A Branch-and-Price Algorithm for the Bin Packing Problem with Conflicts

Ruslan Sadykov\textsuperscript{1} François Vanderbeck\textsuperscript{1,2}

\textsuperscript{1}INRIA — Bordeaux Sud-Ouest, ReAlOpt team
\textsuperscript{2}Université Bordeaux I

ROADEF, February 25, 2010
Contents

Introduction

The algorithm

Résultats
Problem definition

▶ Given
  ▶ an infinite number of bins of size $W$,
  ▶ a set $N = \{1, \ldots, n\}$ of items $i$ of sizes $w_i$,
  ▶ a graph $G = (N, E)$ of conflicts between items.
▶ Pack the items into a minimum number of bins.
▶ Two items in conflict cannot be in the same bin.
Motivations

Mutual exclusion scheduling (generalisation)

(Baker & Coffman, 96), (Gardi, 05)

temps

- Total length of tasks assigned to a person $\leq W$
- Overlapping tasks cannot be assigned to the same person: conflict graph is interval

Other applications

- Examination scheduling (Laporte & Desroches, 84)
- Parallel computation (Jansen, 99), (Beaumont et al., 08)
- Product delivery (Christofides et al., 79)
IP formulation

Variables:

- $y_k = 1$ if bin $k$ is used, otherwise $y_k = 0$
- $x_{ik} = 1$ if item $i$ is put to bin $k$, otherwise $x_{ik} = 0$

$$\min \sum_{k \in K} y_k$$

$$\sum_{k \in K} x_{ik} = 1, \quad i \in N,$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \quad k \in K,$$

$$x_{ik} + x_{jk} \leq y_k, \quad (i, j) \in E, k \in K,$$

$y_k \in \{0, 1\}, \quad k \in K,$

$x_{ik} \in \{0, 1\}, \quad i \in N, k \in K.$
Contents

Introduction

The algorithm

Résultats
Set covering formulation

\( B = \) set of valid single bin packings
\( \lambda_b = 1 \) if subset \( b \in B \) of items occupies a bin

Formulation

\[
\min \sum_{b \in B} \lambda_b \\
\sum_{b \in B : i \in b} \lambda_b \geq 1, \quad i \in N, \\
\lambda_b \in \{0, 1\}, \quad b \in B.
\]

Pricing problem

\[
\max \sum_{i \in N} \pi_i x_i \\
\sum_{i \in N} w_i x_i \leq W, \\
x_i + x_j \leq 1, \quad (i, j) \in E, \\
x_i \in \{0, 1\}, \quad i \in N.
\]

(Knapsack with conflicts)
Knapsack with conflicts: existent approaches

Arbitrary conflict graphs
NP-hard, \( \approx 100 \) seconds for solving exactly a 1000-items instance (Hifi, Michrafy, 07) — slow.

Structured conflict graphs

- Trees and chordal graphs: dynamic programming algorithm with complexity \( O(nW^2) \) (Pferschy, Shauer, 09) — slow.
- Interval graphs: these instances we can solve fast.
Interval conflict graphs: dynamic programming

$P(i, w)$ — solution value of the subproblem with the first $i$ items and bin size $w$

$$P(i, w) = \max \left\{ P(prec_i, w - w_i) + p_i, 
\quad P(i - 1, w) \right\}$$

Complexity: $O(nW)$, same as for the knapsack without conflicts!
Interval conflict graphs: enumeration algorithm

A simple Branch-and-Bound algorithm, extension of (Carraghan & Pardalos, 90):

- Dual bound: relaxation of conflicts between non-fixed items and integrality.
- Depth-first search.
- Branching:

\[
\begin{align*}
\pi_i & = 1 \\
x_{i_1} & = 1 \\
x_{i_2} & = 0 \\
x_{i_3} & = 1 \\
x_{i_4} & = 0 \\
\end{align*}
\]

- Time we spent at each node explored is \( O(n) \).
Generic branching scheme

\[ \lambda \text{ is fractional} \iff \text{exists a pair } i, j \text{ such that } \sum_{i,j \in b} \lambda_b \neq \{0, 1\} \]

Branching: either item \( i \) and \( j \) are in the same bin or not, we add constraints to the pricing subproblem.

Ryan&Foster scheme

- \( x_j = x_i \)
- \( x_j + x_i \leq 1 \)

The interval conflict graph structure can be broken.

Generic branching scheme

- **Same bin**: either \( x_i = x_j = 0 \) or \( x_i = x_j = 1 \)
- **Different bins**: either \( x_i = 0 \) or \( x_i = 1, x_j = 0 \)

- Does not brake graph structure.
- Better LP bound after branching.
- More subproblems to solve.
“Diving” rounding generic heuristic

Will be presented at ISCO’10 : (Joncour et al., 10)

Combines :

- Depth-first search : we choose a column \( (\lambda_B \leftarrow 1) \) and change the right-hand side of the formulation
- The number of generated columns at each node (except the root is limited)
- Diversification (Limited Discrepancy Search)
- Pre-processing

\[ \text{MaxDepth} = 2 \]
\[ \text{MaxDiscrepancy} = 2 \]
Implementation

We used **BaPCod** (a generic **Branch-and-Price Code**) being developed by ReAlOpt

By default, we have:

- Stabilized column generation procedure
- Generic branching
- Generic “diving” heuristic

The user supplies:

- Compact problem formulation
- An oracle for solving the pricing problem
Instances de test

Due to (Gendreau, Laporte, and Semet, 04):

- Sizes (integer):
  - Uniforms ($u$): $w_i \in U[20, 100]$, $W = 150$.
  - Triples ($t$): $w_i \in U[250, 500]$ (in triples), $W = 1000$.

- “Conflictness” of items: $p_i \in U[0, 1]$.

- Conflict graph density: $\delta \in \{0, 0.1, \ldots, 0.9\}$.

- Conflict graphs structure:

\[
(i, j) \in E \text{ iff } \frac{p_i + p_j}{2} \geq 1 - \delta
\]

\[
p_i \geq 1 - \delta
\]

\[
p_i < 1 - \delta
\]
Comparison of exact algorithms

MIMT : (Muritiba, Iori, Malaguti, and Toth, 09)
ELGN : (Elhedhli, Li, Gzara, and Naoum-Sawaya, 09)

The times adjusted using www.spec.org

<table>
<thead>
<tr>
<th>class</th>
<th>MIMT</th>
<th>ELGN</th>
<th>Notre</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>not opt.</td>
<td>time</td>
<td>not opt.</td>
</tr>
<tr>
<td>t60</td>
<td>0%</td>
<td>61</td>
<td>0%</td>
</tr>
<tr>
<td>t120</td>
<td>5%</td>
<td>2976</td>
<td>3%</td>
</tr>
<tr>
<td>t249</td>
<td>4%</td>
<td>2581</td>
<td>11%</td>
</tr>
<tr>
<td>t501</td>
<td>4%</td>
<td>5060</td>
<td></td>
</tr>
<tr>
<td>u120</td>
<td>0%</td>
<td>46</td>
<td>0%</td>
</tr>
<tr>
<td>u250</td>
<td>0%</td>
<td>171</td>
<td>1%</td>
</tr>
<tr>
<td>u500</td>
<td>5%</td>
<td>3512</td>
<td>14%</td>
</tr>
<tr>
<td>u1000</td>
<td>2%</td>
<td>3057</td>
<td></td>
</tr>
</tbody>
</table>
## Comparison of heuristics

- **PH**: population heuristic based on tabu search *(Muritiba et al., 09)*
- **DH**: “diving heuristic” (without LDS)
- **DH LDS**: “diving heuristic” (with LDS)

<table>
<thead>
<tr>
<th>class</th>
<th>PH gap</th>
<th>PH time</th>
<th>DH gap</th>
<th>DH time</th>
<th>DH LDS gap</th>
<th>DH LDS time</th>
</tr>
</thead>
<tbody>
<tr>
<td>t60</td>
<td>0.45%</td>
<td>60</td>
<td>0.11%</td>
<td>2</td>
<td>0.00%</td>
<td>2</td>
</tr>
<tr>
<td>t120</td>
<td>0.62%</td>
<td>64</td>
<td>0.66%</td>
<td>4</td>
<td>0.00%</td>
<td>9</td>
</tr>
<tr>
<td>t249</td>
<td>0.39%</td>
<td>83</td>
<td>0.35%</td>
<td>8</td>
<td>0.00%</td>
<td>51</td>
</tr>
<tr>
<td>t501</td>
<td>0.21%</td>
<td>94</td>
<td>0.16%</td>
<td>17</td>
<td>0.00%</td>
<td>280</td>
</tr>
<tr>
<td>u120</td>
<td>0.10%</td>
<td>36</td>
<td>0.18%</td>
<td>4</td>
<td>0.00%</td>
<td>5</td>
</tr>
<tr>
<td>u250</td>
<td>0.21%</td>
<td>83</td>
<td>0.07%</td>
<td>9</td>
<td>0.00%</td>
<td>23</td>
</tr>
<tr>
<td>u500</td>
<td>0.20%</td>
<td>112</td>
<td>0.03%</td>
<td>17</td>
<td>0.00%</td>
<td>120</td>
</tr>
<tr>
<td>u1000</td>
<td>0.22%</td>
<td>172</td>
<td>0.01%</td>
<td>34</td>
<td>0.002%</td>
<td>822</td>
</tr>
</tbody>
</table>
Open instances

10 instances which was not solved by (Muritiba et al., 09).

<table>
<thead>
<tr>
<th>Name</th>
<th>class</th>
<th># nodes</th>
<th>Sol.</th>
<th>Improv.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6_3_6</td>
<td>t120</td>
<td>609</td>
<td>41</td>
<td>0</td>
<td>3m45s</td>
</tr>
<tr>
<td>7_3_4</td>
<td>t249</td>
<td>1</td>
<td>83</td>
<td>-1</td>
<td>29s</td>
</tr>
<tr>
<td>8_3_4</td>
<td>t501</td>
<td>1</td>
<td>167</td>
<td>-2</td>
<td>53s</td>
</tr>
<tr>
<td>3_4_4</td>
<td>u500</td>
<td>1</td>
<td>204</td>
<td>-3</td>
<td>30s</td>
</tr>
<tr>
<td>3_4_5</td>
<td>u500</td>
<td>1</td>
<td>206</td>
<td>-1</td>
<td>29s</td>
</tr>
<tr>
<td>3_4_7</td>
<td>u500</td>
<td>1</td>
<td>208</td>
<td>-4</td>
<td>21s</td>
</tr>
<tr>
<td>3_4_8</td>
<td>u500</td>
<td>1</td>
<td>205</td>
<td>-1</td>
<td>25s</td>
</tr>
<tr>
<td>3_4_9</td>
<td>u500</td>
<td>1</td>
<td>196</td>
<td>-4</td>
<td>36s</td>
</tr>
<tr>
<td>4_4_8</td>
<td>u1000</td>
<td>1</td>
<td>404</td>
<td>-7</td>
<td>4m5s</td>
</tr>
<tr>
<td>4_4_10</td>
<td>u1000</td>
<td>1</td>
<td>397</td>
<td>-9</td>
<td>7m9s</td>
</tr>
</tbody>
</table>

Effect of the “diving” heuristic
Instances with arbitrary conflict graphs

- The same instances except that the conflicts were generated arbitrarily.
- Time limit: 1 hour.

<table>
<thead>
<tr>
<th>class</th>
<th>not opt.</th>
<th>time (opt.)</th>
<th>gap (not opt.)</th>
<th>nodes enum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>t60</td>
<td>0%</td>
<td>0</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>t120</td>
<td>0%</td>
<td>3</td>
<td></td>
<td>1316</td>
</tr>
<tr>
<td>t249</td>
<td>5.6%</td>
<td>135</td>
<td>1.1%</td>
<td>3418</td>
</tr>
<tr>
<td>t501</td>
<td>27.8%</td>
<td>476</td>
<td>0.6%</td>
<td>16479</td>
</tr>
<tr>
<td>u120</td>
<td>0%</td>
<td>1</td>
<td></td>
<td>255</td>
</tr>
<tr>
<td>u250</td>
<td>1.1%</td>
<td>11</td>
<td>1.0%</td>
<td>405</td>
</tr>
<tr>
<td>u500</td>
<td>8.9%</td>
<td>47</td>
<td>0.5%</td>
<td>481</td>
</tr>
<tr>
<td>u1000</td>
<td>11.1%</td>
<td>331</td>
<td>0.3%</td>
<td>398</td>
</tr>
</tbody>
</table>
Conclusions and perspectives

Conclusions

▶ Knapsack problem with interval conflict graph can be solved efficiently (and fast !) by dynamic programming.
▶ Generic branch-and-price solver BaPCod is competitive with specialized oracle.
▶ Very good performance of the generic “diving” heuristic.

Future research

▶ We need fast algorithms for the knapsack problem with arbitrary conflict graphs (can we do noticeably better that Cplex ?)
▶ Improvement of BaPCod (other generic primal heuristics, pre-processing,...)