Bin Packing with Conflicts: a Generic Branch-and-Price Algorithm

Ruslan Sadykov
RealOpt team, INRIA Bordeaux — Sud-Ouest, 351 cours de la Libération, 33405 Talence France
Ruslan.Sadykov@inria.fr

François Vanderbeck
Université Bordeaux I and RealOpt team, INRIA Bordeaux — Sud-Ouest, 351 cours de la Libération, 33405 Talence France fv@math.u-bordeaux1.fr

The bin packing problem with conflicts consists in packing items in a minimum number of bins of limited capacity while avoiding joint assignments of items that are in conflict. Our study demonstrates that a generic implementation of a Branch-and-Price algorithm using specific pricing oracle yields comparatively good performance for this problem. We use our black-box Branch-and-Price solver BaPCod, relying on its generic branching scheme and primal heuristics. We developed a dynamic programming algorithm for pricing when the conflict graph is an interval graph, and a depth-first-search branch-and-bound approach for pricing when the conflict graph has no special structure. The exact method is tested on instances from the literature where the conflict graph is an interval graph, as well as harder instances that we generated with an arbitrarily conflict graph and larger number of items per bin. Our computational experiment report sets new benchmark results for this problem, closing all open instances of the literature in one hour of CPU time.

Key words: branch-and-price; bin packing; knapsack; conflict graphs; interval graphs.

---

1. Introduction

In the Bin Packing Problem (BPP), items of different sizes/weights must be packed into a minimum number of identical bins of given capacity. In the variant with conflicts (denoted BPPC), a graph is given where nodes represent items and edges represent conflicts between pairs of items: any two items that are linked by an edge cannot be assigned to the same bin. Thus, the problem is a combination of the Bin Packing Problem and the Vertex Coloring Problem. It arises in many real-world applications, such as examination scheduling (Laporte and Desroches, 1984), parallel computing and database storage (Jansen, 1999), product delivery (Christofides et al., 1979), resource clustering in highly distributed parallel computing

(Beaumont et al., 2008). The special case of an interval conflict graph is a realistic model on its own. It arises for instance in the mutual exclusion scheduling problem (Baker and Coffman, 1996; Gardi, 2009) in which all items or tasks are represented by time intervals associated to their schedule, and a conflict between tasks arises when the associated intervals overlap. A related application arise in workforce planning (Gardi, 2005): the problem is to assign tasks to a minimum number of workers; each task is defined by a start time and a completion time; tasks assigned to the same worker may not overlap in time; the total duration of the tasks assigned to a worker may not exceed a given bound.

The BPPC was considered by Jansen and Öhring (1997) and Jansen (1999) who developed approximation algorithms for special cases of conflict graphs. Several computational studies on the problem have recently been published. Gendreau et al. (2004) have evaluated six heuristics and lower bounding strategies for the problem. Different heuristics, lower bounds and an exact algorithm based on a branch-and-price approach were proposed by Fernandes Muritiba et al. (2010). A special purpose branch-and-price algorithm was developed by Elhedhli et al. (2011).

Here, we show that the BPPC can be efficiently solved by a generic branch-and-price algorithm: we use the generic software platform, BaPCod, that is developed in our team. It includes the generic branching scheme of Vanderbeck (2010) and the generic column generation based primal heuristics of Joncour et al. (2010). The pricing subproblem is a Knapsack Problem with Conflicts (KPC). We developed a specific branch-and-bound oracle for a general conflict graph and a dynamic programming solver for the special case of an interval graph. To the best of our knowledge, the latter dynamic program is an original contribution although a dynamic program for the more general case of the KPC in a chordal graph can be found in Pferschy and Schauer (2009). Our computational results demonstrates that our approach improves on the state-of-art branch-and-price algorithms proposed by Fernandes Muritiba et al. (2010) and by Elhedhli et al. (2011).

The paper is organized as follows. In Section 2, we provide a compact formulation and an extended (set covering) integer programming formulation of the problem. Our algorithm is presented in Section 3. Results of computational experiments are reported in Section 4. In Section 5, we draw conclusions.

2. Formulations of the problem

Formally, the BPPC can be described as follows. We are given K identical bins of capacity W, a set $V = \{1, 2, \ldots, n\}$ of items characterized by a non-negative capacity consumption $w_i \leq W$, and a conflict graph $G = (V, E)$, where E is a set of edges such that $(i, j) \in E$ when i and j are in conflict. The problem is to assign items to bins, using a minimum number of bins, while ensuring that the total weight of the items assigned to a bin does not exceed the bin capacity W, and that no two items that are in conflict are assigned to the same bin. The number K is assumed to be large enough to guarantee feasibility; more precisely it is a valid upper bound on the number of bins in an optimal solution (note that $K \leq n$).

A natural and compact integer programming formulation makes use of binary variables x_{ik} taking value 1 if item i is assigned to bin k and 0 otherwise, and binary variables y_k taking value 1 if bin k is used and 0 otherwise:

$$\min \sum_{k=1}^{K} y_k \tag{1a}$$

$$s.t. \quad \sum_{k=1}^{K} x_{ik} \geq 1, \qquad i = 1, \ldots, n, \tag{1b}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \qquad k = 1, \ldots, K, \tag{1c}$$

$$x_{ik} + x_{jk} \leq y_k, \qquad (i, j) \in E, \quad k = 1, \ldots, K, \tag{1d}$$

$$y_k \in \{0, 1\}, \qquad k = 1, \ldots, K, \tag{1e}$$

$$x_{ik} \in \{0, 1\}, \qquad i = 1, \ldots, n, \quad k = 1, \ldots, K. \tag{1f}$$

Constraints (1b) require that each item is assigned to a bin; constraints (1c) enforce the bin capacity; and constraints (1d) formulate the conflicts. The objective (1a) is to minimize the number of used bins.

The linear programming relaxation of formulation (1) is weak, even if valid inequalities are added (Martello and Toth, 1990). Instead, as for a standard bin packing problem, one can use a set covering reformulation of (1) (Fernandes Muritiba et al., 2010; Elhedhli et al., 2011). Such reformulation results from applying the Dantzig-Wolfe decomposition principle to (1) (Vanderbeck and Savelsbergh, 2006): once (1b) are dualized in a Lagrangian way, subsystem (1c-1f) decomposes into a subproblem for each bin k. Let $\mathcal{B}$ be the family of all the subsets of items which are not in conflict and fit into one bin, i.e., the solutions to a

subproblem. Each subset $B \in \mathcal{B}$ is defined by an indicator vector x^B ($x_i^B = 1$ if item i is in set B) and associated with a binary variable λ_B taking value 1 if the corresponding subset of items is selected to fill one bin. The reformulation is:

$$\min \sum_{B \in \mathcal{B}} \lambda_B \tag{2a}$$

$$s.t. \quad \sum_{B \in \mathcal{B}} x_i^B \lambda_B \geq 1, \qquad i = 1, \ldots, n, \tag{2b}$$

$$\lambda_B \in \{0, 1\}, \qquad B \in \mathcal{B}. \tag{2c}$$

Here, constraints (2b) replace constraints (1b), all other constraints of the compact formulation (1) are build in the definition of feasible sets: $B \in \mathcal{B}$.

Formulation (2) is tackled using a branch-and-price approach: at each node of a branch-and-bound tree, the linear relaxation of (2) is solved by column generation to provide a lower bound. The calculation of this lower bound is done by iteratively solving:

- the *restricted master problem* (RMP) which is the linear relaxation of (2) with a restricted number of variables;
- and the *pricing problem* which determines whether there exists a variable λ_B to be added to (RMP) in order to improve its current solution; this amounts to searching for the set $B \in \mathcal{B}$, solution to subsystem (1c-1f), which yields the minimum reduced cost column for (RMP).

Let $\{\pi_i\}_{i \in V}$ be a current dual solution of the (RMP). Then, the pricing problem can be formulated as

$$\max \sum_{i=1}^{n} \pi_i z_i \tag{3a}$$

$$s.t. \quad \sum_{i=1}^{n} w_i z_i \leq W \qquad i = 1, \ldots, n, \tag{3b}$$

$$z_i + z_j \leq 1, \qquad (i, j) \in E, \tag{3c}$$

$$z_i \in \{0, 1\}, \qquad i = 1, \ldots, n. \tag{3d}$$

Model (3) is a Knapsack Problem with Conflicts (KPC), as studied, for example, by Hifi and Michrafy (2007).

3. A branch-and-price algorithm

Our algorithm for solving the BPPC is a branch-and-price procedure. It is to be distinguished from the branch-and-price algorithms used by Fernandes Muritiba et al. (2010) and Elhedhli et al. (2011) for the following features:

1. the way we solve the pricing problem (denoted KPC);
2. the branching rule that we use;
3. the use of a generic column generation based heuristic.

Before going into details, we briefly present the "features" of the branch-and-price algorithms described in the literature.

Fernandes Muritiba et al. (2010) use combinatorial lower bounds and a tabu search based heuristic as a "preprocessing" step. (The instances of their numerical test bed have interval conflict graphs with typically large sub-cliques which make their combinatorial lower bound relatively tight.) In their branch-and-price, the pricing problem is solved by a greedy heuristic. If the latter fails to produce a column with a negative reduced cost, the MIP solver *CPLEX 10* is called for exact pricing. Branching is performed by implementing a disjunctive constraint on the λ variable with the largest fractional part, with priority to the round-up branch. A depth first strategy is used.

Elhedhli et al. (2011) developed a standard column generation approach where the pricing problem is solved by the MIP solver *CPLEX* after adding maximal clique inequalities generated using the Qualex library (Busygin, 2006). (The instances of their numerical test bed have interval conflict graphs with typically large sub-cliques which make such inequalities attractive.) Their branch-and-price algorithm rely on the branching rule of Ryan and Foster (1981) in which two items are selected and put to the same bin at the first child node and constrained to be in different bins at the other child node. However, their implementation of such branching is specific: instead a classic binary search tree, several pairs of items are considered simultaneously. In the first branch, they are all considered for joint assignment; while the other child nodes enumerate the disjoint assignments for each pair. Primal solutions are obtained using a rounding heuristic for the set covering problem (2).

3.1. Solving the Knapsack Problem with Conflicts

In selecting an algorithm to solve the KPC with interval and arbitrary conflict graphs, the first obvious choice is to apply an IP solver to formulation (3). However, our preliminary tests showed that very efficient IP solvers such as CPLEX are not fast enough to be called many times during the column generation procedure. An alternative specialized branch-and-cut algorithm for the KPC was proposed by Hifi and Michrafy (2007). It is faster than CPLEX only on a small fraction of instances with conflict graph density of around 1%. Therefore, we developed our own specialized algorithms for the KPC.

First consider the Knapsack Problem with Interval Conflict Graphs (KPICG). Formally, a graph $G = (V, E)$ is an interval graph if, to each vertex $v \in V$, one can associate an open interval I_v of the real line, i.e., $I_v = (a_v, b_v)$ for $a_v, b_v \in \mathbb{R}$ with $a_v < b_v$, such as two distinct vertices $u, v \in V$ are adjacent, i.e., form an edge, if and only if $I_u \cap I_v \neq \emptyset$. The family $\{I_v\}_{v\in V}$ is an interval representation of G. See Figure 1 for an illustration.

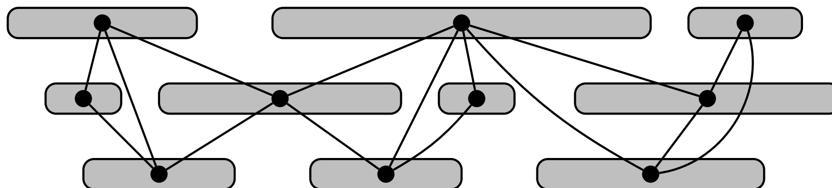

Figure 1: An interval graph and its interval representation

Recently, Pferschy and Schauer (2009) proposed a pseudo-polynomial algorithm for the KPC with chordal conflict graphs which is a super-class of interval graphs. The complexity of their algorithm is $O(nW^2)$. Its pseudo-polynomial complexity of order two makes this procedure too time consuming to be used for pricing within a branch-and-price algorithm. We designed a dynamic programming algorithm with a lower complexity for the special case of interval graph.

Our algorithm exploits the interval representation of the conflict graph: $\{I_i\}_{i\in V}$. Note that the procedure of (Corneil et al., 1998) allows to detect whether or not a graph $G = (V, E)$ is an interval graph and if it is, the algorithm produces an interval representation of the graph; its complexity is $O(|V| + |E|)$.

Definition 1. Considering an interval conflict graph, $G = (V, E)$, assume the items, $i \in V$, are indexed in non-decreasing order of the right endpoints of the corresponding intervals

$\{I_i = (a_i, b_i)\}_{i \in V}$ (the ties are resolved arbitrarily), i.e., $b_i \leq b_j$ if $i < j$. Let Q_i denote the set of items with indexes smaller than i that are not in conflict with i:

$$Q_i = \{j : \ j < i, (i, j) \notin E\}, \quad \forall i \in V.$$

Let $prev_i$ be the item in Q_i with the largest index (or 0 if $Q_i = \emptyset$):

$$prev_i = \begin{cases} \max\{j : \ j \in Q_i\}, & Q_i \neq \emptyset, \\ 0, & Q_i = \emptyset, \end{cases} \qquad \forall i \in V.$$

Observation 1. Consider an interval conflict graph, $G = (V, E)$. Given the item indexing of Definition 1, for every pair $i, j \in V$, such that $1 \leq j \leq prev_i$, i and j are not in conflict, i.e., $(i, j) \notin E$, while when $prev_i < j < i$, i and j are in conflict, i.e., $(i, j) \in E$.

Let $P(i, w)$ be the value of an optimal solution of the KPICG for the first i items and knapsack size w. With this notation, the solution of model (3) gives $P(n, W)$. By Observation 1, the solution set associated with $P(i, w)$ either includes item i and cannot include any items j such as $prev_i < j < i$; or it does not include item i and it reproduces the solution set for $P(i - 1, w)$. Therefore,

Observation 2.

$$P(i, w) = \max\{P(prev_i, w - w_i) + p_i, \ P(i - 1, w)\}. \tag{4}$$

The value $P(n, W)$ is the solution to KPICG. The associated solution set B can be retrieved by backtracking from value $P(n, W)$ to value $P(0, 0)$. The dynamic programming algorithm stemming from Observation 2 is formally presented as Function `DP`. It is easy to see that the time and the space complexity of the dynamic programming algorithm are both $O(nW)$ once the values *prev* are known. This complexity is more tractable in practice than that of the algorithm by Pferschy and Schauer (2009).

Next, we consider the knapsack problem with an arbitrary conflict graph. We developed a recursive enumeration procedure for the KPC that combines the classic depth-first-search based branch-and-bound algorithm for the 0-1 Knapsack Problem (Kelleler et al., 2004, section 2.4) with the enumeration algorithm for solving the maximum clique (or maximum independent set) problem by Carraghan and Pardalos (1990). The latter also makes use of a depth-first-search strategy, while dual bounds are obtained by simply ignoring all conflicts between free vertices, i.e. vertices which have not yet been fixed via branching decisions.

Function DP($n, p_{[1,\ldots,n]}, w_{[1,\ldots,n]}, W, prev_{[1,\ldots,n]}$)

```
for w ← 0 to W do
    P(0, w) ← 0;
for i ← 1 to n do
    for w ← 0 to w_i − 1 do
        P(i, w) ← P(i − 1, w);
        l_iw ← 0;
    for w ← w_i to W do
        P(i, w) ← P(i − 1, w);
        l_iw ← 0;
        if P(i, w) < P(prev_i, w − w_i) + p_i then
            P(i, w) ← P(prev_i, w − w_i) + p_i;
            l_iw ← 1;
w ← W;
B ← ∅;
i ← n;
while i > 0 do
    if l_iw = 1 then
        B ← B ∪ {i};
        w ← w − w_i;
        i ← prev_i;
    else
        i ← i − 1;
return B;
```

Definition 2. For each item $i \in V$, we define the list C_i of items in conflict with i. At any node of the enumeration tree, we denote by S^1 the set of items that have been selected in the current partial knapsack solution and by S^0 the items that have been fixed to 0. The set $F = (V \setminus (\cup_{i \in S^1} C_i \cup S^1 \cup S^0))$ denotes free items that are not fixed to either 0 or 1 in previous branching decisions and are not in conflict with items in S^1. Assume that items are indexed in the non-decreasing order of their "efficiency", i.e., of their ratio p_i/w_i. Then, $succ_i(F)$ denotes the item following i in the sorted sub-list of items of F. By extension $succ_0(F)$ denote the first element in F, while $last(F)$ the last element in F and $succ_{last(F)}(F) = n+1$.

During the depth-first-search, upper (dual) bounds UB are computed at each node of the tree by solving the continuous relaxation of the residual knapsack problem on set F, ignoring conflict constraints:

$$UB = \max \sum_{i \in F} p_i x_i + \sum_{i \in S^1} p_i \tag{5a}$$

$$s.t. \quad \sum_{i \in F} w_i x_i \leq W - \sum_{i \in S^1} w_i \qquad i \in F, \tag{5b}$$

$$0 \leq x_i \leq 1, \qquad i \in F. \tag{5c}$$

As the items in F are sorted according to their efficiencies, problem (5) can be solved in $O(n)$ time using a greedy algorithm (Kelleler et al., 2004).

If the current upper bound UB is smaller or equal to the value LB of the incumbent solution, we prune the node, putting an end to further recursive calls to the enumeration procedure. Otherwise, we Branch: for each item $i \in F$, we consider a child node where i is added to S^1 and all items of F that precede i in the ordering are added to S^0. As the items in F and in the conflict list C_i of the i-th item in F are sorted in the same order, each child node can be created in time $O(n)$. Thus, the time spent per node is linear. The recursive enumeration procedure for KPC is formally presented as Function `Node`(p, w, S^1, F, LB, B), where p is the current profit, w the current weight, S^1 the set of items fixed to 1, F the set of free items, LB the current lower bound, and B the associated current incumbent solution. Our depth-first-search branch-and-bound algorithm for KPC is invoked by calling `Node`$(0, 0, \emptyset, V, 0, \emptyset)$.

We also experimented with variants of the above algorithm. In a first variant, items were added to the partial solution in the non-increasing order of the ratio of profit by number of conflicts. In a second variant, dual bounds were computed by solving a continuous knapsack

```
Function Node(p, w, S^1, F, LB, B)
  if p > LB then
  |  LB ← p;
  |_ B ← S^1;
  UB ← p;
  c ← w;
  i ← succ_0(F);
  while c < W and i ≤ last(F) do
  |  if c + w_i ≤ W then
  |  |  UB ← UB + p_i;
  |  |  c ← c + w_i;
  |  |_ i ← succ_i(F);
  |  else
  |  |  UB ← UB + (W − c) · (p_i/w_i);
  |_ |_ c ← W;
  if UB ≤ LB then
  |_ return B;
  i ← succ_0(F);
  while p + (W − w) · (p_i/w_i) > LB and i ≤ last(F) do
  |  j ← succ_i(F);
  |  F ← F \ {i};
  |  if w + w_i ≤ W then
  |  |  Ŝ^1 ← S^1 ∪ {i};
  |  |  F̂ ← F \ C_i;
  |  |_ B ← Node(p + p_i, w + w_i, Ŝ^1, F̂, LB, B);
  |_ i ← j;
  return B;
```

problem that takes into account some of the conflicts only. Indeed, continuous knapsack problems with disjoint special ordered set (SOS) constraints can be solved using the $O(n^2)$ algorithm by Johnson and Padberg (1981). Therefore, we search for a conflict sub-graph that define a family of disjoint cliques, $C \subset V$, each of which defines a SOS constraint: $\sum_{i \in C} x_i \leq 1$. The family of cliques is constructed using the following iterative greedy procedure: items are indexed in non-decreasing order of their ratio p_i/w_i; at each iteration, one clique is formed and the corresponding vertices are not considered in subsequent iterations; to build a clique, we select the remaining item with smallest index (maximum ratio p_i/w_i) and add subsequent items that are in conflict with it as long as it forms a clique. The two above variants were not as successful experimentally as that of Function `Node`(p, w, S^1, F, LB, B).

Note that, in the column generation procedure, we do not have to solve the pricing

problem to optimality: the procedure can iterate as long as a solution with a negative reduced cost is found. Fernandes Muritiba et al. (2010) applied a heuristic to try to find such a solution before relying on an exact algorithm. This approach decreases the average time needed to solve the pricing problem. However, our computational study showed that the overall branch-and-price algorithm performance is not as good when using a non exact pricing problem solver because of the slower convergence of the column generation procedure. Indeed, the simplex algorithm is experimentally faster when pivoting on the most negative reduced cost variable.

3.2. The branching rule

The linear relaxation of (2) is traditionally called the master. It can be shown that the solution $\bar{\lambda}$ to the master is binary, i.e. $\bar{\lambda} \in \{0,1\}^{\mathcal{B}}$, if and only if for all item pairs $i, j \in V$, the number of selected subsets that contain both i and j is either zero or one. Equivalently $\bar{\lambda}_B \in \{0,1\}$, $\forall B \in \mathcal{B}$, if and only if solution $\bar{\lambda}$ can be projected on a corresponding solution $(\bar{x}, \bar{y})$ to the integer problem (1) – the projection is defined in Vanderbeck (2010). The result is known for the bin packing problem, or vertex coloring, but it applies more generally for a set partitioning like master problem with a pure binary subproblem (Vanderbeck, 2010; Vanderbeck and Wolsey, 2010).

Therefore, a natural branching scheme is the following. At a given node of the branch-and-price tree, given master solution $\bar{\lambda}$, one can identify two items $i, j \in V$ such that $0 < \sum_{B:i,j \in B} \bar{\lambda}_B < 1$, and branch by enforcing that these two items are either assigned to the same bin or must be in different bins. Such branching constraint can be enforced by removing inappropriate columns and by constraining the subproblem to avoid regenerating the inappropriate columns. In the first child node, where item i and j must be assigned to the same bin, one adds constraints $x_i = x_j$ to the pricing problem (3) and removes from the master all columns that do not satisfy it. In the second child, constraint $x_i + x_j \leq 1$ is added to the subproblem and columns with $x_i = x_j = 1$ are removed from the master. This scheme was originally proposed by Ryan and Foster (1981). It was used in the approach of Elhedhli et al. (2011)

Observe that adding branching constraints to (3) implies modifications to the pricing problem that are compatible with the KPC model: constraint $x_i = x_j$ amounts to contracting the associated vertices of the conflict graph into one vertex representing the item pair i, j; while $x_i + x_j \leq 1$ amounts to adding a conflict edge. However, the special structure of the

conflict graph might be lost. In particular, for applications where the initial conflict graph is an interval graph, additional conflicts introduced by branching will typically destroy this structure that made the pricing problem solvable in pseudo-polynomial time as we showed in Section 3.1. Thus, our dynamic programming algorithm on which the method rely for good performance cannot be used when the Ryan and Foster branching scheme is applied.

Instead, we use the generic branching scheme proposed by Vanderbeck (2010) that was specially designed to preserve the structure of the pricing problem. The scheme proceeds by progressively partitioning into column classes the set $\mathcal{B}$ of feasible pricing problem solutions and by implementing separate pricing on each class. A class is defined by restricting the solution set via fixing some variables to zero or one. Hence, pricing over a class can be done using the initial oracle since the latter can handle some variable fixing. The implementation developed in Vanderbeck (2010) guarantees that the number of created classes remains polynomial in the input size. Fractional master solutions are eliminated by adding branching constraint in the master that force an integer lower bound on the number of columns selected in each defined class. The dual bounds after branching are proved to be as strong as if branching constraints had been defined in the subproblem.

To be more specific let us examine how the generic branching scheme of Vanderbeck (2010) applies to the present problem. It takes a form closely related to the Ryan and Foster branching scheme. At the root node, a pair $i, j \in V$ is selected such that $0 < \sum_{B:i,j\in B} \bar{\lambda}_B < 1$. Then, in Node 1, branching is implemented by requiring that $\sum_{B:i,j\in B} \lambda_B \geq 1$ and $\sum_{B:i\notin B, j\notin B} \lambda_B \geq K - 1$ in the master (K is the number of available bins); there are two column classes and two associated pricing problems, the first consider solutions where $x_i = x_j = 1$ and the second solutions where $x_i = x_j = 0$. In Node 2, branching is implemented by requiring $\sum_{B:i\in B, j\notin B} \lambda_B \geq 1$ and $\sum_{B:i\notin B} \lambda_B \geq K - 1$ in the master and there are 2 pricing problems, the first consider solutions where $x_i = 1$ and $x_j = 0$ and the second solutions where $x_i = 0$. At subsequent branch-and-price nodes, branching is implemented by further partitioning existing column classes (for details see Vanderbeck (2010)).

3.3. Column generation based primal heuristic

As it was showed by previous research and by our own computational experiments, the set covering formulation is a very tight formulation that provides quite good dual bounds for the BPPC. Therefore, combined with a good primal heuristic, the column generation approach can be a very successful algorithm.

We use a generic diving heuristic which is a depth-first heuristic search in the branch-and-price tree that is presented in Joncour et al. (2010). Here, the branch-and-price enumeration is not driven by the branching scheme of Section 3.2, but simply by fixing λ_B variables. At each branch-and-price node, the master is solved by column generation, then a branch corresponding to rounding-to-one a λ_B variable is selected heuristically based on a greedy strategy. The master is then updated: deleting rows of (2) associated to items already covered and deleting columns covering those items. The master is re-optimized with a limit on the number of column generation iterations and the process is reiterated.

The solution obtained through the initial depth first exploration of the tree is considered as a reference incumbent solution. To further explore the solution space, we use limited backtracking as a diversification mechanism as developed in Joncour et al. (2010). This generic primal heuristic implemented in the software platform BaPCod relies on the concept of Limited Discrepancy Search (Harvey and Ginsberg, 1995). Specifically, we avoid choosing columns in a tabu list that consists of columns selected in previous branches from which we wish to diversify the search. The tabu list of columns at a branch-and-price node is the union of the tabu list of its ancestor and the columns chosen in previous child nodes of the ancestor. The tabu list of the root node is empty. A node which is not the first child node of its ancestor is explored only if the size of its tabu list is smaller or equal to `maxDiscrepancy` and its depth is smaller or equal to `maxDepth`, where `maxDiscrepancy` and `maxDepth` are two control parameters. In our implementation, we set parameters `maxDiscrepancy` $= 2$ and `maxDepth` $= 3$. The resulting search tree is illustrated in Figure 2.

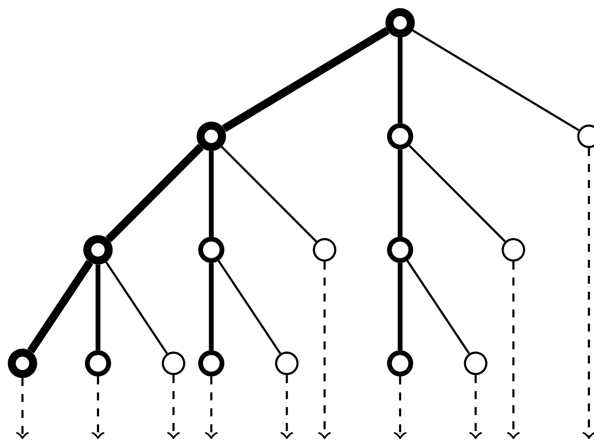

Figure 2: The search tree of the diving heuristic with parameters `maxDepth` $= 3$, `maxDiscrepancy` $= 2$; a doted line denotes a pure dive down in the branch-and-price tree.

4. Computational experiments

Our algorithm was developed using the software platform BaPCod — a generic Branch-and-Price Code. BaPCod is a library of C++ classes developed within the INRIA research team RealOpt at the University of Bordeaux. Our algorithm relies on the generic features of the solver for the branching scheme, the primal heuristic, and basic preprocessing. Therefore, the only application specific implementation consists in providing the problem formulation and the oracles for solving the pricing problem.

4.1. Instances with interval conflict graphs

We tested our procedure on instances obtained using the generation procedure of Gendreau et al. (2004); the latter is based on the bin packing test instances of Falkenauer (1996). There are 8 classes of instances. In the first four classes denoted below by "u", the items have an integer weight uniformly distributed in the range $[20, 100]$ while bins have capacity 150. The number n of items takes value in $\{120, 250, 500, 1000\}$. The next four classes referenced below by "t" (for "triplets") involve items with weights w_i uniformly distributed in the range $[250, 500]$, to be packed into bins of capacity $W = 1000$. Items are generated by triplet: every third item, i_{3s}, has a weight $w_{i_{3s}} = W - w_{i_{3s-1}} - w_{i_{3s-2}}$ for $s = 1, \ldots, \frac{n}{3}$. Thus, an optimal solution requires $\frac{n}{3}$ bins that are filled at full capacity with exactly three items. The number of items is, respectively, $n = 60$, 120, 249 and 501. Conflict graphs are characterized by different density values δ, varying from 0.1 to 0.9. This is done by assigning a value ρ_i, to each vertex, $i \in V$, according to a continuous uniform distribution in $[0, 1]$. Then, a conflict is created for item pair (i, j) if $(\rho_i + \rho_j)/2 \leq \delta$. For each of the 8 classes and each $\delta \in \{0.1, 0.2, \ldots, 0.9\}$, 10 of 20 instances were generated.

Observe that the above conflict generation scheme results in an interval conflict graph. Assume that items are indexed in the non-increasing order of their values ρ_i. An interval representation of such graph is similar to the one depicted in Figure 3. Therein, the intervals I_i associated to each item/vertex $i \in V$ are shown from the bottom to the top in order of their index number, i.e., in the non-increasing order of their values ρ_i. The definition of intervals I_i formalizes the condition that a conflict exists if $(\rho_i + \rho_j)/2 \leq \delta$. If two items, i and j have both ρ-value larger or equal to δ they cannot be in conflict; hence, they have non-overlapping intervals. Let $L = \{i : \rho_i > \delta\}$ and define interval $I_i = (i - 1, i)$ for items $i \in L$. If two items, i and j have both ρ-value smaller or equal to δ they must be in conflict;

hence, they have overlapping intervals. For items $j \in V \setminus L$, one can set $I_j = (a_j, |L| + 1)$ where $a_j = \min\{i - 1 : i \in L \text{ and } \rho_i + \rho_j \leq 2\delta\}$. Note that all these intervals overlap on $(|L|, |L| + 1)$. Now consider two items $i, j \in V$ with $\rho_i > \delta$, $\rho_j \leq \delta$, and therefore $i < j$ and note that their interval overlap only if $\rho_i + \rho_j \leq 2\delta$, with the property that if I_i overlaps with I_j, then it must overlap with every I_l such as $j < l$.

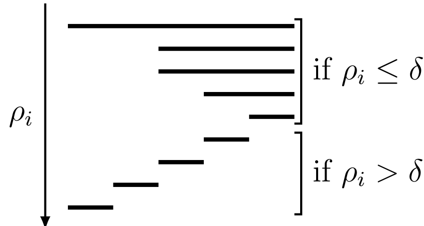

Figure 3: Structure of the interval representation of the conflict graphs

We have therefore showed that an interval definition exists that yields the desired conflict graph. However, for the purpose of our specialized dynamic programming algorithm, one only needs the values of $prev_i$ of Definition 1. They can be obtained as follows:

$$prev_i = \begin{cases} i - 1, & \rho_i > \delta, \\ \max\{j : \ (\rho_j + \rho_i)/2 > \delta\}, & \rho_i \leq \delta \end{cases}$$

assuming items are indexed in the non-increasing order of their values ρ_i.

In our numerical experiments, we first compared our algorithm and the algorithm of Fernandes Muritiba et al. (2010), which we denote FMIMT. The comparison was carried on the test instances that were kindly provided to us by these authors. For each class and each density, there are 10 instances. Hence, there are 90 instances for each class. In comparison to FMIMT, we tested three versions of our algorithm: (1) using our branch-and-price approach with specialized DP pricing but without the primal heuristic, (2) running the algorithm with the column generation based primal heuristic, but relying on the branch-and-bound oracle for pricing in a general conflict graph instead of the specialized DP pricing oracle for interval graphs (3) the full-blown branch-and-price approach with DP pricing and primal heuristic. Our algorithm was run using one thread on a Dell PowerEdge T300 workstation with an Intel Xeon X5460 3.16 GHz processor. Algorithm FMIMT was run on a Pentium IV 3 GHz processor. By `www.spec.org`, our machine is roughly 3.75 times faster. Therefore, **we multiplied our computing time by 3.75** for the purpose of this comparison.

In Table 1, we report the number of unsolved instances within the time limit, denoted ¬opt, out of 90 instances (except for our algorithm with the primal heuristic that solved all

class	FMIMT		Our w/o heur.		Our w/o DP sp		Our w DP sp & heur.	
	¬opt	av. time	¬opt	av. time	av. time	max. time	av. time	max. time
t60	0	38.7	0	1.0	0.9	27.9	0.8	6.5
t120	5	1860.3	1	156.0	26.9	1971.3	37.8	2956.4
t249	4	1582.1	2	334.2	30.0	235.5	29.3	130.6
t501	4	3163.6	0	245.9	407.7	2818.7	189.1	960.8
u120	0	29.4	0	3.4	2.6	55.7	2.8	26.2
u250	0	107.1	0	23.9	13.5	32.4	12.5	35.9
u500	5	2195.4	1	318.0	132.2	501.2	70.3	154.9
u1000	2	1911.9	0	1401.2	940.2	3335.6	437.6	1133.1

Table 1: Comparison of our algorithm with the algorithm of Fernandes Muritiba et al. (2010)

instances to optimality); the average solution time; and the maximum solution time for our algorithm with primal heuristic. The time limit was 10 hours for FMIMT and 1 hour for our algorithm. Our algorithm with the primal heuristic solved all instances to optimality and it is faster by an order of magnitude than FMIMT. Using the heuristic allowed us to solve 4 more instances and it speeds up our algorithm considerably. Additionally, we observed that our root node lower bound was equal to the optimal solution for all instances but 3. All but 4 instances were solved at the root node (thanks to the primal heuristic). Note also that using specialized DP pricing oracle reduces the running time of the algorithm only for large instances (500 items and more).

Secondly, we compare the above three versions of our algorithms (without and with DP pricing or primal heuristic) to the algorithm of Elhedhli et al. (2011), which we denote ELGN. The comparison was carried on the test instances that were kindly provided to us by these authors. For each class and each density, there are 20 instances. So, we have 180 instances for each class. The comparison involve results obtained on different computers: algorithm ELGN was run on a Sun Blade 2500 workstation with an Ultrasparc IIIi 1.6 GHz processor. By `www.spec.org`, our machine is roughly six times faster. Therefore, in this comparison, **the solution time of our algorithm is multiplied by 6**. The time limit was 1 hour for ELGN and 10 minutes for our algorithm (the ELGN time limit was divided by 6).

In Table 2, we report the number of unsolved instances within the time limit, denoted ¬opt (except for our algorithm with primal heuristic that solved all instances to optimality); the average solution time; and the maximum solution time. The algorithm ELGN includes a primal heuristic: a rounding procedure with no backtracking. Results of Table 2 indicates that our algorithm with DP pricing and primal heuristic is an order of magnitude faster. All

	ELGN		Our w/o heur.		Our w/o DP sp		Our w DP sp & heur.	
class	¬opt	av.time	¬opt	av.time	av.time	max.time	av.time	max.time
t60	0	3.2	0	0.9	0.9	8.9	1.3	7.7
t120	3	118.6	0	5.0	5.5	34.0	6.9	30.0
t249	10	398.0	4	157.1	65.0	1024.9	53.8	383.2
u120	0	47.0	0	2.4	3.2	15.4	3.7	9.4
u250	1	183.1	1	37.0	21.5	99.5	21.2	73.3
u500	13	1253.8	5	277.5	214.6	1479.9	115.2	310.4

Table 2: Comparison of our algorithm with the algorithm of Elhedhli et al. (2011)

instances were solved at the root node thanks to the primal heuristic. The root node lower bound is equal to the optimal solution for all tested instances.

4.2. Instances with arbitrary conflict graphs

As there are no test instances of the problem with an arbitrary conflict graph available in the literature, we generated them ourselves. First, we took the same instances as those used above, but generated the conflict graphs randomly in the following way. We begin with the empty graph. We iteratively select an item pair (i, j) at random (with uniform distribution); then edge (i, j) is added to the graph if it is not already defined. The procedure is interrupted as soon as the desired graph density is reached.

The resulted eight classes are referenced below by "ta" and "ua". For each class and each density, there are 10 instances. So, we have 90 instances per class. We tested our algorithm with and without the primal heuristic on these instances. In Table 3, we compare the number of unsolved instances within 1 hour, denoted ¬opt; the average solution time (only for the solved instances); and the average remaining gap (only for the unsolved instances). The exact method does not typically produce feasible solutions until it identifies the optimum. Thus, when the heuristic is not used and the algorithm does not solved the problem to optimality, then generally no feasible solution is available. Therefore, for the version of the algorithm without the primal heuristic, the gap statistics are not provided. For comparison purposes, Table 3 also reproduces on the right-hand the results for instance classes "u" and "t" with interval conflict graph (therein our computing times are not multiplied by a correction factor anymore).

As shown in Table 3, instances with arbitrary conflict graphs are significantly harder to solve than the instances with interval conflict graphs. For arbitrary conflict graphs, the primal heuristic helps a lot when solving instances in class "ua". On the contrary, it increases

	General Conflict Graph					Interval C G	
	Our algo w/o heur		Our algo with heur.			Our algo with heur.	
class	not opt.	av. time	not opt.	av. time	gap	class	av. time
ta60	0	0.3	0	0.2	0%	t60	0.2
ta120	0	2.3	0	4.2	0%	t120	4.1
ta249	7	97.6	6	137.3	1.2%	t249	8.6
ta501	23	215.0	25	392.6	0.6%	t501	50.4
ua120	0	1.4	0	0.7	0%	u120	0.7
ua250	1	41.2	2	9.0	1.0%	u250	3.5
ua500	27	234.3	8	39.0	0.5%	u500	19.1
ua1000	33	713.0	8	286.2	0.3%	u1000	116.7

Table 3: Results obtained by our algorithm on instances with arbitrary conflict graphs and comparison of solution time on instances with interval conflict graphs

the running time when solving class "ta" instances. However, it guarantees to obtain a good feasible solution. Solving pricing problems using the depth-first-search branch-and-bound algorithm takes 32.9% of the overall running time on the average.

4.3. Instances with a larger number of items per bin

Observe that, in previous classes of instances, the number of items per bin does not exceed 3 on average. We generated additional classes of more difficult instances denote below by "d" and "da". They consist of items with integer weights uniformly distributed in the range $[500, 2500]$, to be packed into bins of capacity $W = 10000$. In class "d", conflict graphs are interval graphs. They were generated using the same procedure as for classes "t" and "u". In class "da", conflict graphs are arbitrary, as in classes "ta" and "ua". The number n of items is 120, 250, and 500. There are on average 8 items per bin. For each class and each density, there are 10 instances. So, we have 90 instances per class again. We tested our algorithm with the primal heuristic on these instances. In Table 4, we report the number of unsolved instances within 1 hour, denoted $\neg$opt, (out of 90), the average solution time (only for the solved instances), and the average remaining gap (only for the unsolved instances).

Our algorithm solved all class "d" instances. As for class"u" and "t", the lower bound provided by the column generation procedure are very tight: it was always equal to the optimum solution. The primal heuristic is very good: it found optimum solutions for all but 3 instances. Our numerical experiment revealed that, for these instances where the bin capacity is large, the dynamic programming algorithm was slower than the depth-first-search branch-and-bound algorithm developed the general KPC. Hence, we used the latter. The

Our algorithm with primal heuristic							
class	¬opt	av. time	gap	class	¬opt	av. time	gap
d120	0	8.9	0.0%	da120	23	23.2	4.7%
d250	0	53.4	0.0%	da250	40	23.3	3.7%
d500	0	486.8	0.0%	da500	41	137.6	3.9%

Table 4: Results obtained by our algorithm on hard instances with interval and arbitrary conflict graphs

solution time for instances "d" are therefore much larger than for instances "u" and "t". The increased computing time is also due to a slower convergence of the column generation algorithm. In our experiments the oracle for the KPC took on the average only 18.6% of the overall computation time. (The increasing computing times explain why we did not test instances in class "d" and "da" with 1000 jobs.)

The performance of our algorithm for class "da" instances is much worse than for other tested classes. Slightly less than half of the instances remains unsolved after one hour of computation time. The remaining gap is almost an order of magnitude larger than for class "ta" and class "ua" instances. The difficulty of these instances depends a lot on the conflict graph density. Details on this can be found in Section 4.5.

4.4. Efficiency of the primal heuristic

In this subsection, we present the results obtained using the primal heuristic only, without branching. We tested two variants of the heuristic: a pure diving approach (DH) without the Limited Discrepancy Search (meaning that the `maxDiscrepancy` parameter is equal to 0) and the variant used in the above test (DH with LDS) with the parameters `maxDiscrepancy` $= 2$ and `maxDepth` $= 3$.

	"u"		"t"		"ua"		"ta"		"da"	
#items	time	gap	time	gap	time	gap	time	gap	time	gap
60			0.2	0.21%			0.1	0.22%		
120	0.6	0.12%	0.9	0.45%	0.6	0.18%	0.7	0.71%	1.0	3.20%
250	3.1	0.11%	5.7	0.31%	3.7	0.14%	4.0	0.69%	7.6	2.79%
500	17.8	0.04%	34.7	0.16%	28.1	0.12%	38.2	0.47%	73.7	2.35%
1000	110.0	0.02%			205.1	0.08%			760.8	1.88%

Table 5: Results with the pure diving heuristic DH alone.

In Tables 5 and 6, we present, for all the above instance classes, the average running time of the heuristics and the average gap of the solutions found in per cent from the best

#items	"u" time	"u" gap	"t" time	"t" gap	"ua" time	"ua" gap	"ta" time	"ta" gap	"da" time	"da" gap
60			0.2	0%			0.2	0%		
120	0.7	0%	1.2	0.01%	0.7	0%	1.5	0.05%	6.0	1.25%
250	3.5	0%	8.6	0%	6.2	0.03%	23.7	0.32%	60.0	1.69%
500	19.0	0%	50.4	0%	63.1	0.05%	271.3	0.28%	541.5	1.81%
1000	116.7	0%			516.8	0.03%			6036.3	1.48%

Table 6: Results with our diving heuristics with Limited Discrepancy Search, DH LDS.

known dual bound. The pure diving heuristic (DH) produces solutions of a high quality. The Limited Discrepancy Search (in DH LDS) improves significantly the performance of the heuristic, especially for instances of classes "u", "t", and "ua". A disadvantage of our primal heuristic is that its running time increases rapidly with the number of items; this can be noted specially for class "da" that are the most difficult instances.

Fernandes Muritiba et al. (2010) have proposed a population based heuristic (PH) for the problem. It consists in a tabu search algorithm and a diversification procedure. In Table 7, we compare the two variants of our primal heuristic with PH, the heuristic of Fernandes Muritiba et al. (2010). As it was done above, the running time of our heuristics is multiplied by 3.75 to compensate the difference in the computers speed. The results of Table 7 show that DH is faster than PH for instances with less than 500 items and produce on average significantly better solutions than the population heuristic (except for class "u120"). DH LDS is only slightly slower than the PH and produces optimal solutions for all instances except one.

class	PH gap	PH time	DH gap	DH time	DH LDS gap	DH LDS time
t60	0.45%	37.5	0.18%	0.7	0%	0.8
t120	0.62%	40.0	0.48%	3.5	0.03%	5.1
t249	0.39%	51.9	0.29%	21.3	0%	29.3
t501	0.21%	58.9	0.16%	130.3	0%	189.1
u120	0.10%	22.4	0.16%	2.3	0%	2.8
u250	0.21%	52.1	0.10%	11.8	0%	12.5
u500	0.20%	69.7	0.05%	66.2	0%	70.3
u1000	0.22%	107.8	0.02%	412.5	0%	437.6

Table 7: Comparison of our primal heuristics with the population heuristic of Fernandes Muritiba et al. (2010).

4.5. Impact of the density of the graph to the results

	"u"	"t"	"ua"			"ta"			"da"		
density	time	time	¬opt	time	gap	¬opt.	time	gap	¬opt	time	gap
10%	15.5	6.1	0%	71.1	0%	0%	43.5	0%	0%	57.0	0.0%
20%	16.7	11.0	0%	67.7	0%	0%	50.8	0%	0%	41.7	0.0%
30%	19.5	**21.8**	0%	67.8	0%	0%	105.3	0%	0%	56.0	0.0%
40%	**30.1**	10.8	0%	71.4	0%	0%	171.2	0%	7%	44.7	1.3%
50%	26.8	9.8	0%	64.8	0%	18%	**481.7**	**0.9%**	20%	47.9	4.5%
60%	19.6	7.5	0%	76.5	0%	15%	80.3	0.7%	**90%**	**319.1**	3.3%
70%	15.8	5.7	5%	101.7	**0.7%**	20%	2.9	0.6%	**90%**	186.9	4.7%
80%	13.1	4.5	15%	100.2	0.5%	0%	9.7	0%	73%	107.2	**4.9%**
90%	10.5	3.0	**25%**	**115.9**	0.3%	**25%**	116.9	0.6%	67%	4.6	3.0%

Table 8: Impact of the conflict graph density on the results obtained by our branch-and-price algorithm.

We proceed to show how the difficulty of instances depends on the density of the conflict graph. Instances here are grouped according to their classes. Table 8 presents results obtained with our algorithm with the primal heuristic for different conflict graph density. We report the percentage of unsolved instances within 1 hour denoted "¬opt; the average solution time (only for the solved instances), and the average remaining gap (only for the unsolved instances). For classes "u" and "t", we present only the time statistic, as all these instances were solved to optimality.

One can observe that the impact of the graph density on the difficulty of instances highly depends on the graph class. The most difficult instances with interval conflict graph are with 50% density or slightly below. On the contrary, the most difficult instances with arbitrary conflict graph are with a high density. Highly oscillating results for instance class "ta" can be explained by their particular structure: there are threshold values for the density of the conflict graph which change a lot the quality of the column generation dual bounds.

In Tables 9 and 10, we present the impact of conflict graph density on the performance of the primal heuristics DH and DH LDS: we report the average running time and the average gap of the solutions found. The observation is similar to the one concerning the impact of density on the efficiency of the branch-and-price algorithm. The heuristic DH sometimes requires more time and produces solutions with larger relative gap for instances with small density. This is due to the larger running time of pricing oracle and the fact that the absolute solution values are smaller.

	"u"		"t"		"ua"		"ta"		"da"	
density	time	gap	time	gap	time	gap	time	gap	time	gap
10%	14.3	0.08%	3.2	**0.76%**	**70.0**	0.01%	**17.6**	0.65%	32.7	1.15%
20%	15.9	0.04%	3.8	0.72%	67.6	0.01%	16.2	0.75%	31.1	1.25%
30%	19.2	0.05%	5.3	0.45%	67.1	0.05%	12.7	0.81%	27.6	1.98%
40%	**28.1**	0.12%	**9.3**	0.37%	64.7	0.07%	11.1	0.42%	24.0	1.61%
50%	24.7	**0.22%**	9.1	0.15%	63.6	0.08%	9.2	0.42%	21.3	1.64%
60%	18.3	0.11%	6.8	0.17%	59.2	0.18%	8.5	0.13%	25.6	3.98%
70%	14.5	0.07%	5.2	0.07%	54.8	0.21%	7.1	0.30%	**45.9**	**5.45%**
80%	12.7	0.03%	4.1	0.03%	47.8	0.24%	8.1	0.42%	26.5	5.14%
90%	9.4	0.01%	2.8	0.03%	39.4	**0.34%**	6.5	**0.82%**	12.4	2.82%

Table 9: Impact of the conflict graph density on the results obtained using heuristic DH.

	"u"		"t"		"ua"		"ta"		"da"	
density	time	gap	time	gap	time	gap	time	gap	time	gap
10%	15.5	0%	6.1	0%	74.7	0%	51.2	0%	92.5	0.04%
20%	16.7	0%	11.0	0%	74.4	0%	42.8	0%	84.5	0.09%
30%	19.5	0%	**14.0**	**0.03%**	69.1	0%	72.3	0.11%	36.7	0.00%
40%	**30.1**	0%	10.8	0%	69.7	0%	112.8	0.27%	79.8	0.04%
50%	26.8	0%	9.8	0%	66.7	0%	**120.8**	**0.42%**	81.0	0.71%
60%	19.6	0%	7.5	0%	81.5	0%	80.1	0.13%	384.4	2.99%
70%	15.8	0%	5.7	0%	106.9	0.04%	86.3	0.12%	**543.4**	**4.27%**
80%	13.1	0%	4.5	0%	171.8	0.06%	9.4	0.12%	348.2	3.81%
90%	10.5	0%	3.0	0%	**605.5**	**0.16%**	91.7	0.29%	170.7	2.31%

Table 10: Impact of the conflict graph density on the results obtained using heuristic DH LDS.

These results also allow us to give the following recommendations. For the instances of classes "ua" and "da", the pure diving heuristic should be used if the graph density is large. For other instances with an arbitrary conflict graph, the DH LDS requires a reasonable increase in the running time but produces significantly better solutions. For instances with an interval conflict graph, the DH LDS should always be used.

5. Conclusions

In this paper, we present a branch-and-price algorithm for the bin packing problem with conflicts that we implemented using the software platform BaPCod. The only problem specific "features" of our implementation are the formulations and the oracles for solving the pricing problem. Our algorithm was tested on instances from the literature and newly generated ones. Our computational results can be summarized as follows:

- On instances from the literature, our algorithm outperforms the existing algorithms. These instances are rather specific: the number of items in a bin is small (3 on average) and the conflict graph is an interval graph. Here, our algorithm gives comparatively better results even when a generic branch-and-bound pricing oracle is used, i.e. when the interval graph structure is not exploited.

- Instances where the solution involves a higher number of items per bin and the conflict graph has no special structure, are much harder. In particular, the linear relaxation bound stemming from the set covering formulation is not as tight when the conflict graph is not an interval graph.

- The generic column generation based primal heuristic built into BaPCod contributes a lot to the success of our algorithm, and compares favorably with the population based heuristics from the literature. The Limited Discrepancy Search improves significantly the efficiency of the diving heuristic.

- The generic BaPCod solver is a competitive tool once a problem specific oracle is provided for solving the pricing problem.

In addition to the new benchmarks, the highlights of our study are:

- a novel dynamic programming algorithm of complexity $O(nW)$ for the knapsack problem with an interval conflict graph;

- a depth-first-search branch-and-bound algorithm for the Knapsack Problem with Conflicts that has proved to be efficient in practice and outperforms the CPLEX 11.0 solver on instances with conflict graphs of density 10% and more;

- an illustration of the interest of exploiting structure of solved instances (the fact that standard BPPC test instances of the literature had interval conflict graphs was not noticed in the previous research work).

References

Baker, Brenda S., Edward G. Coffman. 1996. Mutual exclusion scheduling. *Theoretical Computer Science* **162** 225 – 243.

Beaumont, Olivier, Nicolas Bonichon, Philippe Duchon, Hubert Larchevêque. 2008. Distributed approximation algorithm for resource clustering. *Structural Information and Communication Complexity*, *Lecture Notes in Computer Science*, vol. 5058. Springer Berlin / Heidelberg, 61–73.

Busygin, Stanislav. 2006. A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics* **154** 2080–2096.

Carraghan, Randy, Panos M. Pardalos. 1990. An exact algorithm for the maximum clique problem. *Operations Research Letters* **9** 375 – 382.

Christofides, N., A. Mingozzi, P. Toth. 1979. The vehicle routing problem. N. Christofides, A. Mingozzi, P. Toth, C. Sandi, eds., *Combinatorial optimization*. Wiley, Chichester, 315–338.

Corneil, Derek G., Stephan Olariu, Lorna Stewart. 1998. The ultimate interval graph recognition algorithm? *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 175–180.

Elhedhli, Samir, Lingzi Li, Mariem Gzara, Joe Naoum-Sawaya. 2011. A Branch-and-Price Algorithm for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing* **23** 404–415.

Falkenauer, Emanuel. 1996. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* **2** 5–30.

Fernandes Muritiba, Albert E., Manuel Iori, Enrico Malaguti, Paolo Toth. 2010. Algorithms for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing* **22** 401–415.

Gardi, Frédéric. 2005. Ordonnancement avec exclusion mutuelle par un graphe d'intervalles ou d'une classe apparentée : complexité et algorithmes. Ph.D. thesis, Université de la Méditerranée-Aix-Marseille II, Marseille, France.

Gardi, Frédéric. 2009. Mutual exclusion scheduling with interval graphs or related classes. part i. *Discrete Applied Mathematics* **157** 19–35.

Gendreau, Michel, Gilbert Laporte, Frédéric Semet. 2004. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers and Operations Research* **31** 347 – 358.

Harvey, William D., Matthew L. Ginsberg. 1995. Limited discrepancy search. *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 607–613.

Hifi, Mhand, Mustapha Michrafy. 2007. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers and Operations Research* **34** 2657 – 2673.

Jansen, Klaus. 1999. An approximation scheme for bin packing with conflicts. *Journal of Combinatorial Optimization* **3** 363–377.

Jansen, Klaus, Sabine Öhring. 1997. Approximation algorithms for time constrained scheduling. *Information and Computation* **132** 85 – 108.

Johnson, Ellis L., Manfred W. Padberg. 1981. A note on the knapsack problem with special ordered sets. *Operations Research Letters* **1** 18–22.

Joncour, Cédric, Sophie Michel, Ruslan Sadykov, Dmitry Sverdlov, François Vanderbeck. 2010. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics* **36** 695 – 702.

Kelleler, Hans, Ulrich Pferschy, David Pisinger. 2004. *Knapsack problems*. Springer-Verlag Berlin.

Laporte, Gilbert, Sylvain Desroches. 1984. Examination timetabling by computer. *Computers and Operations Research* **11** 351 – 360.

Martello, Silvano, Paolo Toth. 1990. *Knapsack problems: algorithms and computer implementations*. John Wiley and Sons, Inc., New York, NY, USA.

Pferschy, Ulrich, Joachim Schauer. 2009. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications* **13** 233–249.

Ryan, D. M., B. A. Foster. 1981. An integer programming approach to scheduling. A. Wren, ed., *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, 269–280.

Vanderbeck, François, Martin W. P. Savelsbergh. 2006. A generic view of dantzig-wolfe decomposition in mixed integer programming. *Operations Research Letters* **34** 296–306.

Vanderbeck, François. 2010. Branching in branch-and-price: a generic scheme. *Mathematical Programming* Online first.

Vanderbeck, François, Laurence A. Wolsey. 2010. Reformulation and decomposition of integer programs. Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, Laurence A. Wolsey, eds., *50 Years of Integer Programming 1958-2008*. Springer Berlin Heidelberg, 431–502.