p'_j - p'_t \quad \forall j \in K_r$
end

Figure 4: Capacitated weighted clique cover algorithm.

The load-over-weight ratio of the clique does not exceed the smallest ratio among all the vertices in the clique.

Having defined the load of a clique, the following result, for which a proof is given at the end of the section, holds:

Theorem 1. *The weight* $\sum_{K_r \in \mathcal{K}} \Pi_r$ *of a partial weighted clique cover* $\mathcal{K}$ *satisfying*

$$\sum_{K_r \in \mathcal{K}} W_r = c \tag{3}$$

and

$$\min_{K_r \in \mathcal{K}} \left\{ \frac{\Pi_r}{W_r} \right\} \geq \max_{j \in \mathcal{V}} \left\{ \frac{p_j}{w_j} : \sum_{K_r \in \mathcal{K}: j \in K_r} \Pi_r < p_j \right\}. \tag{4}$$

is a valid upper bound for the KPCG.

We define as *Minimum Weight Capacitated Clique Cover* the problem of finding a (possibly partial) clique cover of minimum weight that either: i) is a *complete* clique cover, or ii) satisfies (3) and (4).

Determining the minimum weight capacitated clique cover is NP-hard, since it is a generalization of the minimum weight clique cover. Therefore we developed a greedy algorithm, denoted as capacitated weighted clique cover algorithm, that iteratively builds a clique cover while satisfying (4). The algorithm is stopped as soon as (3) is satisfied or the clique cover is complete.

A sketch of the algorithm is reported in Figure 4. The algorithm iteratively determines the vertex $\bar{i}$ with the best (i.e., the smallest) load-over-weight ratio, and heuristically constructs a maximal clique K_r containing $\bar{i}$. The weight Π_r is subtracted from the weight of every vertex in K_r. The algorithm terminates when all the vertices have been covered according to their corresponding weights or when the capacity is saturated. We denote the upper bound obtained by applying the capacitated weighted clique cover algorithm as ${}_{cap}CC$.

The above algorithm has a worst case computational complexity of $O(n^3)$, realized when G is a complete graph, $c = \infty$ and all the profits are different.

The complexity decreases for sparser graphs, and becomes linear for a graph with empty edge set.

Proof of Theorem 1

We introduce the following notation for convenience, to be used in the proof of Theorem 1:

- $p(\mathcal{K}) := \sum_{K_r \in \mathcal{K}} \Pi_r$,
- $w(\mathcal{K}) := \sum_{K_r \in \mathcal{K}} W_r$.

Let S^* be an optimal solution of the KPCG. We partition S^* into $\bar{S}^*$, that denotes the set of items in the optimal solution that are fully covered by cliques in $\mathcal{K}$, and $\underline{S}^* = S^* \setminus \bar{S}^*$. Similarly, we partition $\mathcal{K}$ into $\bar{\mathcal{K}}$, that denotes the set of cliques containing one item in S^* (note that each clique in $\bar{\mathcal{K}}$ contains exactly one item of S^* because S^* does not contain conflicting items), and $\underline{\mathcal{K}} = \mathcal{K} \setminus \bar{\mathcal{K}}$.

Without loss of generality, we can assume that no item is overcovered, i.e. $\sum_{K_r \in \mathcal{K}} \Pi_r \leq p_i,\ i \in \mathcal{V}$. Indeed, given a (partial) clique cover where one (or more) item is overcovered, it is always possible to derive a (partial) clique cover with smaller or equal weight, in which no item is overcovered. Therefore, for all items in $\bar{S}^*$, since they are fully covered, it holds:

$$\sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} \Pi_r = p_i, i \in \bar{S}^*. \tag{5}$$

In order to prove Theorem 1, we need the following lemma.

Lemma 2. *$w(\underline{\mathcal{K}}) \geq w(\underline{S}^*)$.*

Proof. We first show that $w(\bar{\mathcal{K}}) \leq w(\bar{S}^*)$.

Let K_r be a clique in $\bar{\mathcal{K}}$ containing an item $i \in \bar{S}^*$. By definition of the clique weight: $W_r \leq \Pi_r \min_{h \in K_r} \left\{ \frac{w_h}{p_h} \right\}$, therefore: $W_r \leq \Pi_r \frac{w_i}{p_i}$.

By summing over all the cliques in $\bar{\mathcal{K}}$ that contain item i, we obtain:
$\sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} W_r \leq \frac{w_i}{p_i} \sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} \Pi_r$.

Since no item is overcovered and $i \in \bar{S}^*$, by applying (5), we get that $\sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} W_r \leq \frac{w_i}{p_i} p_i = w_i$.

Now let us sum over all items in $\bar{S}^*$: $\sum_{i \in \bar{S}^*} \sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} W_r \leq \sum_{i \in \bar{S}^*} w_i$.

By the definition of the partitions, and recalling that each clique in $\bar{\mathcal{K}}$ contains exactly one item in $\bar{S}^*$, we obtain that: $w(\bar{\mathcal{K}}) \leq w(\bar{S}^*)$.

Now, since S^* is an optimal solution of the KPCG, then it satisfies the capacity constraint. Therefore: $w(S^*) \leq c$. In addition, from (3), $w(\mathcal{K}) = c$. Then $w(\bar{S}^*) + w(\underline{S}^*) = w(S^*) \leq c = w(\mathcal{K}) = w(\bar{\mathcal{K}}) + w(\underline{\mathcal{K}})$.

Since $w(\bar{\mathcal{K}}) \leq w(\bar{S}^*)$, then $w(\underline{\mathcal{K}}) \geq w(\underline{S}^*)$.

□

We can now prove Theorem 1.

Proof. We want to show that $p(\mathcal{K}) \geq p(S^*)$, i.e. $p(\mathcal{K})$ is a valid upper bound for the KPCG. We will prove it by showing that: $p(\bar{\mathcal{K}}) = p(\bar{S}^*)$ and $p(\underline{\mathcal{K}}) \geq p(\underline{S}^*)$.

- $p(\bar{\mathcal{K}}) = p(\bar{S}^*)$

  From (5) we have: $\sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} \Pi_r = p_i$, $i \in \bar{S}^*$. By summing over all items in $\bar{S}^*$, we get: $\sum_{i \in \bar{S}^*} \sum_{K_r \in \bar{\mathcal{K}}: i \in K_r} \Pi_r = \sum_{i \in \bar{S}^*} p_i$.

  Taking into account that each clique contains exactly item of $\bar{S}^*$, we have: $p(\bar{\mathcal{K}}) = p(\bar{S}^*)$.

- $p(\underline{\mathcal{K}}) \geq p(\underline{S}^*)$

  $p(\underline{S}^*) = \sum_{i \in \underline{S}^*} p_i$; if we multiply and divide by w_i, and replace $\frac{p_i}{w_i}$ with $\max_{j \in \underline{S}^*} \left\{ \frac{p_j}{w_j} \right\}$, we have:

  $\sum_{i \in \underline{S}^*} p_i = \sum_{i \in \underline{S}^*} p_i \frac{w_i}{w_i} \leq \sum_{i \in \underline{S}^*} \max_{j \in \underline{S}^*} \left\{ \frac{p_j}{w_j} \right\} w_i = \max_{j \in \underline{S}^*} \left\{ \frac{p_j}{w_j} \right\} w(\underline{S}^*)$.

  Now, let us replace in (4) $\mathcal{K}$ with $\underline{\mathcal{K}}$ and $\mathcal{V}$ with $\underline{S}^*$: the inequality still holds since $\underline{\mathcal{K}} \subseteq \mathcal{K}$ and $\underline{S}^* \subseteq \mathcal{V}$:

  $\min_{K_r \in \underline{\mathcal{K}}} \left\{ \frac{\Pi_r}{W_r} \right\} \geq \max_{j \in \underline{S}^*} \left\{ \frac{p_j}{w_j} \right\}$.

  Then, we obtain:

  $\max_{j \in \underline{S}^*} \left\{ \frac{p_j}{w_j} \right\} w(\underline{S}^*) \leq \min_{K_r \in \underline{\mathcal{K}}} \left\{ \frac{\Pi_r}{W_r} \right\} w(\underline{S}^*)$.

  By applying Lemma 2, we obtain:

  $\min_{K_r \in \underline{\mathcal{K}}} \left\{ \frac{\Pi_r}{W_r} \right\} w(\underline{S}^*) \leq \min_{K_r \in \underline{\mathcal{K}}} \left\{ \frac{\Pi_r}{W_r} \right\} w(\underline{\mathcal{K}})$.

  By definition of the partitions and by replacing the minimum over $\underline{\mathcal{K}}$ with the corresponding sum:

  $\min_{K_r \in \underline{\mathcal{K}}} \left\{ \frac{\Pi_r}{W_r} \right\} w(\underline{\mathcal{K}}) = \min_{K_r \in \underline{\mathcal{K}}} \left\{ \frac{\Pi_r}{W_r} \right\} \sum_{K_h \in \underline{\mathcal{K}}} W_h \leq \sum_{K_h \in \underline{\mathcal{K}}} W_h \frac{\Pi_h}{W_h} = p(\underline{\mathcal{K}})$.

  By combining the inequalities, we finally obtain: $p(\underline{\mathcal{K}}) \geq p(\underline{S}^*)$.

□

2.3 Upper bounds relations

In this section we discuss the theoretical relations between the presented upper bounds: the LP-relaxation of models (1a)-(1d) and (2a)-(2d), the weighted clique cover bound and the proposed capacitated weighted clique cover bound.

A straightforward way of deriving an upper bound for the KPCG is to consider the LP-relaxation of model (1a)-(1d) and neglect the conflict constraints (1c). The resulting relaxed problem corresponds to the LP-relaxation of a 0-1 KP. This upper bound is denoted as fractional KP ($_{frac}KP$) in the following.

We say that bound A dominates bound B if the value of A is never larger than the value of B, thus providing a better information on the optimal solution value of the KPCG.
Since clique inequalities (2c) are a strengthening of inequalities (1c), we have that:

Proposition 3. *The LP-relaxation of model (2a)-(2d) dominates the LP-relaxation of model (1a)-(1d) .*

Proposition 4. *If $\mathcal{C}$ is the set of all cliques of $\mathcal{G}$, then the LP-relaxation of model (2a)-(2d) dominates the bound obtained by optimally solving the minimum weight clique cover problem.*

Proof. The minimum weight clique cover problem is the dual of the LP-relaxation of model (2a)-(2d) without the capacity constraint (2b). □

Many graph classes contain exponentially many maximal cliques, thus it is impractical to use all the cliques. In our implementation $\mathcal{C}$ is a subset of the clique set and, similarly, the clique cover problem is heuristically solved by considering a subset of the clique set. For these reasons, the dominance property of Proposition 4 does not apply to our implementation. Nevertheless, as discussed in Section 3, the LP-relaxation of model (2a)-(2d) gives tighter bounds.

Proposition 5. *The bound obtained by optimally solving the minimum weight capacitated clique cover problem dominates the bound obtained by optimally solving the minimum weight clique cover problem.*

Proof. Any clique cover can be transformed into a partial clique cover that satisfies (3) by removing the cliques with worst load-over-weight ratio. The resulting partial clique cover, by construction, satisfies (4). □

Proposition 6. *The bound obtained by optimally solving the minimum weight capacitated clique cover problem dominates ${}_{frac}KP$.*

As previously mentioned, we do not solve to optimality the minimum weight clique cover and minimum weight capacitated clique cover problems, instead, these problems are tackled by means of heuristic algorithms, producing the upper bounds CC and ${}_{cap}CC$, respectively. Concerning the relations between CC, ${}_{cap}CC$ and ${}_{frac}KP$, by applying similar arguments to those above, we have:

Proposition 7. *If CC and ${}_{cap}CC$ consider the same cliques in the same order, then ${}_{cap}CC$ dominates CC.*

and

Proposition 8. *${}_{cap}CC$ dominates ${}_{frac}KP$.*

3 Computational experiments

The procedures described in the previous sections were coded in C, and the resulting algorithms were tested on a workstation equipped with an Intel Xeon E3-1220 3.1 GHz CPU and 16 GB of RAM, running a Linux operating system.

3.1 Benchmark instances

When considering exact algorithms, two different groups of instances of the KPCG were proposed in the literature.

A first group of instances considers conflict graphs with densities ranging from 0.1 to 0.9. These instances appear in papers that solve to optimality the Bin Packing Problem with Conflict Graph through a Branch-and-Price algorithm [6, 4, 19]. In this case, the KPCG arises as subproblem in column

generation, and the profits associated with the items vary from iteration to iteration. All the last three mentioned papers derive their instances from the set of Bin Packing problems proposed by Falkenauer [5]. They consist of 8 classes each composed by 10 instances; in the first 4 classes the items have a weight with uniform distribution in $[20, 100]$ and the knapsack capacity is $c = 150$. The number n of items is 120, 250, 500 and 1000, respectively. The last 4 classes have weights with uniform distribution in $[250, 500]$ and $c = 1000$. Items are generated by triplets and they consist of $60, 120, 349, 501$ items. Each triplet in this class is generated so as to form an exact packing in the knapsack. Following [19], we have generated random conflict graphs for each instance with density values in the range from 0.1 to 0.9. We have also added a profit associated with each item (profits are not defined for the Bin Packing Problem with Conflict Graph). We have considered two settings: random profits uniformly distributed in the range $[1, 100]$, and correlated profits: $p_i = w_i + 10$ $(i = 1, \ldots, n)$. Finally, since in the original setting only a small number of items can be accommodated in the knapsack, we have considered the same instances with larger capacity of the knapsack (i.e., the original capacity is multiplied by an integer coefficient). Each dataset[1] is denoted by a capital letter, identifying the setting used for the profit ("R" for random profits and "C" for correlated profits) and a number denoting the multiplying coefficient for the knapsack capacity. In addition, 20 instances introduced in [11], for which only heuristic results are reported in the literature, have similar densities. However, these instances are not available anymore.

A second group of instances considers very sparse conflict graphs, with densities ranging from 0.001 to 0.02, as the ones considered in [20, 11, 12]. Instances from these papers are not available anymore, so we generated low density instances as described in [20, 11, 12].

We focus our experiments on the first set of instances, as it contains a wide range of graph densities. Tackling instances with very sparse conflict graphs requires alternative solution strategies, as discussed at the end of this section. As reference in our computational experiments, we take the algorithm by Sadykov and Vanderbeck [19], which is tested on instances with the same graph densities we focus on. In [19], the $_{frac}KP$ upper bound is considered. The bound is obtained by solving the LP-relaxation of model (1a)-(1d), in which conflict constraints (1c) are neglected. It corresponds to the LP-relaxation of a 0-1 KP and can be solved in $O(n)$ time (recall that items are ordered by non-increasing profit over weight ratio), by using a greedy algorithm (see Kellerer et al. [15]). The branching rule used in [19] is described at the beginning of Section 2.1 (before the improvement we introduce).

In [19] the KPCG appears as a subproblem in the column generation process, hence, item profits are defined by the dual variables values and are not directly available. We therefore generated item profits, and compare our Branch-and-Bound algorithm with our implementation of the Branch-and-Bound algorithm by [19]. In addition, we compare our Branch-and-Bound algorithm solutions with the solutions obtained by models (1a)-(1d) and (2a)-(2d) solved by CPLEX 12.6. At the end of Section 3.3, we briefly report on results obtained when considering the second set of very sparse instances.

[1]Instances are available at `www.or.deis.unibo.it`

3.2 Computational performance of the Branch-and-Bound algorithm components.

The first component of the Branch-and-Bound algorithm we evaluate is the upper bounding procedure. In Section 2.3, we investigated the theoretical relations among upper bounds; in this section we report on testing the upper bounds on the considered set of benchmark instances. In particular, we consider the LP-relaxation of the 0-1 KP upper bound (fractional knapsack, ${}_{frac}KP$), used in [19], the weighted clique cover bound (CC) used in [9] for the MWSSP, and the proposed capacitated weighted clique cover bound (${}_{cap}CC$). The aim of this comparison is to show that it is crucial to take into account both the incompatibility constraints and the capacity constraint. The results are shown in Tables 1 and 2.

In Table 1 the instances are grouped by dataset, and in Table 2 the instances are grouped by density, reported in the first column of the corresponding table, respectively. Then, for each considered upper bound UB, the tables report the percentage optimality gap computed as: $100\frac{UB-z^*}{z^*}$, where z^* is the optimal (or the best known) solution value.

As expected, the quality of the bound based on the linear relaxation of the 0-1 KP constraints only (column ${}_{frac}KP$) decreases with the increase of the density of the incompatibility graph. The CC bound, that takes into account the incompatibility constraints and drops the knapsack constraints, is very weak even on dense instances. The ${}_{cap}CC$ bound, which combines the information on incompatibilities and capacity, outperforms the other combinatorial bounds. As mentioned in Section 2.2, it dominates ${}_{frac}KP$ and it dominates CC as well if the clique cover is the same. Since we use a greedy heuristic to compute the clique cover, this theoretical property cannot be applied to our implementation. Nevertheless, in all the considered instances, ${}_{cap}CC$ gives a stronger upper bound than CC.

dataset	${}_{frac}KP$	CC	${}_{cap}CC$
R1	29.69	1608.59	18.50
R3	97.59	764.15	41.07
R10	378.01	641.81	104.01
C1	4.25	2446.71	2.89
C3	20.41	823.44	14.43
C10	176.93	416.07	92.54

Table 1: Upper bounds percentage optimality gaps. Instances aggregated by dataset.

Next we evaluate the benefit obtained by each of the Branch-and-Bound components (the newly proposed upper bounding procedure ${}_{cap}CC$ and branching rule ${}_{pre}KP$) *separately*. To this aim, we embed each of the two components in the Branch-and-Bound algorithm by Sadykov and Vanderbeck [19], and compare the obtained results with the original version of the algorithm. This comparison is performed to show that both the proposed upper bounding procedure and branching rule, even used separately, are more effective than the existing ones.

density	$_{frac}KP$	CC	$_{cap}CC$
0.1	11.96	1572.31	6.64
0.2	26.66	1322.58	15.37
0.3	46.61	1192.75	26.64
0.4	69.07	1119.96	38.89
0.5	95.90	1060.65	50.36
0.6	124.97	1015.25	59.43
0.7	164.72	974.07	69.05
0.8	217.79	923.03	74.84
0.9	302.65	870.56	68.93

Table 2: Upper bounds percentage optimality gaps. Instances aggregated by density.

We first embed, in our implementation of the algorithm in [19], the $_{cap}CC$ bound and keep the original branching rule, then we embed our improved branching rule $_{pre}KP$ and keep the $_{frac}KP$ upper bound.

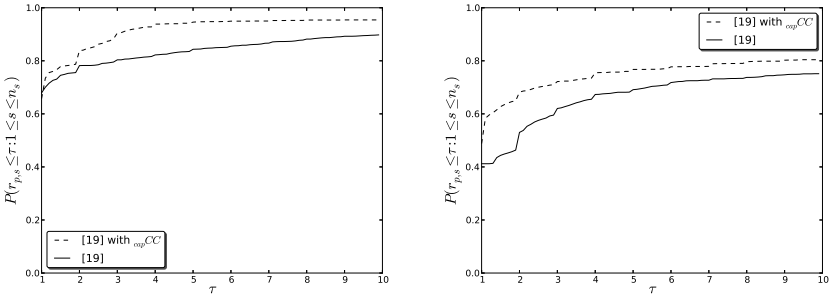
The time limit for each instance is set to 1800 seconds. The results are shown in Figures 5 and 6. We report, on the left of each figure, the results on the random instances, while, on the right, we report the results on the correlated instances. In particular, for each group of instances, we report the performance profiles, drawn according to the methodology introduced in [3], of the modified algorithms (indicated by [19] *with* $_{cap}CC$, and [19] *with* $_{pre}KP$) and of the original algorithm (indicated by [19]). The performance profile of an algorithm is the cumulative distribution function for a performance metric: in Figures 5 and 6, the metric is the computing time for solving an instance of the KPCG. For each algorithm and for each instance, we consider the time τ needed by the algorithm to solve the instance, normalized with respect to the fastest algorithm. For each value τ on the horizontal axis, a performance profile reports, on the vertical axis, the fraction of the dataset for which the algorithm is at most τ times slower than the fastest algorithm.

When the reference algorithm [19] is enhanced with the $_{cap}CC$ bound, the results are improved for both random and correlated instances, as can be seen in Figure 5. The same happens when the reference algorithm [19] embeds the improved branching rule $_{pre}KP$, as can be seen in Figure 6.

3.3 Comparison of the Branch-and-Bound algorithm with state-of-the-art methods

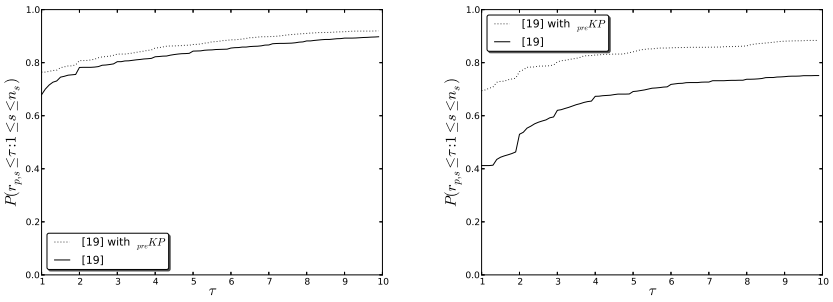
In this section we compare the performance of the new Branch-and-Bound algorithm, which is obtained by combining the proposed bounding procedure $_{cap}CC$ and the improved branching rule $_{pre}KP$, with the algorithm in [19] and with models (1a)-(1d) and (2a)-(2d) solved by the MIP solver of CPLEX 12.6. The time limit is set to 1800 seconds. Figure 7 reports the performance profiles for random and correlated instances, respectively. In Figure 7, the proposed Branch-and-Bound algorithm is denoted by *BCM* (Bettinelli, Cacchiani, Malaguti).

The first information that clearly appears from performance profiles is that



(a) Random instances (b) Correlated instances

Figure 5: Performance profiles for the KPCG algorithm in [19] and the algorithm [19] *with* $_{cap}CC$, obtained by using the $_{cap}CC$ bound. Each curve represents the probability to have a computing time ratio smaller or equal to τ with respect to the best performing algorithm. On the left we show random instances and on the right correlated ones.

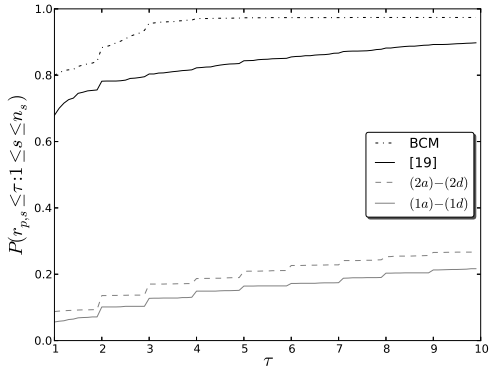


(a) Random instances (b) Correlated instances

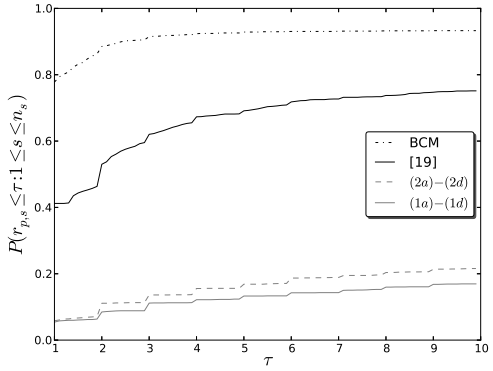
Figure 6: Performance profiles for the KPCG algorithm in [19] and the algorithm [19] *with* $_{pre}KP$, obtained by using the improved branching procedure $_{pre}KP$. Each curve represents the probability to have a computing time ratio smaller or equal to τ with respect to the best performing algorithm. On the left we show random instances and on the right correlated ones.

the KPCG is difficult for the CPLEX MIP solver, and that it is worth to design a specialized algorithm, although exploiting the stronger clique model (2a)-(2d) has some advantages on the weaker model (1a)-(1d).

Both Branch-and-Bound algorithms largely outperform the CPLEX MIP solver; among the two, the newly proposed algorithm improves on the results of the reference one [19] on both the computing times and the percentage of solved instances. For example, if we consider random instances, the new algorithm is the fastest for 80% of the instances, and can solve 97.1% of the instances within a computing time not exceeding 4 times the computing time of the fastest method ($\tau = 4$). The algorithm in [19] is the fastest for 67.9% of the instances, and can solve 82.2% of the instances within a computing time not exceeding 4 times the computing time of the fastest method ($\tau = 4$). If we consider correlated



(a) Random instances



(b) Correlated instances

Figure 7: Performance profiles for the KPCG algorithm in [19], the new Branch-and-Bound algorithm BCM, and the CPLEX MIP solver applied to formulations (2a)-(2d) and (1a)-(1d). Each curve represents the probability to have a computing time ratio smaller or equal to τ with respect to the best performing algorithm.

instances, the new algorithm is the fastest for 77.8% of the instances, and can solve 92.4% of the instances within 4 times the computing time of the fastest method ($\tau = 4$). The algorithm in [19] is the fastest for 41.2% of the instances, and can solve 67.3% of the instances within 4 times the computing time of the fastest method ($\tau = 4$). In addition, we can see that, on the random instances, the proposed Branch-and-Bound algorithm solves 97.3% when $\tau = 5$, while the algorithm in [19] reaches at most 89.8% when $\tau = 10$. On the correlated instances, the difference between the two algorithms is more evident: BCM solves 92.9% of the instances when $\tau = 5$ while the algorithm in [19] reaches the 75.1% when $\tau = 10$.

In Tables 3 and 4 we report detailed results for the random and correlated instances, respectively, grouped by classes; in Tables 5 and 6 we report detailed results for random and correlated instances, grouped by density. We omit the re-

sults obtained by solving model (1a)-(1d), as they are worse than those obtained with model (2a)-(2d). In all tables, the average computing time and number of nodes are computed with respect to the number of instances optimally solved by the corresponding method.

Concerning the difficulty of the instances, datasets with tight knapsack constraints (C1 and R1) are easy for both Branch-and-Bound algorithms and require a smaller number of visited nodes, indeed, all algorithms can solve all instances. The number of nodes that are explored by BCM, and the corresponding computing time, are, on average, significantly reduced with respect to the Branch-and-Bound in [19], as it can be observed in all the tables, when both algorithms solve all instances. In addition, as also shown in the performance profiles, BCM solves a larger number instances than both other methods.

Among the 8 classes of instances, class 4 (instances with 1000 items and uniform weights) is the most difficult one, as testified by the number of nodes, the computing times and the number of unsolved instances. In particular for datasets R10 and C10, which are the most difficult ones, 40 and 50 instances remain unsolved, respectively, when considering the BCM algorithm, i.e., the best performing one.

By considering the density of the conflict graphs, we see that instances with very high densities are easier for the compared Branch-and-Bound algorithms: both algorithms can solve all the instances with densities larger or equal to 0.7, while none can solve all instances with densities between 0.1 and 0.5 in datasets C10 and R10 i.e., when the knapsack capacity is large.

Low density instances. For the second set of instances, with graph densities ranging between 0.007 and 0.016, the proposed algorithm is not particularly effective: indeed, it relies on explicitly exploiting incompatibilities between items, and, when the conflict graph is very sparse, other methods, such as [20, 12], are more efficient.

One may wonder whether methods designed for low density instances could be effective on graphs with higher densities. To partially answer this question, we consider the exact method proposed in [12], as it has better performance than that of [20]. In particular, we focus on the reduction procedure, that is a crucial step of the algorithm proposed in [12]: it consists of fixing some decision variables to their optimal values (see [12] for further details). The reduction procedure is initialized, in [12], with a heuristic solution value obtained by applying the reactive local search algorithm proposed in [11]. Instead, we directly provide the reduction procedure with the optimal solution value. It turned out that, on the first set of instances (with graph densities between 0.1 and 0.9), the reduction procedure is not effective. In particular, on R1 and on C1, even though the percentage of fixed variables is high (around 80% on average), the reduction procedures requires long computing times (two order of magnitude larger than the total time BCM needs to obtain the optimal solutions). On the contrary, already on R3 and C3, the reduction procedure is not capable of reducing the instance size: the percentage of fixed variables is on average around 15% and 5%, respectively. Thus, this kind of approach is not appropriate for larger density graphs.

	R1							
	[19]			BCM			(2a)-(2d)	
class	solved	time	nodes	solved	time	nodes	solved	time
1	90	0.000	45.2	90	0.000	26.4	90	0.056
2	90	0.000	73.2	90	0.000	41.2	90	0.242
3	90	0.001	166.6	90	0.001	82.7	90	1.153
4	90	0.001	333.4	90	0.009	160.9	90	4.662
5	90	0.000	87.8	90	0.000	16.6	90	0.011
6	90	0.000	390.1	90	0.000	32.2	90	0.043
7	90	0.003	2378.9	90	0.000	62.6	90	0.218
8	90	0.033	14982.2	90	0.002	96.8	90	1.030

	R3							
	[19]			BCM			(2a)-(2d)	
class	solved	time	nodes	solved	time	nodes	solved	time
1	90	0.000	1091.5	90	0.000	279.0	90	0.495
2	90	0.003	3941.1	90	0.009	866.9	90	3.696
3	90	0.045	21626.8	90	0.100	4053.8	90	61.066
4	90	0.524	125983.1	90	1.152	20481.3	65	461.742
5	90	0.000	418.8	90	0.000	107.5	90	0.097
6	90	0.000	1465.7	90	0.000	304.2	90	0.549
7	90	0.006	5807.7	90	0.012	1049.5	90	4.379
8	90	0.074	43100.0	90	0.090	3916.6	90	78.189

	R10							
	[19]			BCM			(2a)-(2d)	
class	solved	time	nodes	solved	time	nodes	solved	time
1	90	0.470	845744.2	90	0.028	7336.2	90	1.101
2	90	30.755	29171776.1	90	3.587	311343.0	90	96.027
3	69	207.035	107696773.2	84	199.585	8202490.5	40	536.021
4	40	106.035	22116493.5	50	195.765	4895498.4	8	979.029
5	90	0.008	31518.4	90	0.000	389.9	90	0.110
6	90	1.261	2314126.5	90	0.047	12237.5	90	1.166
7	88	105.037	97723100.2	90	6.766	586538.9	90	144.993
8	70	178.822	79047091.8	80	159.751	6649506.5	40	508.469

Table 3: Exact methods on the random datasets. Results aggregated by classes.

	C1							
	[19]			BCM			(2a)-(2d)	
class	solved	time	nodes	solved	time	nodes	solved	time
1	90	0.000	495.7	90	0.000	71.4	90	0.168
2	90	0.012	8345.8	90	0.001	107.2	90	0.946
3	90	0.022	8510.3	90	0.003	177.7	90	5.630
4	90	11.828	1913753.8	90	0.010	451.6	90	40.030
5	90	0.000	4359.6	90	0.000	164.5	90	0.021
6	90	0.016	25241.2	90	0.000	230.1	90	0.067
7	90	0.098	65014.7	90	0.001	203.6	90	0.339
8	90	0.735	255015.1	90	0.002	338.2	90	2.843

	C3							
	[19]			BCM			(2a)-(2d)	
class	solved	time	nodes	solved	time	nodes	solved	time
1	90	0.005	12930.4	90	0.003	1948.4	90	1.563
2	90	0.089	79040.6	90	0.061	11126.6	90	32.572
3	90	2.008	1029233.1	90	0.955	80368.4	72	413.528
4	88	64.828	17624047.3	90	19.443	761690.3	21	209.631
5	90	0.031	109383.5	90	0.000	827.9	90	0.135
6	88	69.174	120484341.8	90	0.005	2831.3	90	1.470
7	70	8.198	7365262.0	90	0.064	14061.9	90	35.566
8	60	34.744	14750601.3	90	0.686	82463.3	79	342.026

	C10							
	[19]			BCM			(2a)-(2d)	
class	solved	time	nodes	solved	time	nodes	solved	time
1	90	24.508	46535034.1	90	0.847	390718.0	90	2.349
2	61	21.300	18558165.0	86	149.856	27200583.0	75	212.905
3	50	182.781	68685161.7	54	149.385	18954904.4	24	712.534
4	30	53.275	12702323.9	40	151.904	10968827.5	0	0
5	90	0.811	2954067.7	90	0.008	9454.4	90	0.189
6	88	76.573	149901735.6	90	2.765	1266635.1	81	2.718
7	60	17.971	13926764.8	77	114.528	21362729.7	63	250.630
8	50	184.604	68934062.6	50	22.482	2843795.3	20	800.996

Table 4: Exact methods on the correlated datasets. Results aggregated by classes.

	R1							
	[19]			BCM			(2a)-(2d)	
density	solved	time	nodes	solved	time	nodes	solved	time
0.1	80	0.012	6811.3	80	0.001	31.6	80	0.126
0.2	80	0.009	4482.8	80	0.000	35.1	80	0.250
0.3	80	0.006	2998.1	80	0.001	44.9	80	0.408
0.4	80	0.006	2588.0	80	0.001	60.2	80	0.541
0.5	80	0.005	1896.8	80	0.001	62.6	80	0.685
0.6	80	0.002	950.1	80	0.002	75.1	80	0.792
0.7	80	0.001	537.4	80	0.003	86.3	80	1.206
0.8	80	0.000	302.5	80	0.002	94.1	80	1.678
0.9	80	0.000	197.6	80	0.003	94.4	80	2.656

	R3							
	[19]			BCM			(2a)-(2d)	
density	solved	time	nodes	solved	time	nodes	solved	time
0.1	80	0.026	20333.5	80	0.004	274.1	80	0.124
0.2	80	0.028	13781.9	80	0.031	1241.2	80	0.543
0.3	80	0.067	21820.6	80	0.132	2923.2	80	5.630
0.4	80	0.137	35977.3	80	0.279	5312.3	80	44.325
0.5	80	0.180	44764.2	80	0.401	7877.4	77	89.022
0.6	80	0.158	42919.7	80	0.387	8415.2	73	97.661
0.7	80	0.090	29528.4	80	0.208	5553.8	71	80.626
0.8	80	0.037	14777.2	80	0.072	2425.7	74	118.057
0.9	80	0.011	4961.3	80	0.018	918.3	80	136.017

	R10							
	[19]			BCM			(2a)-(2d)	
density	solved	time	nodes	solved	time	nodes	solved	time
0.1	69	110.369	88023913.8	70	14.340	652371.4	78	105.306
0.2	48	212.425	200328771.6	54	100.734	4412874.2	50	144.009
0.3	50	26.802	21984852.8	70	315.999	13251314.6	50	139.575
0.4	70	260.592	101365617.8	70	23.951	1125284.5	50	65.728
0.5	70	20.220	7493461.6	80	114.559	2904611.6	50	38.771
0.6	80	50.684	10770967.7	80	9.434	267046.5	50	25.660
0.7	80	3.968	1011439.6	80	1.190	44200.2	70	402.629
0.8	80	0.358	111981.9	80	0.191	7977.5	70	167.614
0.9	80	0.031	12384.8	80	0.026	1509.0	70	38.564

Table 5: Exact methods on the random datasets. Results aggregated by density.

	C1							
	[19]			BCM			(2a)-(2d)	
density	solved	time	nodes	solved	time	nodes	solved	time
0.1	80	12.337	1972293.4	80	0.000	79.0	80	0.101
0.2	80	0.880	188977.5	80	0.000	90.6	80	0.248
0.3	80	0.272	90194.7	80	0.000	111.0	80	0.464
0.4	80	0.147	63056.6	80	0.001	202.6	80	0.607
0.5	80	0.302	104502.7	80	0.001	168.3	80	1.028
0.6	80	0.246	91745.9	80	0.004	270.1	80	4.088
0.7	80	0.083	37551.2	80	0.003	274.5	80	4.823
0.8	80	0.027	14043.1	80	0.004	357.4	80	17.450
0.9	80	0.006	3463.0	80	0.005	408.6	80	27.491

	C3							
	[19]			BCM			(2a)-(2d)	
density	solved	time	nodes	solved	time	nodes	solved	time
0.1	56	163.731	206171329.1	80	0.051	3667.9	79	8.640
0.2	60	3.792	1837069.9	80	0.567	26078.8	80	34.728
0.3	70	15.724	9309906.6	80	3.422	120223.8	72	55.221
0.4	80	32.466	12229559.4	80	5.568	190780.4	69	90.409
0.5	80	11.195	2494963.8	80	8.279	351816.9	66	198.429
0.6	80	5.598	1451069.2	80	4.300	252950.2	52	83.907
0.7	80	1.950	594962.3	80	1.320	95969.1	64	336.618
0.8	80	0.530	168275.2	80	0.305	26311.1	70	135.991
0.9	80	0.080	28463.3	80	0.058	6934.7	70	77.380

	C10							
	[19]			BCM			(2a)-(2d)	
density	solved	time	nodes	solved	time	nodes	solved	time
0.1	29	195.022	384555162.0	47	203.917	36828873.3	35	100.320
0.2	30	116.997	221125214.2	36	260.334	46619765.3	30	2.970
0.3	30	3.204	5552855.4	50	57.207	13484306.0	38	319.944
0.4	50	39.479	30808014.2	54	132.369	17104415.3	50	218.665
0.5	70	250.275	94082917.5	70	29.312	3776521.3	50	79.238
0.6	70	13.200	5351330.9	80	72.695	5359030.3	50	46.125
0.7	80	19.800	4900728.8	80	5.405	441719.0	50	26.422
0.8	80	1.163	343428.4	80	0.375	32385.0	70	281.333
0.9	80	0.078	28554.0	80	0.061	7139.2	70	162.185

Table 6: Exact methods on the correlated datasets. Results aggregated by density.

4 Conclusion

The Knapsack Problem with Conflict Graph is a relevant extension of the 0-1 Knapsack Problem in which incompatibilities between pairs of items are defined. It has received considerable attention in the literature as a stand-alone problem, as well as a subproblem arising in solving, e.g., the Bin Packing Problem with Conflicts. The problem is computationally challenging, and cannot be easily tackled through a Mixed-Integer formulation solved with a general purpose MIP solver.

In this paper a new Branch-and-Bound algorithm, based on improved upper bounding and branching procedures, is proposed. Its performance is tested on a set of instances derived from classical Bin Packing Problems, having graph densities between 0.1 and 0.9. The obtained results show that the Branch-and-Bound algorithm outperforms, in our implementation, a recent state-of-the-art algorithm from the literature, as well as a Mixed Integer Programming formulation tackled through a general purpose solver.

References

[1] H. Akeb, M. Hifi, and M. E. Ould Ahmed Mounir. Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60(4):811–820, 2011.

[2] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375–382, 1990.

[3] E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[4] S. Elhedhli, L. Li, M. Gzara, and J. Naoum-Sawaya. A branch-and-price algorithm for the bin packing problem with conflicts. *INFORMS Journal on Computing*, 23(3):404–415, 2011.

[5] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[6] A.E. Fernandes-Muritiba, M. Iori, E. Malaguti, and P. Toth. Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing*, 22(3):401–415, 2010.

[7] M. Fischetti and A. Lodi. Local branching. *Mathematical programming*, 98 (1-3):23–47, 2003.

[8] M. Gendreau, G. Laporte, and F. Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, 31:347–358, 2004.

[9] S. Held, W. Cook, and E.C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4:363–381, 2012.

[10] M. Hifi. An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. *Engineering Optimization*, 46(8): 1109–1122, 2014.

[11] M. Hifi and M. Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6):718–726, 2006.

[12] M. Hifi and M. Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & operations research*, 34(9):2657–2673, 2007.

[13] M. Hifi and N. Otmani. An algorithm for the disjunctively constrained knapsack problem. *International Journal of Operational Research*, 13(1): 22–43, 2012.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[15] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[16] E. Malaguti, M. Monaci, and P. Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8:174–190, 2011.

[17] S. Martello and P. Toth. *Knapsack problems*. Wiley New York, 1990.

[18] U. Pferschy and J. Schauer. The knapsack problem with conflict graphs. *J. Graph Algorithms Appl.*, 13(2):233–249, 2009.

[19] R. Sadykov and F. Vanderbeck. Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255, 2013.

[20] T. Yamada, S. Kataoka, and K. Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43(9), 2002.