Exact and Heuristic Algorithms for the Cardinality Constrained P|# ≤Ki|Cmax Problem

Mauro Dell’Amico
DISMI, Università di Modena e Reggio Emilia

Manuel Iori, Silvano Martello, Michele Monaci
DEIS, Università di Bologna
Outline

- Brief Review on P||Cmax
- Brief Review on P|#<=K|Cmax
- Description of P|#<=K|Cmax problem
- Lower bounds
- Reduction criteria
- Heuristics
- Local search procedures
- Scatter Search
- Branch & bound
- Preliminary computational results
- Column generation approach
P||Cmax

In the classical $P||C_{max}$ problem we are given:

- $m$ processors $i$
  - Parallel
  - Identical
- $n$ jobs $j$
  - each characterized by an integer processing time $p_j$

Aim: minimize the maximum completion time of a job ($makespan$)

Strongly NP-Hard
P||C_{max}

Strongly correlated to the one-dimensional Bin Packing Problem (BPP: determine the minimum number of bins of capacity $C$ used to pack $n$ items of weight $c_j$)

Given $n$ jobs $j$ of length (weight) $P_j$, $m$ identical processors and a threshold value $L$:
Determine if it exists a P||C_{max} solution of value $z=L$

Determine if it exists a BPP solution with capacity of the bins $C=L$ and value $z=m$
Problem $P||C_{max}$ can be formally stated as:

$$\min z$$

$$\sum_{j=1}^{n} p_j x_{ij} \leq z \quad (i = 1, \ldots, m)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j = 1, \ldots, n)$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, \ldots, m; \quad j = 1, \ldots, n)$$

Where:

- $x_{ij}$ takes value 1 if job $j$ is assigned to proc $i$, 0 otherwise
- $z$ represents the value of the makespan
P||Cmax Literature

Three fields notation:

Surveys:

P|Cmax Literature

Heuristics:


An exact approach:
An exact Approach

Dell’Amico and Martello

- Improved continuous lower bounds
- Combinatorial lower bounds
  - Partitioning of the items into subsets in order to find the minimum number of machines necessary to process them (BPP)
- Heuristics from literature
- New heuristics
- Branch & Bound approach

Results:

\[ m \in (3,15), \ n \in (10,10000), \ P_j \text{ generated randomly in many ways} \]

Majority of instances solved optimally, or usually within 1% gap

Computational times low (generally some seconds)
Problem $P|\#\leq K|C_{\text{max}}$ can be formally stated as:

$$\text{min } z$$

$$\sum_{j=1}^{n} p_j x_{ij} \leq z \quad (i = 1, \ldots, m)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j = 1, \ldots, n)$$

$$\sum_{j=1}^{n} x_{ij} \leq k \quad (i = 1, \ldots, m)$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, \ldots, m; \quad j = 1, \ldots, n)$$

Where:

- $x_{ij}$ takes value 1 if job $j$ is assigned to proc $i$, 0 otherwise
- $z$ represents the value of the makespan
Cardinality Constraint

Each processor can handle at max $k$ jobs

$$\Rightarrow n \leq km$$

The special case arising when

$$n = km$$

is defined as K-Partitioning Problem ($KPP$)

Deeply studied in literature

Strongly NP-hard for a general $k$

For $K=2$ \implies trivially solvable in $O(n \log n)$
P|#\leq K|C_{\text{max}} \text{ Literature}

Lower bounds:


Survey:
PhD Thesis by M. Bruglieri (Politecnico Milan)

Heuristic and exact approach:
Dell’Amico, I., Martello. “Heuristic algorithms and scatter search for the cardinality constrained P||C_{\text{max}} problem”. Submitted to *Journal of Heuristics*. 
Computational results

Dell’Amico, I. and Martello

- Improved continuous lower bounds
- Combinatorial lower bounds
- Heuristics from literature
- New heuristics
- Local search
- Scatter Search approach
- Branch & Bound approach
Computational results

\( n \) in \( \{10 - 400\} \), \( m \) in \( \{3 - 50\} \), \( k \) in \( \{3 - 50\} \)

\( P_j \) randomly created according to different criteria

Around 10000 instances in total
Computational results

*K-partitioning and perfect packing instances*

$m$ in \{8 – 30\}, $k$ in \{3 – 25\} and \(n = mk\)

\(P_j\) randomly created (around 5000 instances in total)
We are given:

- $m$ parallel processors $i$, divided into subsets $m_1, \ldots, m_r$
  - In subset $m_i$ each processor can receive at max $K_i$ jobs
- $n$ jobs $j$
  - each characterized by an integer processing time $p_j$
- Aim: minimize the maximum completion time of a job ($makespan$)

Strongly NP-Hard


When $n$ is equal to the sum of $K_i$: $K_i$-partitioning problem

Assumption: jobs sorted by non-increasing values of $p_j$
  processors sorted by non-decreasing values of $K_i$
\[ P|\# \leq K_i|C_{\text{max}} \]

Problem \( P|\# \leq K_i|C_{\text{max}} \) can be formally stated as:

\[
\begin{align*}
\min & \quad z \\
\sum_{j=1}^{n} p_j x_{ij} & \leq z \quad (i = 1, \ldots, m) \\
\sum_{i=1}^{m} x_{ij} & = 1 \quad (j = 1, \ldots, n) \\
\sum_{j=1}^{n} x_{ij} & \leq k_i \quad (i = 1, \ldots, m) \\
x_{ij} & \in \{0, 1\} \quad (i = 1, \ldots, m; \ j = 1, \ldots, n)
\end{align*}
\]

Where:

- \( x_{ij} \) takes value 1 if job \( j \) is assigned to proc \( i \), 0 otherwise
- \( z \) represents the value of the makespan
Literature

Heuristics:


Scatter Search:


A New Algorithm

- Lower bounds
- Reduction procedure
- Heuristics
  - List heuristics
  - Multi Fit (Subset) strategies
  - Mix strategies
- Local search
- Scatter Search
- Branch & Bound
- Further improvement: column generation approach
Simple Lower Bounds

- Any lower bound for $P||C_{max}$ is valid for the new problem
- Any lower bound for $P|\# \leq K|C_{max}$ with $K \geq K_m$ is valid for the new problem
  - Use the exact approach for $P|\# \leq K_m|C_{max}$
  - Eventually improve the lower bound
  - Check the feasibility of the heuristic solution and eventually update the incumbent solution
  - Good performance for “compact” values of $K_i$
  - Poor performance for “sparse” values of $K_i$
Lower Bounds

- Improved continuous lower bound

\[ L_1 = \left\lfloor \frac{1}{m} \sum_{j=1}^{n} p_j \right\rfloor \]

Continuous lower bound

\[ L_2 = \max \left( L_1; p_1 + \sum_{j=n-K_1+2}^{n} p_j \right) \]

Longest job plus \( K_1 - 1 \) smallest jobs

\[ L_3 = \max \left( L_2; \sum_{j=n-K_m+1}^{n} p_j \right) \]

Sum of the \( K_m \) smallest jobs

- Similar considerations can be done for other cases
Reduction Procedure

It tries to assign the largest jobs to the processors with lowest $K_i$

Set $m_r = \{\text{processor 1}\}$, $n_r = \{\text{first } K_1 \text{ jobs}\}$

Repeat

solve reduced problem($n_r, m_r, k_r$) and get solution $Z_r$

if $Z_r \leq L$

//reduce the problem

assign first $n_r$ jobs to $m_r$ processors
continue algorithm with $n - n_r$ and $m - m_r$ processors

else

set $m_r = m_r + \text{next processor } i$
set $n_r = n_r + \text{next } K_i \text{ jobs}$

endif

Until stopping criterion are met
Reduction Procedure

The procedure can be iterated if a success occurs.

The procedure can be called any time the lower bound L is improved.

The reduced problem is solved by branch and bound approach (see later).

Useful for very different values of $K_i$. 
Heuristics: List Strategies

Order jobs in a given order and assign them to processors according to a certain strategy.

- \textit{LPTki}, derived from \textit{LPT} by Graham

  define \textit{car}(i) and \textit{jobs}(i) as the total charge ad number of jobs of processor \(i\) after having assigned a certain number of jobs (order jobs according to non-increasing values of \(p_j\))

  for each job \(j\) in order
  
  assign job \(j\) to processor with lowest \textit{car}(i) and \textit{jobs}(i)<K_i

It generally gives good results with high \(n/m\) (as shown by probabilistic analysis by Coffman et al.) and cardinality constraint not “tight”.
Heuristics: Multi Fit Strategies

Given a threshold value $L$, try to assign the jobs to the processors according to tailored strategies so as to not exceed $L$

(Using Duality properties between $P||C_{max}$ and $BPP$)

- $MS_{Ki}$, $MS_{2ki}$ and $MS_{3ki}$:
  derived from $MS$ by Dell’Amico and Martello
  (order processors according to increasing values of $K_i$)
  for each processor
    assign subsets of jobs such that:
    - the cardinality is not greater than $k_i$
    - total processing time as close as possible to (without exceeding) $L$
  the residual jobs, if any, are then assigned through $LPT_{ki}$

Assignment is obtained through several tries, best values are kept in memory

Good behavior dependent from lower bound $L$
Heuristics: Mixed Strategies

Alternate List strategies and Multi fit strategies

- **MIXki**
  - Receives in input a feasible solution
  - Assignes the first $n'$ jobs as in the input solution
  - Sorts processors according to decreasing values of $\text{car}(i)$
  - Assignes to each processor a subset of unassigned jobs, such that the resulting total processing time:
    - closest to, without exceeding, $L$ if such subset exists
    - closest to $L$ otherwise
    - Cardinality constrained respected
    - (Done through complete enumeration)
  
  Created for Ki-Partitioning instances but appliable to other instances by adding dummy jobs with $P_j=0$
Heuristics: Mixed Strategies

Alternating List and Multi fit strategies

- $MIX2ki$
  - Receives in input a feasible solution
  - Assignes the first $k'$ jobs as in the input solution to each processor (by taking into account cardinality constraint for each $K_i$)
  - Sorts processors according to decreasing values of $car(i)$
  - Assignes to each processor a subset of unassigned jobs, such that the resulting total processing time:
    - closest to, without exceeding, $L$ if such subset exists
    - closest to $L$ otherwise
    - Cardinality constrained respected
    - (Done through complete enumeration)
Heuristics: Mixed Strategies

Derived from Jimeno, Gutierrez and Mokotoff (2001)

- $F_G H$-Ki

Define a set of limits \{$lim$\}

- $Lim_0 = 0$
- $Lim_1 = L$
- $Lim_2 = p_m$
- ...
- $Lim_{i+1} = f(\alpha, Lim_i)$

When assigning job $j$ (in order), consider $\min\{car(i)\}$

If $\min\{car(i)\} < lim$ then

use $LPTK_i$ rule

else

use McNaughton rule: assign job to processor $i$ s. t. $car(i) + p_j$

maximum without exceeding $L$ (or exceeding it in the minimum way)
Heuristics: Mixed Strategies

- **FGH**-$\infty$-Ki
  - If $\lim=o$ then Mc Naughton rule
  - If $\lim=L$ then $LPTK_i$ rule
  - It can be called with different values of $\infty$ and finds a solution for each value of $\lim$

- **DGH1-Ki, DGH2-Ki**
  - As in the previous algorithm, but with different rules for the definition of subsets $\{\lim\}$
Local Search

- All procedures receive in input a feasible solution, with processors sorted by non-decreasing $car(i)$

- MOVE: for each processor $i$, let $j$ be the largest job currently assigned to $i$, execute the following steps:
  a. find the first processor $h>i$, such that $jobs_i(h)<k_i$ and $car(h)+p_j<car(i)$, move job $j$ from $i$ to $h$
  b. if no such $h$ exists, let $j$ be the next largest job of $i$, if any, and go to a.

As soon as a move is executed, the procedure is re-started, until no further move is possible
Local Search

EXCHANGE: for each processor $i$, let $j$ be the largest job currently assigned to $i$, then execute:

a. find the first processor $h>i$, if any, such that there is a job $q$, currently assigned to $h$, satisfying $p_q < p_j$ and $car(h) - p_q + p_j < car(i)$, and interchange $j$ and $q$

b. if no such $h$ exists, let $j$ be the next largest job of $i$, if any, and go to a.

As soon as an exchange is executed, the procedure is restarted, until no further exchange is possible

RE-OPT: for each processor $i$ s.t. $(L-1) <= car(i) < Z$, execute:

- Remove from the instance the jobs assigned to $i$, set $m=m-1$
- Solve the reduced instance through $LPT_{k_i}$, MOVE and EXCHANGE
- Complete the solution by re-adding the removed processor $i$
Scatter Search

- Create a starting reference set $R$ of feasible solutions
- Improve them through $MIXki$, $MIX2ki$ and local search algorithms
- Associate with each solution a “quality” index (fitness)
- Create a new reference set $R'$:
  - Insert the best solutions (Elite) of $R$ in $R'$
  - Choose a subset $C$ of solutions which guarantees a certain level of quality and of diversity
  - Combine the solutions in $C$ through a tailored strategy, in order to create new solutions
  - Improve them through local search
  - Use diversification techniques to escape from local minima (immigration rate)
- Set $R=R'$ and iterate the process until stopping criteria are met
Scatter Search

- **Fitness** $f(l)$
  \[ f(l) = \frac{z}{z-L} \]

- **Diversity** $d(l)$: diversity of the solution from the actual elite
  \[ d(l) = \sum_{e \in E} |\{j \in \{1, \ldots, n\} : y_{e,j} \neq y_{e,j}\}| \]

- **Combination**
  - Choose a combination subset $C$ (in our case the elite)
  - Define a $m \times n$ matrix $M$ with
  - $M_{ij} =$ sum of $f(q)$ such that $j$ is assigned to $i$ in solution $q$ ($q \in C$)
  - Create new random solution by assigning job $j$ to proc $i$ with probability linked to $M_{ij}$, respecting the cardinality constraint

Good heuristic solutions for the problem with limited computational times
Enumeration Algorithm

- Depth-first Branch & Bound
- Jobs sorted by decreasing $p_j$
- During enumeration
  - processors sorted by decreasing ($K_i$-jobs($i$)), breaking ties by decreasing \(\text{car}(i)\)
  - For each job $j$ in order
    - sort processors
    - for each processor $i$ in order
      - compute dominance criteria
      - compute lower bounds
      - assign job $j$ to processor $i$
Enumeration Algorithm

when assigning the last set $N'$ of jobs:

$m$  

$C'(i), K'(i) = \text{residual capacities of processor } i$

$Z_{h-1}$
Enumeration Algorithm

- Improved continuous lower bound
  
  compute the residual area $A_{res}$ as the sum of $C'(i)$
  
  if a processor $i$ cannot receive any more jobs
  
  (i.e. $K'(i)=0$ or $C'(i)+p_m>z_h-1$)
  
  $A_{res}=A_{res}-C'(i)$
  
  if $A_{res} <$ sum of residual $p_j$ compute backtracking

- $K_i$ lower bounds

  Similar procedure for residual $K_i$ capacity

  Consideration on the minimum number of jobs that must be assigned in future to the current processor
Enumeration Algorithm

- Max flow lower bounds

\[
\max z' = \sum_{i=1}^{m} \sum_{j \in N'} p_j x_{ij} \\
\sum_{j \in N'} p_j x_{ij} \leq C'(i) \quad (i = 1, \ldots, m) \\
\sum_{j \in N'} x_{ij} \leq K'(i) \quad (i = 1, \ldots, m) \\
\sum_{i=1}^{m} x_{ij} \leq 1 \quad (j \in N') \\
x_{ij} \in \{0, 1\} \quad (i = 1, \ldots, m; j \in N')
\]

If \( z' = \text{sum of residual } p_j \), go on in the enumeration, otherwise \( (z_h \text{ cannot be improved}) \) compute backtracking

We consider the Linear relaxation of the model
Enumeration Algorithm

- Efficient implementation:
  
  consider the model restricted to (1), (2), (4) and (5), create the graph:

  ![Graph Diagram]

  Arcs from source to jobs have capacity $p_j$
  Arcs from jobs to processors have capacity $p_j$ and are drawn only if the processor can accept the job (cardinality and processing time)
  Arcs from processor to terminal have capacity $C'(i)$
  If max flow < sum of residual $p_j$ then compute backtracking
Enumeration Algorithm

- Efficient implementation:
  consider the model restricted to (1), (3), (4) and (5), create the graph:

  Arrows from source to jobs have capacity=1
  Arrows from jobs to processors have capacity=1 and are drawn only if the processor can accept the job (cardinality and processing time)
  Arrows from processor to terminal have capacity = $K'(i)$
  If max flow < sum of residual jobs then compute backtracking
Enumeration Algorithm

- **Surrogate relaxation:**

\[
\begin{align*}
\max z' &= \sum_{i=1}^{m} \sum_{j \in N'} p_j x_{ij} \\
\sum_{j \in N'} \tilde{p}_j x_{ij} &\leq \tilde{C}'(i) \quad (i = 1, \ldots, m) \\
\sum_{i=1}^{m} x_{ij} &\leq 1 \quad (j \in N') \\
x_{ij} &\in \{0, 1\} \quad (i = 1, \ldots, m; j \in N')
\end{align*}
\]

Where

\[
\tilde{p}_j = p_j + T
\]

\[
\tilde{C}'(i) = C'(i) + TK'(i)
\]
Enumeration Algorithm

Problem: finding good values of T:

Surrogate Dual

So far we used “relevant” values of T

Subgradient Optimization

Nice ideas?
Preliminary computational Results

We coded the algorithm on C and tested it on different instances

\( n \) in \( \{10 - 500\} \) \( m \) in \( \{3 - 50\} \)

<table>
<thead>
<tr>
<th>Class</th>
<th>Ranges</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P_j ) in ([10,1000]) {*}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( P_j ) in ([200,1000])</td>
<td>Uniform</td>
</tr>
<tr>
<td>3</td>
<td>( P_j ) in ([500,1000])</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( \mu = 25 )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( \mu = 50 )</td>
<td>Exponential</td>
</tr>
<tr>
<td>6</td>
<td>( \mu = 100 )</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>( \mu = 100, \sigma = 33 )</td>
<td>Normal</td>
</tr>
<tr>
<td>8</td>
<td>( \mu = 100, \sigma = 66 )</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>( \mu = 100, \sigma = 100 )</td>
<td></td>
</tr>
</tbody>
</table>
Preliminary computational Results

$K_i$ created according to 10 different subclasses

Define $r = n/m$ (rounded up if not integer)

$K_i$ are distributed uniformly in the following ranges

<table>
<thead>
<tr>
<th>Subclass</th>
<th>Ranges</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[r-1; r]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[r-1; r+1]</td>
<td>$K_i$-partitioning</td>
</tr>
<tr>
<td>3</td>
<td>[r-2; r+2]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[r; r+1]</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[r; r+2]</td>
<td>$P</td>
</tr>
<tr>
<td>6</td>
<td>[r; r+3]</td>
<td></td>
</tr>
</tbody>
</table>
Preliminary computational Results

Define a segment of total length \( L = \alpha n \)
Cut the segment into \( m \) pieces, whose length(>1) produces \( K_i \)

<table>
<thead>
<tr>
<th>Subclass</th>
<th>Ranges</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>( \alpha = 1 )</td>
<td>( K_i )-partitioning</td>
</tr>
<tr>
<td>8</td>
<td>( \alpha = 1.25 )</td>
<td>( P</td>
</tr>
<tr>
<td>9</td>
<td>( \alpha = 1.5 )</td>
<td>( P</td>
</tr>
<tr>
<td>10</td>
<td>( \alpha = 2 )</td>
<td>( P</td>
</tr>
</tbody>
</table>
Preliminary computational Results

For each \((m,n)\) and for each subclass of \(K_i\) 10 instances were randomly created.

<table>
<thead>
<tr>
<th></th>
<th>(m=3)</th>
<th></th>
<th>(m=4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(n)</td>
<td>(N^\circ) opt</td>
<td>% gap</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0,00</td>
<td>0,00</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
<td>0,00</td>
<td>0,00</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>0,00</td>
<td>0,01</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>0,00</td>
<td>0,06</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>0,00</td>
<td>0,74</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>0,00</td>
<td>2,36</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>0,00</td>
<td>2,24</td>
</tr>
</tbody>
</table>

Times in seconds on a Pentium IV 1700 MHz
## Preliminary computational Results

<table>
<thead>
<tr>
<th>m=5</th>
<th>m=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>N° opt</td>
</tr>
<tr>
<td>10</td>
<td>/</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
</tr>
</tbody>
</table>
## Preliminary computational Results

<table>
<thead>
<tr>
<th>n</th>
<th>N° opt</th>
<th>% gap</th>
<th>Time</th>
<th>m=20</th>
<th>N° opt</th>
<th>% gap</th>
<th>Time</th>
<th>m=40</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>10</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>25</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>25</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>50</td>
<td>69</td>
<td>0,26</td>
<td>1,40</td>
<td>50</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>100</td>
<td>75</td>
<td>0,72</td>
<td>2,01</td>
<td>100</td>
<td>65</td>
<td>0,26</td>
<td>29,93</td>
<td>/</td>
</tr>
<tr>
<td>200</td>
<td>85</td>
<td>0,82</td>
<td>2,21</td>
<td>200</td>
<td>67</td>
<td>1,11</td>
<td>20,85</td>
<td>/</td>
</tr>
<tr>
<td>400</td>
<td>99</td>
<td>0,06</td>
<td>2,36</td>
<td>400</td>
<td>79</td>
<td>0,79</td>
<td>11,21</td>
<td>/</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>0,00</td>
<td>1,46</td>
<td>500</td>
<td>100</td>
<td>0,00</td>
<td>5,45</td>
<td>/</td>
</tr>
</tbody>
</table>
### Preliminary computational Results

<table>
<thead>
<tr>
<th>m=50</th>
<th>n</th>
<th>N° opt</th>
<th>% gap</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>25</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>50</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>100</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>200</td>
<td>78</td>
<td>0,49</td>
<td>10,34</td>
<td>/</td>
</tr>
<tr>
<td>400</td>
<td>88</td>
<td>0,28</td>
<td>16,93</td>
<td>/</td>
</tr>
<tr>
<td>500</td>
<td>99</td>
<td>0,03</td>
<td>6,56</td>
<td>/</td>
</tr>
</tbody>
</table>
Further Improvements

- Column Generation Approach
  Let $L$ be the best lower bound and $U$ the incumbent solution

  We try a binary search for values $V$ between $L$ and $U$

  We formulate the problem as a 1-dimensional Bin Packing Problem, with capacity of the bins equal to $V$ and cardinality constraints imposed on each bin

  If for a value $V$ we found a solution $= m$, we update $U=V$
  If we prove that no solution exists we set $L=V$
Further Improvements

- Column Generation Approach

We consider a Set Covering formulation of the problem. Let $S$ be the set of all possible fillings of a bin, our aim is to minimize the number of fillings used. For small instances we can consider all the possible fillings. For bigger instances we start with a limited set of fillings, compute the solution and find out if in the dual some constraints are violated. This is done through “slave” problem Knapsack-01, solved by:
  - Dynamic Programming
  - MIPOPT
  - Branch & Bound by Martello and Toth