Models and Algorithms for the Bin Packing Problem with Fragile Objects

Manuel A. Alba Martinez\textsuperscript{(1)}, François Clautiaux\textsuperscript{(2)}, Mauro Dell’Amico\textsuperscript{(1)}, Manuel Iori\textsuperscript{(1)}

\textsuperscript{(1)} – University of Modena and Reggio Emilia, Italy
\textsuperscript{(2)} – Université des Sciences et Technologies de Lille, France

CTW 2011 – Rome
Formal definition of BPPFO

In the *Bin Packing Problem with Fragile Objects* (BPPFO) we are given:
- $n$ items of weight $w_j$ and fragility $f_j$ ($j = 1, \ldots, n$)
- a large number of uncapacitated bins

Let $J(i)$ denote the set of items assigned to a bin $i$ in a given solution. A solution is feasible if for each bin $i$:

$$
\sum_{j \in J(i)} w_j \leq \min_{j \in J(i)} \{ f_j \}
$$

We seek the minimal number of bins needed to pack all items.
Formal definition of BPPFO

In the *Bin Packing Problem with Fragile Objects* (BPPFO) we are given:

- $n$ items of weight $w_j$ and fragility $f_j$ ($j = 1, \ldots, n$)
- a large number of uncapacitated bins

Let $J(i)$ denote the set of items assigned to a bin $i$ in a given solution. A solution is feasible if for each bin $i$

$$\sum_{j \in J(i)} w_j \leq \min_{j \in J(i)} \{ f_j \}$$
Formal definition of BPPFO

In the Bin Packing Problem with Fragile Objects (BPPFO) we are given:

- $n$ items of weight $w_j$ and fragility $f_j$ ($j = 1, \ldots, n$)
- a large number of uncapacitated bins

Let $J(i)$ denote the set of items assigned to a bin $i$ in a given solution. A solution is feasible if for each bin $i$

$$\sum_{j \in J(i)} w_j \leq \min_{j \in J(i)} \{f_j\}$$

We seek the minimal number of bins needed to pack all items.
Formal definition of BPPFO

In the *Bin Packing Problem with Fragile Objects* (BPPFO) we are given:

- $n$ items of weight $w_j$ and fragility $f_j$ ($j = 1, \ldots, n$)
- a large number of uncapacitated bins

Let $J(i)$ denote the set of items assigned to a bin $i$ in a given solution. A solution is feasible if for each bin $i$

$$\sum_{j \in J(i)} w_j \leq \min_{j \in J(i)} \{ f_j \}$$

We seek the minimal number of bins needed to pack all items.

The BPPFO generalizes the *Bin Packing Problem* (BPP)

- set all fragilities equal to the BPP bin capacity
- NP-hard
Problem Description

An example

Figure: (a) A BPPFO instance and (b) its optimal solution
Motivation: Sharing offices! ...
Motivation: Sharing offices! ...
... and assigning cellular calls to frequency channels

**Problem Description**

**Motivation**

Code Division Multiple Access (CDMA) systems:
- Limited number of frequency channels of large capacity
- Assigning many calls to a channel may produce interferences
- Each call is characterized by a *noise* and a *tolerance*
- Each call assigned to a channel where total noise $\leq$ call tolerance
... and assigning cellular calls to frequency channels

To model (CDMA) systems as a BPPFO it is enough to set

- frequency channel = bin
- call = item, noise = weight and tolerance = fragility
Related Literature

The literature on the BPPFO is still small:

  Approximation schemes for the on-line version

  Approximation schemes and a good lower bound
The literature on the BPPFO is still small:

  Approximation schemes for the on-line version

  Approximation schemes and a good lower bound

This talk summarizes two contributions:

- Clautiaux, Dell’Amico, Iori and Khanafer. Lower and upper bounds for the bin packing problem with fragile objects (2010, submitted)
  Benchmark, first computational results, bounds

- Alba Martinez, Clautiaux, Dell’Amico and Iori. Exact algorithms for the bin packing problem with fragile objects (2011, work in progress)
A Compact Model for the BPPFO

We sort items according to non-decreasing fragility and model the BPPFO as follows.

\[
\begin{align*}
\min & \quad \sum_{i=1}^{n} y_i \\
y_j + \sum_{i=1}^{j-1} x_{ji} &= 1 & j &= 1, \ldots, n \\
\sum_{j=i+1}^{n} w_j x_{ji} &\leq (f_i - w_i) y_i & i &= 1, \ldots, n \\
x_{ji} &\leq y_i & i &= 1, \ldots, n, j &= i + 1, \ldots, n \\
y_i &\in \{0, 1\} & i &= 1, \ldots, n \\
x_{ji} &\in \{0, 1\} & i &= 1, \ldots, n, j &= i + 1, \ldots, n.
\end{align*}
\]

**Variables**
- \(y_i = 1\) if item \(i\) is the item of smallest fragility in its bin
- \(x_{ji} = 1\) if item \(j\) is packed with item \(i\) \((i < j)\)
Fractional Lower Bounds

BPP relaxation:

\[ L_0 = \left\lceil \sum_{j=1}^{n} \frac{w_j}{f_{\text{max}}} \right\rceil \]

where \( f_{\text{max}} = \max_{j=1,...,n} \{f_j\} \). Worst case of \( L_0 \) is arbitrarily bad.
Fractional Lower Bounds

BPP relaxation:

\[ L_0 = \left\lceil \sum_{j=1}^{n} \frac{w_j}{f_{\text{max}}} \right\rceil \]

where \( f_{\text{max}} = \max_{j=1,\ldots,n} \{f_j\} \). Worst case of \( L_0 \) is arbitrarily bad

A tighter relaxation:

\[ L_1 = \left\lceil \sum_{i=1}^{n} \frac{w_j}{f_j} \right\rceil \]

Still, worst case of \( L_1 \) is arbitrarily bad: consider \( n \) items with \( w_j = n^{j-1} \) and \( f_j = n^j, j = 1, \ldots, n \). Optimal solution = \( n \) bins, whereas

\[ L_1 = \sum_{i=1}^{n} \frac{n^{j-1}}{n^j} = \sum_{i=1}^{n} \frac{1}{n} = 1. \]
A Better Lower Bound

Fractional lower bound $L_2$ by Bansal et al. (2009):

```
input : $n$ items sorted by non-decreasing fragility
output: Lower bound value $L_2$

$L_2 = 1$;
$f_{res} = f_1 - w_1$; (first bin)
for $j = 2$ to $n$ do
  if $w_j \leq f_{res}$ then
    $f_{res} = f_{res} - w_j$; (fill current bin)
  else
    $L_2 = L_2 + 1$; (open new bin)
    $w_{res} = w_j - f_{res}$;
    $f_{res} = f_j - w_{res}$;
  end
end

return $L_2$;
```

```
1 2
2 3 4
4 5 6
```
Proposition

The worst case performance of $L_2$ is $1/2$ and is tight.

Sketch of the proof:

- Build a heuristic solution from the assignment obtained by Alg. $L_2$
- Bin 1 has at most one fractional item: remove the two portions of this item from bins 1 and 2 and pack it alone in a new bin
- Reiterate with bin 2, now with at most one fractional item, and so on
- For each of the $L_2$ bins we open at most one new bin, so we use at most $2L_2$ bins and the worst case is $L_2/2L_2 = 1/2$
- Performance is tight: consider $n$ items with $w_j = 1/2 + \varepsilon$ and $f_j = 1$, $j = 1, \ldots, n$. Optimal solution = $n$ bins, whereas $L_2 = n/2 + 1$
Branching Scheme

Branching based on the decision variables of the compact model:

- At level $j$, assign item $j$ to:
  - any open bin $i$ ($x_{ji} = 1$)
  - a new bin $j$ ($y_j = 1$)

Good result: we can include the performed branching decisions in $L_2$

<table>
<thead>
<tr>
<th>input</th>
<th>$I_1$ = set of items fixed by branching, $I_2$ = set of remaining items, $z_1$ = number of bins opened to pack $I_1$, $r_1$ = total number of units of free space in the $z_1$ bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Lower bound value $L'_2$</td>
</tr>
</tbody>
</table>

Cut each item $j$ of $I_2$ into $w_j$ items $(1, f_j)$ and call $\overline{I}_2$ the new set; Sort items of $\overline{I}_2$ by non-decreasing fragility; Return $L'_2 = z_1 + L_2 \left( \overline{I}_2 \setminus \{ \text{first } r_1 \text{ items of } \overline{I}_2 \} \right)$;
Branching Scheme (continued)

Proposition

$L'_2$ is a valid lower bound on the optimal value of a BBPFO instance with
the aforementioned branching decisions.

Sketch of the proof:

- there is always an optimal solution in which the first $z_1$ bins are
  packed as much as possible
- Packing items of $\bar{I}_2$ in the first $z_1$ bins does not reduce their fragility
  (ordering is important!)
- Packing the first items of $\bar{I}_2$ in the $z_1$ bins dominates other possible
  packings as allows more fragility in the successive bins
A set covering model

- $P$: set of valid patterns $p$ (feasible fillings of a bin)
- $a_{jp}$: 1 if $j$ is in pattern $p$, 0 otherwise (constant)
- $z_p$: 1 if pattern $p$ is used, 0 otherwise (decision variable)

**Master problem**

\[
\min \sum_{p \in P} z_p \\
\sum_{p \in P} a_{jp} z_p \geq 1 \quad j = 1, \ldots, n \\
z_p \in \{0, 1\} \quad \forall p \in P.
\]

**Pricing subproblem**

$\pi_j$: dual variable associated to $j$

Knapsack Problem with Fragile Objects (KPFO):

maximize sum of $\pi_j$, s.t. sum of weights does not exceed fragility
Branch-and-Price Pricing Subproblem

KPFO solution with ILP

\[
\max \sum_{j=1}^{n} \pi_j (\alpha_j + \beta_j)
\]

\[
\sum_{j=1}^{n} \beta_j = 1
\]

\[
\sum_{j=1}^{n} \beta_j (f_j - w_j) - \sum_{j=1}^{n} w_j \alpha_j \geq 0
\]

\[
\alpha_k + \sum_{j=k}^{n} \beta_j \leq 1 \quad k = 1, \ldots, n
\]

\[
\alpha_j \in \{0, 1\} \quad j = 1, \ldots, n
\]

\[
\beta_j \in \{0, 1\} \quad j = 1, \ldots, n
\]

Variables

- \(\beta_j = 1\) if \(j\) is the item of smallest fragility in the bin
- \(\alpha_j = 1\) if \(j\) is in the bin but is not the item of smallest fragility
There are $n$ items hence there are at most $n$ possible fragilities for the bin. The KPFO can be solved by the following algorithm.

```
input  : A KPFO instance
output: $z_{kpfo} = \text{optimal KPFO value}$

$z_{kpfo} = 0$;
for $j = 1$ to $n$ do
    remove items 1, 2, ..., $j - 1$ from the instance;
    pack item $j$ in the bin;
    solve a standard knapsack (KP) with
        bin capacity $= f_j - w_j$
        items set $j + 1, j + 2, \ldots, n$
    let $z_{kp}$ be the KP solution value, $z_{kpfo} = \max\{z_{kpfo}, \pi_j + z_{kp}\}$;
end
return $z_{kpfo}$;
```
KPFO solution with Dynamic Programming

We solve KPFO with dynamic programming by sorting items by non-increasing fragility:

- DP table with $n$ columns and $f_{\text{max}}$ rows
- Column $j$: propagate states of Column $j - 1$ by selecting or not item $j$.
- The optimal solutions are in the “black” states (white ones are dominated, grey ones are redundant), indeed:
  - Let $\phi(i, c)$ be the max KP sol value for the $i$ first items and capacity at most $c$
  - $z_{\text{KPFO}} = \max_{c=1,\ldots,f_{\text{max}}} \phi(\max\{i : f_i \geq c\}, j)$
  - A state is explored at most once, hence we get same complexity of KP: $O(n \cdot f_{\text{max}})$
Branching Scheme

Same Branching as the B&B, at level $j$, assign item $j$ to:
- any open bin $i$ ($x_{ji} = 1$)
- a new bin $j$ ($y_j = 1$)

Good points:
1. we can use $L'_2$ before invoking the solution of the KPFO
2. branching information can be inserted into pricing:
   1. branching on $x_{ji} = 1$ ⇒ creation of an “aggregate” item $\langle j \cup i \rangle$
   2. branching on $y_j = 1$ ⇒
      - additional constraint in the ILP
      - additional call to KP in the KP-based method
      - duplicate column in the DP table (without affecting complexity!)

Bad points:
- linear relaxation may be unaffected by branching
We took 135 BPP instances from set 1 of Scholl et al. (1997) and multiplied them by 49 fragility configurations:

- **Uncorrelated instances**: \( f_j \in [w_{\text{max}} + k_1(k_2 C - w_{\text{max}})/5, k_2 C] \);
- **Weakly correlated instances**: \( f_j \in [k_1 w_j, k_2 C] \);
- **Strongly correlated instances**: \( f_j = k_1 w_j \).
Computational Results

Benchmark Set

Benchmark set

We took 135 BPP instances from set 1 of Scholl et al. (1997) and multiplied them by 49 fragility configurations:

- **Uncorrelated instances**: \( f_j \in [w_{\text{max}} + k_1(k_2 C - w_{\text{max}})/5, k_2 C] \);
- **Weakly correlated instances**: \( f_j \in [k_1 w_j, k_2 C] \);
- **Strongly correlated instances**: \( f_j = k_1 w_j \).

We tested instances with compact model and kept the most difficult ones:

- **Class 1**: uncorrelated with \( k_1 = 1 \) and \( k_2 = 3 \);
- **Class 2**: uncorrelated with \( k_1 = 1 \) and \( k_2 = 5 \);
- **Class 3**: weakly correlated with \( k_1 = 1 \) and \( k_2 = 5 \);
- **Class 4**: weakly correlated with \( k_1 = 3 \) and \( k_2 = 3 \);
- **Class 5**: weakly correlated with \( k_1 = 3 \) and \( k_2 = 4 \).
Settings

665 benchmarks available at www.or.unimore.it/resources.htm

All algorithms coded in C++

CPLEX12 used as LP and ILP solver

Procedure COMBO by Martello et al. (1999) used to solve the KP

Tests run on a Xeon 2 GHz

At the root node we invoke

- Lower bound $L_2$
- A set of greedy algorithms to compute upper bounds
  (First Fit, Best Fit, Worst-Fit, KP-based heuristics, ...)
Best Algorithm for Pricing

Branch-and-price (B&P) run for 10 CPU minutes. Pricing solved through

- ILP: KPFO model ($\alpha_j$, $\beta_j$)
- KP-based: $n$ calls to COMBO
- DP: new dynamic programming

<table>
<thead>
<tr>
<th>$n$</th>
<th>#opt</th>
<th>%gap</th>
<th>sec</th>
<th>#opt</th>
<th>%gap</th>
<th>sec</th>
<th>#opt</th>
<th>%gap</th>
<th>sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>215</td>
<td>0.37%</td>
<td>66.5</td>
<td>225</td>
<td>0.00%</td>
<td>3.3</td>
<td>225</td>
<td>0.00%</td>
<td>2.1</td>
</tr>
<tr>
<td>100</td>
<td>73</td>
<td>3.17%</td>
<td>446.4</td>
<td>175</td>
<td>0.97%</td>
<td>150.7</td>
<td>180</td>
<td>0.88%</td>
<td>141.1</td>
</tr>
<tr>
<td>200</td>
<td>36</td>
<td>3.07%</td>
<td>506.1</td>
<td>71</td>
<td>1.82%</td>
<td>420.8</td>
<td>75</td>
<td>1.78%</td>
<td>409.3</td>
</tr>
<tr>
<td>avg</td>
<td>324</td>
<td>2.20%</td>
<td>339.7</td>
<td>471</td>
<td>0.93%</td>
<td>191.6</td>
<td>480</td>
<td>0.89%</td>
<td>184.2</td>
</tr>
</tbody>
</table>
### ILP vs B&B vs B&P

Following algorithms for 10 CPU minutes:
- Compact model: model based on $y_i$ and $x_{ji}$
- B&B: branch-and-bound with $L'_2$
- B&P: branch-and-price with DP

<table>
<thead>
<tr>
<th>n</th>
<th>Compact model</th>
<th></th>
<th></th>
<th>B&amp;B</th>
<th></th>
<th></th>
<th>B&amp;P</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#opt</td>
<td>%gap</td>
<td>sec</td>
<td>#opt</td>
<td>%gap</td>
<td>sec</td>
<td>#opt</td>
<td>%gap</td>
<td>sec</td>
</tr>
<tr>
<td>50</td>
<td>219</td>
<td>0.24%</td>
<td>32.4</td>
<td>223</td>
<td>0.06%</td>
<td>8.1</td>
<td>225</td>
<td>0.00%</td>
<td>2.1</td>
</tr>
<tr>
<td>100</td>
<td>93</td>
<td>3.58%</td>
<td>423.3</td>
<td>144</td>
<td>1.67%</td>
<td>234.6</td>
<td>180</td>
<td>0.88%</td>
<td>141.1</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>5.44%</td>
<td>599.5</td>
<td>53</td>
<td>2.59%</td>
<td>459.8</td>
<td>75</td>
<td>1.78%</td>
<td>409.3</td>
</tr>
<tr>
<td>avg</td>
<td>314</td>
<td>3.09%</td>
<td>351.7</td>
<td>420</td>
<td>1.44%</td>
<td>234.2</td>
<td>480</td>
<td>0.89%</td>
<td>184.2</td>
</tr>
</tbody>
</table>
More and more B&Ps

Other B&P attempts:

- B&P: branch-and-price with DP
- VNS + B&P: a *Variable Neighborhood Search* metaheuristic for 4 CPU minutes (if \( n > 50 \)), followed by B&P for 10 CPU minutes
- VNS + B&P - 1 hour: VNS for 4 CPU minutes followed by B&P for 1 CPU hour

<table>
<thead>
<tr>
<th>( n )</th>
<th>#opt</th>
<th>%gap</th>
<th>sec</th>
<th>#opt</th>
<th>%gap</th>
<th>sec</th>
<th>#opt</th>
<th>%gap</th>
<th>sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>225</td>
<td>0.00%</td>
<td>2.1</td>
<td>225</td>
<td>0.00%</td>
<td>2.1</td>
<td>225</td>
<td>0.00%</td>
<td>2.1</td>
</tr>
<tr>
<td>100</td>
<td>180</td>
<td>0.88%</td>
<td>141.1</td>
<td>185</td>
<td>0.80%</td>
<td>268.9</td>
<td>200</td>
<td>0.50%</td>
<td>638.1</td>
</tr>
<tr>
<td>200</td>
<td>75</td>
<td>1.78%</td>
<td>409.3</td>
<td>78</td>
<td>1.71%</td>
<td>594.3</td>
<td>83</td>
<td>1.65%</td>
<td>2353.8</td>
</tr>
<tr>
<td>avg</td>
<td>480</td>
<td>0.89%</td>
<td>184.2</td>
<td>488</td>
<td>0.84%</td>
<td>288.4</td>
<td>508</td>
<td>0.72%</td>
<td>998.0</td>
</tr>
</tbody>
</table>
A difficult instance with 100 items
Conclusions

Problem interesting but difficult

- We tested several lower bounds, upper bounds and exact algorithms
- B&B and B&P are much better than CPLEX on compact model
- B&P achieves the best results
Conclusions

Problem interesting but difficult

- We tested several lower bounds, upper bounds and exact algorithms
- B&B and B&P are much better than CPLEX on compact model
- B&P achieves the best results

Some ideas still to be tested

- *Dual feasible functions* we tested never outperformed $L_2$, are there better ones?
- *Dual cuts* not useful at the root, could they help at all nodes?
- VNS not very helpful, could it provide a set on initial columns?
- Better branching schemes?
- ... 
- Instances are available, feel free to give it a try!
THANK YOU