Weight Annealing Heuristics for Solving the Two-Dimensional Bin Packing Problem

Kok-Hua Loh, Nanyang Technological University
Bruce Golden, University of Maryland
Edward Wasil, American University

11th ICS Conference
Charleston, Jan 2009
Outline of Presentation

- Introduction
- Concept of Weight Annealing
- One-Dimensional Bin Packing Problems
  - Classic Bin Packing
- Two-Dimensional Bin Packing Problems
- Computational Results
- Conclusions
Weight Annealing Concept

- Assigning different weights to different parts of a combinatorial problem to guide computational effort to poorly solved regions
  - Ninio and Schneider (2005)

- Allowing both uphill and downhill moves to escape from a poor local optimum

- Tracking changes in objective function value and how well every region is being solved

- Applied to the Traveling Salesman Problem (Ninio and Schneider 2005)
  - Weight annealing led to mostly better results than simulated annealing
Solving the TSP with Weight Annealing

(1) Local Optimum
(visit sequence: 1-2-3-4-5-6-7-1)

(2) New TSP Configuration

(3) New Local Optimum
(visit sequence: 1-2-4-5-6-3-7-1)

Local Optimum
(original space)
One-Dimensional Bin Packing Problem

A classic combinatorial optimization problem

- Pack a set of $N = \{1, 2, \ldots, n\}$ items, each with size $t_i$, $i=1, 2, \ldots, n$, into identical bins, each with capacity $C$

- Minimize the number of bins without violating the capacity constraints

- Large literature on solving this NP-hard problem

Item List =\{8,7,7,6,6,5,5,4,4,3,3\}  Bin Capacity =15

Bin 1  
\[
\begin{array}{c}
\text{7} \\
\text{8} \\
\end{array}
\]

Bin 2  
\[
\begin{array}{c}
\text{6} \\
\text{7} \\
\end{array}
\]

Bin 3  
\[
\begin{array}{c}
\text{4} \\
\text{5} \\
\end{array}
\]

Bin 4  
\[
\begin{array}{c}
\text{3} \\
\text{3} \\
\end{array}
\]

\[
\begin{array}{c}
\text{4} \\
\text{5} \\
\end{array}
\]

\[
\begin{array}{c}
\text{3} \\
\text{3} \\
\end{array}
\]

\[
\begin{array}{c}
\text{4} \\
\text{5} \\
\end{array}
\]
Outline of Weight Annealing Algorithm

- Construct an initial solution using first-fit decreasing

- Compute and assign weights to items to distort sizes according to the packing solutions of individual bins

- Perform local search by swapping items between all pairs of bins

- Carry out re-weighting based on the result of the previous optimization run

- Reduce weight distortion according to a cooling schedule
Neighborhood Search for Bin Packing Problem

- From a current solution, obtain the next solution by swapping items between bins with the following objective function (suggested by Fleszar and Hindi 2002)

\[
\text{Maximize } f = \sum_{i=1}^{p} (l_i)^2 \\
\quad l_i = \sum_{j=1}^{q_i} t_j \quad \text{bin } i \text{ load} \\
p = \text{number of bins} \\
q_i = \text{number of items in bin } i \\
t_j = \text{size of item } j
\]

\[
f = (5 + 3)^2 + 2^2 = 68 \\
f_{\text{new}} = (5 + 3 + 2)^2 = 100
\]
Neighborhood Search for Bin Packing Problem

- **Swap (1,1)**

  ![Swap (1,1) Diagram](image)

  - Initial Configuration: $f = 162$

  - After Swap: $f_{new} = 164$

- **Swap (1,2)**

  ![Swap (1,2) Diagram](image)

  - Initial Configuration: $f = 162$

  - After Swap: $f_{new} = 164$
Weight Annealing for Bin Packing Problem

- Weight of bin $i$
  \[ \omega_i = 1 + K r_i \]

  residual capacity  \[ r_i = \left( \frac{C - l_i}{C} \right) \]

  $C = \text{capacity}$  
  $l_i = \text{load of bin } i$

- An item in a not-so-well-packed bin, with large $r_i$, will have its size distorted by a large amount

- No size distortions for items in fully packed bins

- $K$ controls the size distortion, given a fixed $r_i$
Weight annealing allows downhill moves in a maximization problem.

Example \( C = 200, \ K = 0.5, \ \omega_i = 1 + 0.5 \left( \frac{200 - l_i}{200} \right) \)

- Transformed space - uphill move
- Original space - downhill move
Weight Annealing for Bin Packing Problem

- **Step 0. Initialization**
  Parameters are $K$ (scaling parameter), $nloop$, $T$ (temperature), and $Tred$.
  Inputs are number of items ($n$), item sizes ($t_j$), bin capacity ($C$), and lower bound ($LB$).
- **Step 1.** Construct an initial solution using first-fit decreasing procedure.
  Compute residual capacity $r_i$.
- **Step 2.** Improve the current solution.
  For $i = 1$ to $nloop$
    Compute the weights $w_i^T = (1 + Kr_i)^T$.
    Do for all pairs of bins
    Perform Swap (1,0){
      Evaluate feasibility and $\Delta f(1,0)$.
      If $\Delta f(1,0) \geq 0$
        Move the item.
        Exit Swap(1,0) and,
        Exit $i$ loop if $LB$ is reached.
        Exit Swap (1,0) if no feasible move with $\Delta f(1,0) \geq 0$ is found.
    }
    Perform Swap (1,1)
    Perform Swap (1,2)
    Perform Swap (2,2)

  $T := T \times Tred$
  End of $i$ loop
- **Step 3.** Output the results.
  Outputs are the number of bins used, the final distribution of items, and $r_i$. 
Two-Dimensional Bin Packing Problems

Problem statement

- Allocate, without overlapping, $n$ rectangular items to identical rectangular bins
- Pack items such that the edges of bins and items are parallel to each other
- Minimize the total number of rectangular bins (NP-hard)

Classifications

- Guillotine cutting
  - 2BPO|G \quad Fixed Orientation (O), Guillotine Cutting (G)
  - 2BP|R|G \quad Allowable 90° Rotation (R), Guillotine Cutting (G)

- Free cutting
  - 2BPO|F \quad Fixed Orientation (O), Free Cutting (F)
  - 2BP|R|F \quad Allowable 90° Rotation (R), Free Cutting (F)
Hybrid first-fit algorithm

- Phase One (one-dimensional horizontal level packing)
  - Arrange the items in the order of non-increasing height
  - Pack the items from left to right into levels; each level $i$ with a maximum width $W$
  - Pack an item (left justified) on the first level that can accommodate it; initiate a new level if no level can accommodate it

- Phase Two (one-dimensional vertical bin packing)
  - Arrange the levels in the order of non-increasing height $h_i$; this is the height of the first item on the left
  - Solve one-dimensional bin packing problems, each item $i$ with size $h_i$ and bin size $H$
Two-Dimensional Bin Packing Problems (2BP|O|G)

- Example:
  - Arrange items in the order of non-increasing height

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
```
Two-Dimensional Bin Packing Problems (2BP|O|G)

An example of hybrid first-fit
Two-Dimensional Bin Packing Problems (2BP|O|G)

Hybrid first-fit algorithm

- Phase One (one-dimensional horizontal level packing)
  - Arrange the items in the order of non-increasing height
  - Pack the items from left to right into levels; each level $i$ with the same width $W$
  - Pack an item (left justified) on the first level that can accommodate it; initiate a new level if no level can accommodate it

- Phase Two (one-dimensional vertical bin packing)
  - Solve one-dimensional bin packing problems, each item $i$ with size $h_i$ and bin size $H$
Two-Dimensional Bin Packing Problems (2BP|O|G)

An example of hybrid first-fit

Phase 1 - One Dimensional Horizontal Level Packing

Phase 2 - One Dimensional Vertical Bin Packing
Two-Dimensional Bin Packing Problems (2BP|O|G)

Weakness of hybrid first-fit

Unoccupied space at each level

Unoccupied space at the top end of each bin
Weight Annealing Algorithm (2BP|O|G)

- Phase One (one-dimensional horizontal level packing)

  - Construct an initial solution
    - Arrange the items in the order of non-increasing height
  - Swap items between levels to minimize the number of levels
    - Objective function

\[
\text{Maximize } f = \sum_{i=1}^{p} (b_i)^2 - \sum_{i=1}^{p} (Wh_i - A_i)
\]

\[
b_i = \sum_{j=1}^{m_i} w_{ij}
\]

\[
w_{ij} = \text{width of item } j \text{ in level } i
\]

\[
m_i = \text{number of items in level } i
\]

\[
p = \text{number of levels}
\]

\[
A_i = \text{sum of item areas in level } i
\]

\[
h_i = \text{height of level } i
\]

\[
W = \text{bin width}
\]
Weight Annealing Algorithm (2BP|O|G)

Example

Maximize \( f = \sum_{i=1}^{n} (b_i)^2 \)

\[ b_1^2 = (5 + 3)^2 = 64 \]
\[ b_2^2 = 2^2 = 4 \]
\[ \Rightarrow f = b_1^2 + b_2^2 = 68 \] (two levels)

\[ f = (5 + 3 + 2)^2 = 100 \] (one level)
Weight Annealing Algorithm (2BP|O|G)

Example

Maximize \( f = \sum_{i=1}^{p} (b_i)^2 - \sum_{i=1}^{p} (W_i - A_i) \)

![Diagram showing the weight annealing algorithm process](image)

- Level 1:
  - Unused area: 10 + 10 = 20, \( h_1 + h_2 = 6 \)

- Level 2:
  - Unused area: 0, \( h_1 + h_2 = 4 \)
Weight Annealing Algorithm (2BP|O|G)

- Phase Two (one-dimensional vertical bin packing)
  - Construct initial solution with first-fit decreasing using level height $h_i$ as item sizes and bin height $H$
  - Swap levels between bins to minimize the number of bins
    - Objective function
      \[
      \text{Maximize } f = \sum_{i=1}^{q} (d_i)^2
      \]
      \[
      d_i = \sum_{j=1}^{m_i} h_{ij}
      \]
      \[
      h_{ij} = \text{height of level } j \text{ in bin } i
      \]
      \[
      m_i = \text{number of levels in bin } i
      \]
      \[
      q = \text{number of bins}
      \]
**Weight Annealing Algorithm (2BP|O|G)**

- **Phase Three**
  - Filling unused space in each level

Maximize \[ f = \sum_{i=1}^{q} (A_i)^2 \]
Phase Three

- Filling unused space at the top of each bin

Maximize \( f = \sum_{i=1}^{q} (A_i)^2 \)
Weight Annealing Algorithm (2BP|O|G)

- Weight assignments

- **Phase One**
  \[ \omega_i = 1 + Kr_i \]
  \[ r_i = \left( \frac{W - b_i}{W} \right) \]

- **Phase Two**
  \[ \omega_i = 1 + Kr_i \]
  \[ r_i = \left( \frac{H - d_i}{H} \right) \]

- **Phase Three**
  \[ \omega_i = 1 + Kr_i \]
  \[ r_i = \left( \frac{HW - A_i}{HW} \right) \]
Solution Procedures

- Tabu search by Lodi, Martello and Toth (1999) for 2BP|O|G, 2BP|R|G, 2BP|O|F, 2BP|R|F

- Guided local search by Faroe, Pisinger and Zachariasen (2003) for 2BP|O|F

- Exact algorithm by Martello and Vigo (1998) for 2BP|O|F

- Constructive algorithm (HBP) by Boschetti and Mingozzi (2003) for 2BP|O|F

- Set covering heuristics by Monaci and Toth (2006) for 2BP|O|F

- 500 test instances in all for 2BP|O|F
## Computational Results for 2BP|O|F

<table>
<thead>
<tr>
<th>Procedures</th>
<th>Number of bins</th>
<th>Running Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabu Search</td>
<td>7364</td>
<td>1436.1</td>
</tr>
<tr>
<td>Guided Local Search</td>
<td>7302</td>
<td>-</td>
</tr>
<tr>
<td>Exact Algorithm†</td>
<td>7313</td>
<td>524.7</td>
</tr>
<tr>
<td>Constructive Algorithm (HBP)</td>
<td>7265</td>
<td>345.9</td>
</tr>
<tr>
<td>Set Covering Heuristics</td>
<td>7248</td>
<td>148.5</td>
</tr>
<tr>
<td>Weight Annealing</td>
<td>7253</td>
<td>119.3</td>
</tr>
</tbody>
</table>

Best Lower Bound (LB): 7173 bins
Weight Annealing is 1.1% above LB
†Timed out on several problems
Conclusions

▪ The application of weight annealing to bin packing and knapsack problems is new

▪ Weight annealing algorithms produce high-quality solutions

▪ Weight annealing algorithms are fast and competitive
  ➢ Easy to understand
  ➢ Simple to code
  ➢ Small number of parameters