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

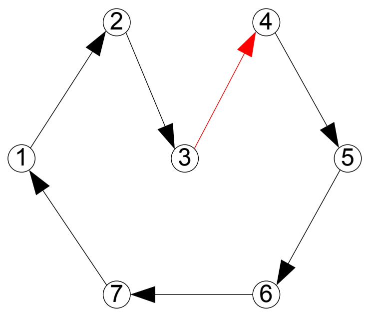
Outline of Presentation

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

Weight Annealing Concept

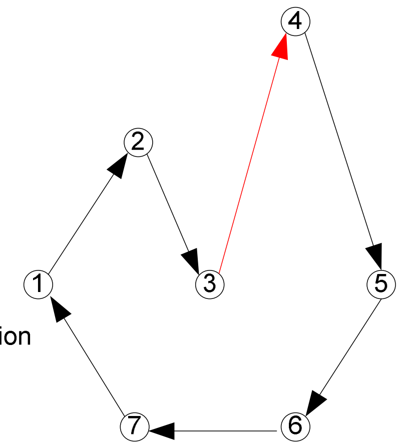
- Assigning different weights to different parts of a combinatorial problem to guide computational effort to poorly solved regions
  - Ninio and Schneider (2005)
- Allowing both uphill and downhill moves to escape from a poor local optimum
- Tracking changes in objective function value and how well every 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



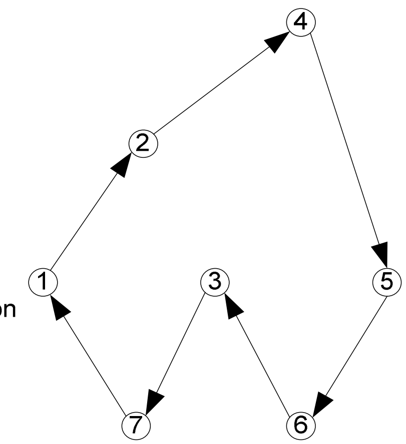
(1) Local Optimum
(visit sequence:1-2-3-4-5-6-7-1)

edge
transformation

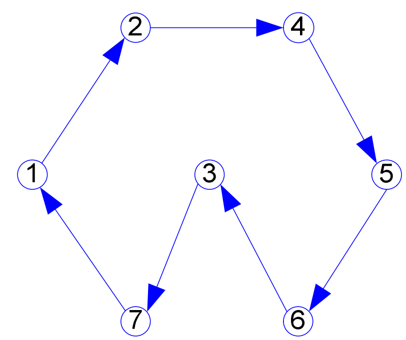


(2) New TSP Configuration

re-optimization



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



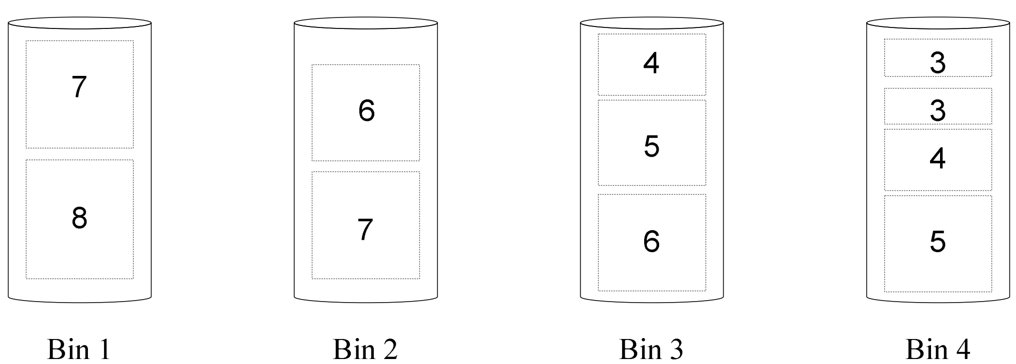
Local Optimum
(original space)

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,…, 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



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)

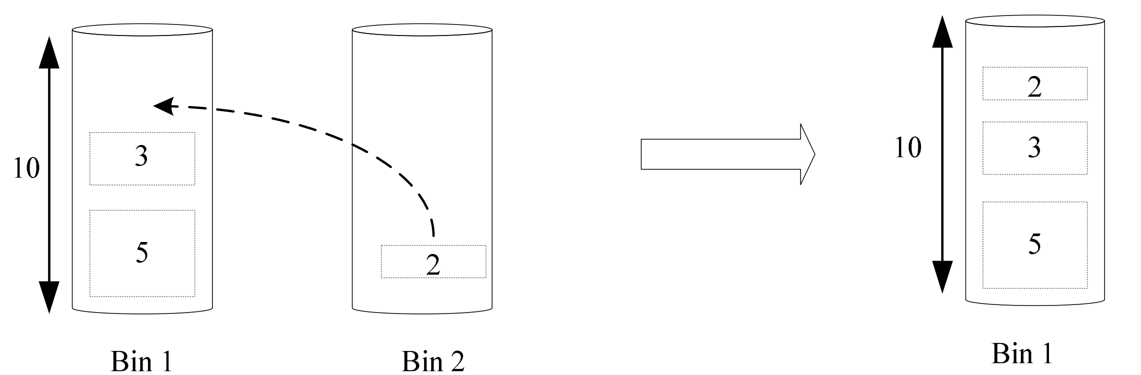
$$\text{Maximize} \quad f = \sum_{i=1}^{p} (l_i)^2$$

$l_i = \sum_{j=1}^{q_i} t_j$ bin i load

p = number of bins

q_i = number of items in bin i

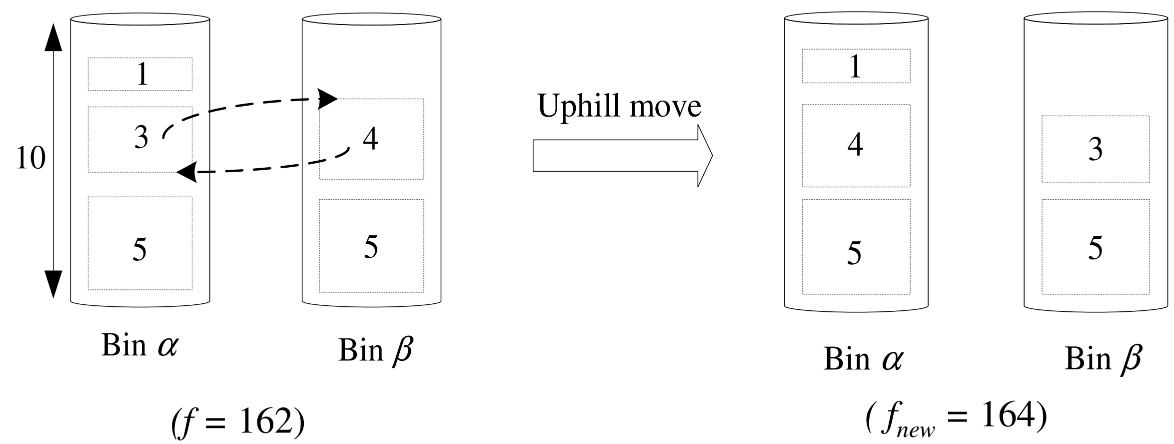
t_j = size of item j



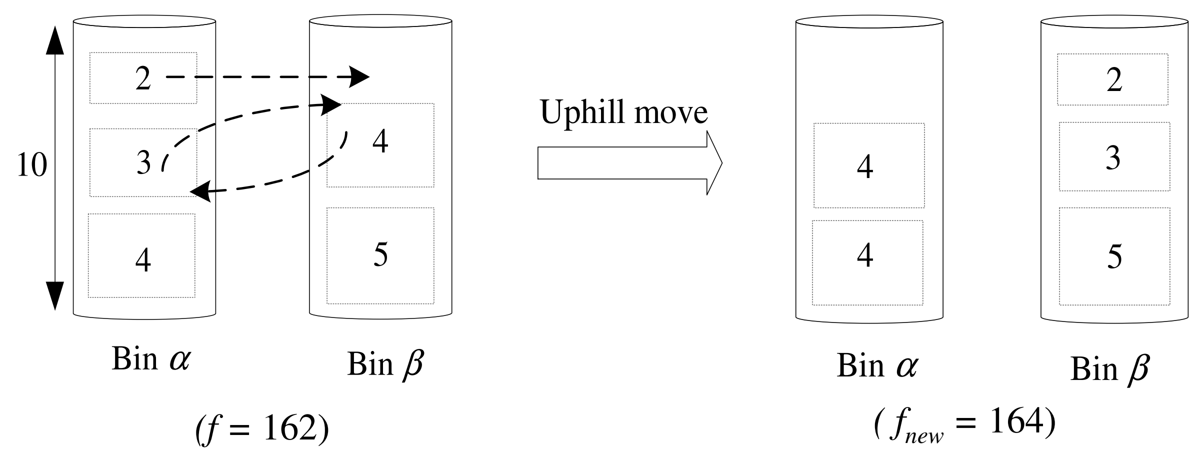
$$f = (5+3)^2 + 2^2 = 68 \qquad f_{new} = (5+3+2)^2 = 100$$

Neighborhood Search for Bin Packing Problem

- Swap (1,1)



- Swap (1,2)



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 = capacity

l_i = 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

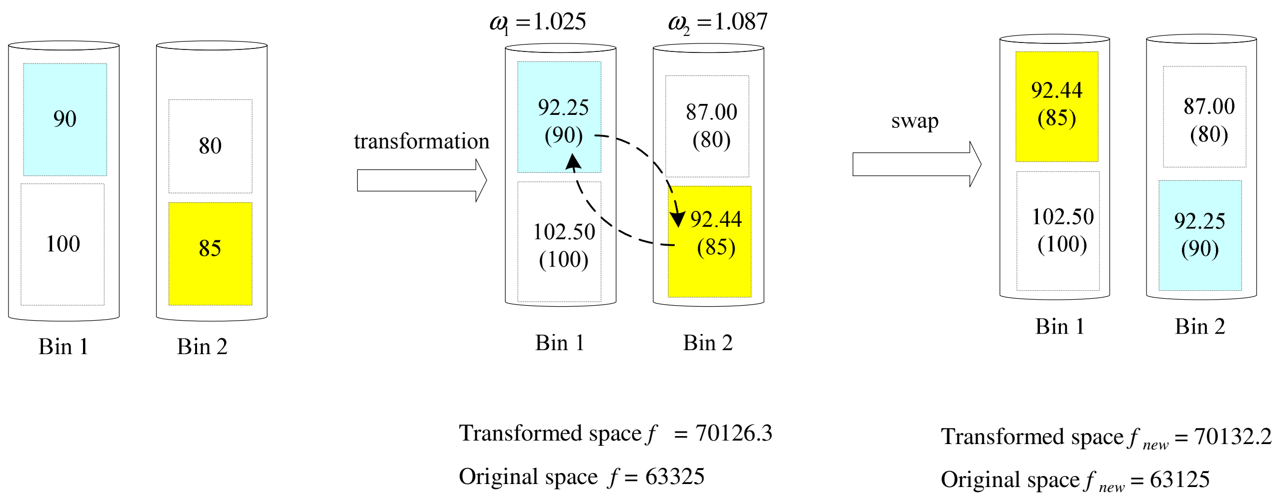
- No size distortions for items in fully packed bins

- K controls the size distortion, given a fixed r_i

Weight Annealing for Bin Packing Problem

- 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
  - 2BP|O|G Fixed Orientation (O), Guillotine Cutting (G)
  - 2BP|R|G Allowable 90° Rotation (R), Guillotine Cutting (G)
- Free cutting
  - 2BP|O|F Fixed Orientation (O), Free Cutting (F)
  - 2BP|R|F Allowable 90° Rotation (R), Free Cutting (F)

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

<u>Hybrid first-fit algorithm</u>

- Phase One (one-dimensional horizontal level packing)

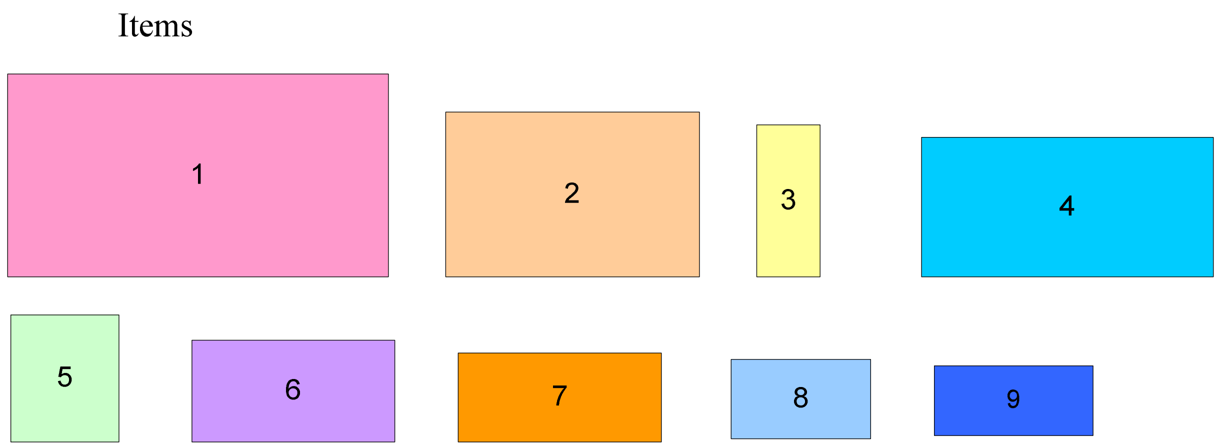
  - Arrange the items in the order of non-increasing height
  - Pack the items from left to right into levels; each level *i* with a maximum width *W*
  - Pack an item (left justified) on the first level that can accommodate it; initiate a new level if no level can accommodate it

- 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*

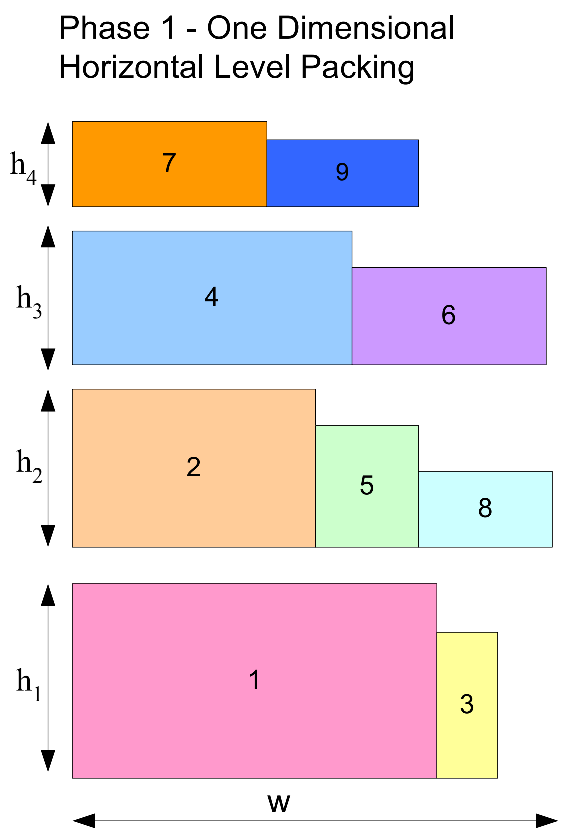
Two-Dimensional Bin Packing Problems (2BP|O|G)

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



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

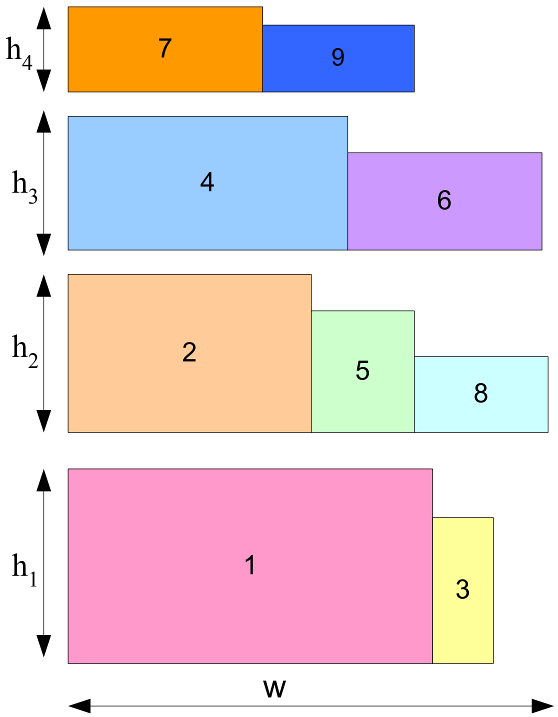
- Phase One (one-dimensional horizontal level packing)
  - Arrange the items in the order of non-increasing height
  - Pack the items from left to right into levels; each level i with the same width W
  - Pack an item (left justified) on the first level that can accommodate it; initiate a new level if 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

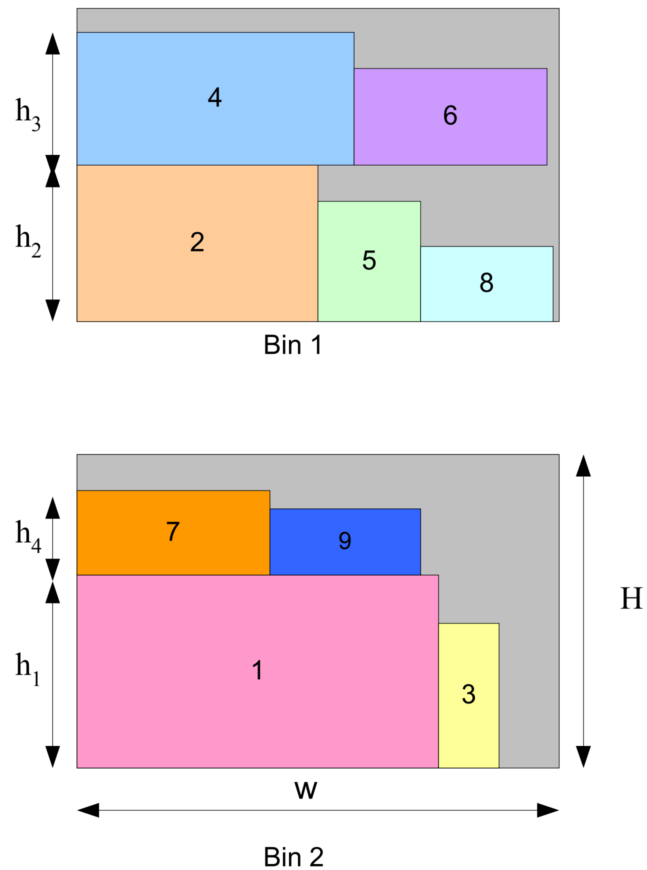
Two-Dimensional Bin Packing Problems (2BP|O|G)

An example of hybrid first-fit

Phase 1 - One Dimensional Horizontal Level Packing

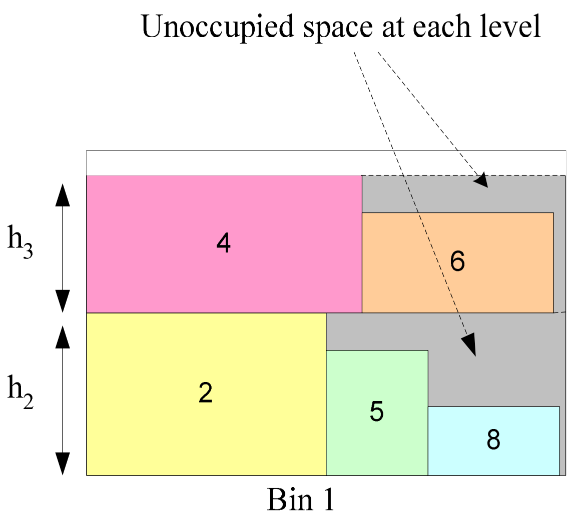


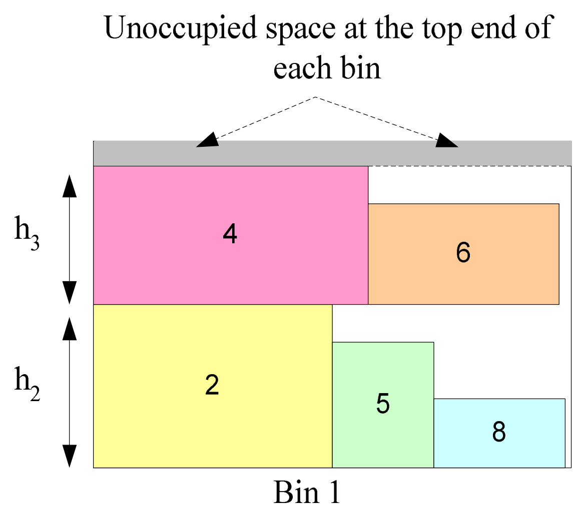
Phase 2 - One Dimensional Vertical Bin Packing



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

Weakness of hybrid first-fit





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

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

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

w_{ij} = width of item j in level i

m_i = number of items in level i

p = number of levels

A_i = sum of item areas in level i

h_i = height of level i

W = bin width

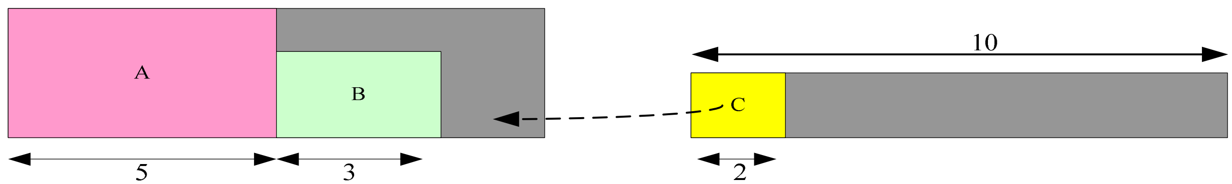
Weight Annealing Algorithm(2BP|O|G)

- Example

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

Level 1

Level 2



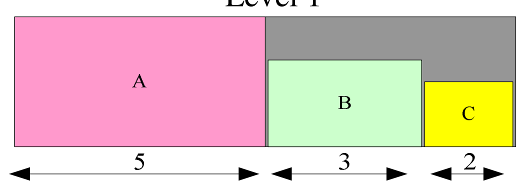
$b_1{}^2=(5+3)^2=64$

$b_2{}^2=2^2=4$

$\Rightarrow f=b_1^2+b_2^2=68$

(two levels)

Level 1



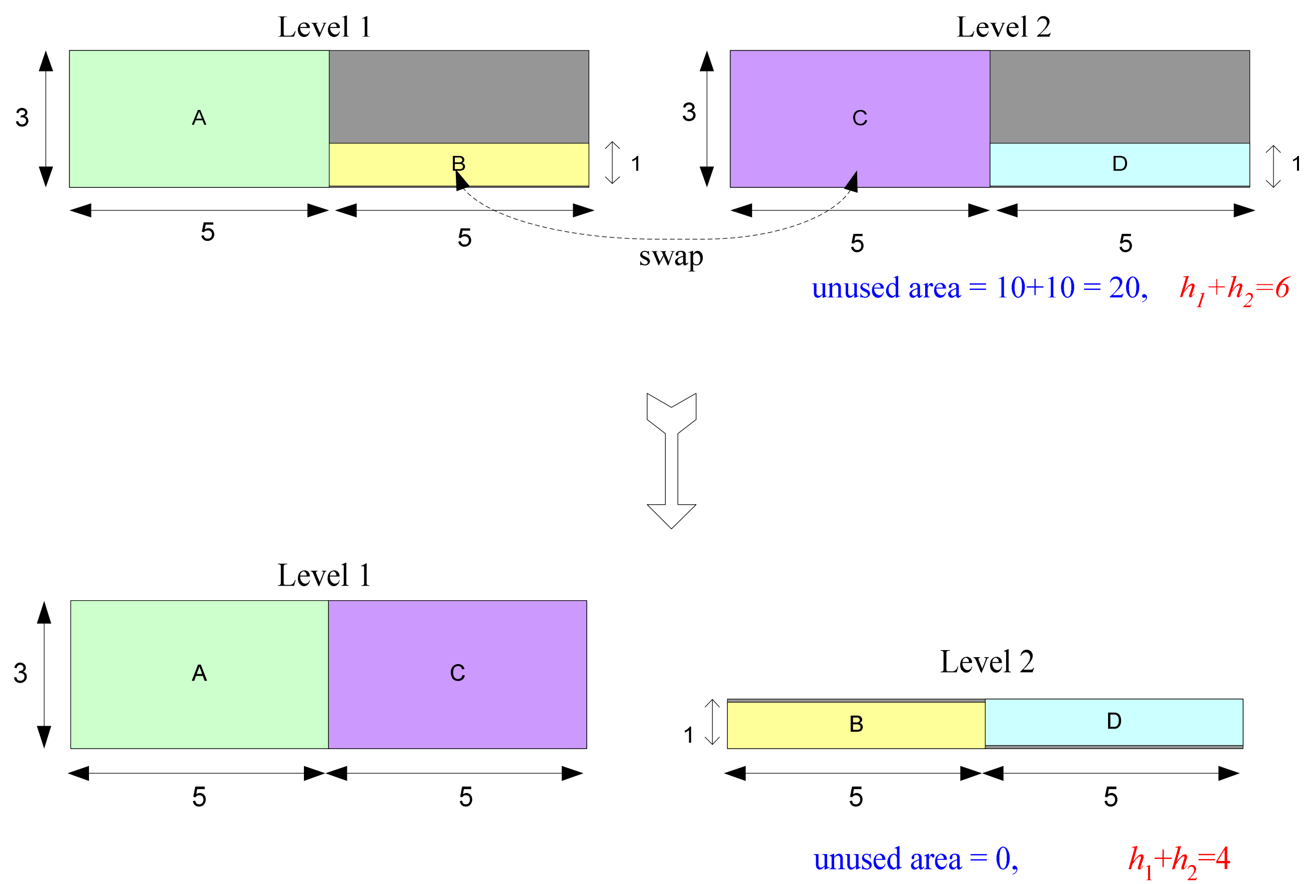
$f=(5+3+2)^2=100$

(one level)

Weight Annealing Algorithm(2BP|O|G)

- Example

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



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

$$\text{Maximize} \quad f = \sum_{i=1}^{q} (d_i)^2$$

$$d_i = \sum_{j=1}^{m_i} h_{ij}$$

h_{ij} = height of level j in bin i

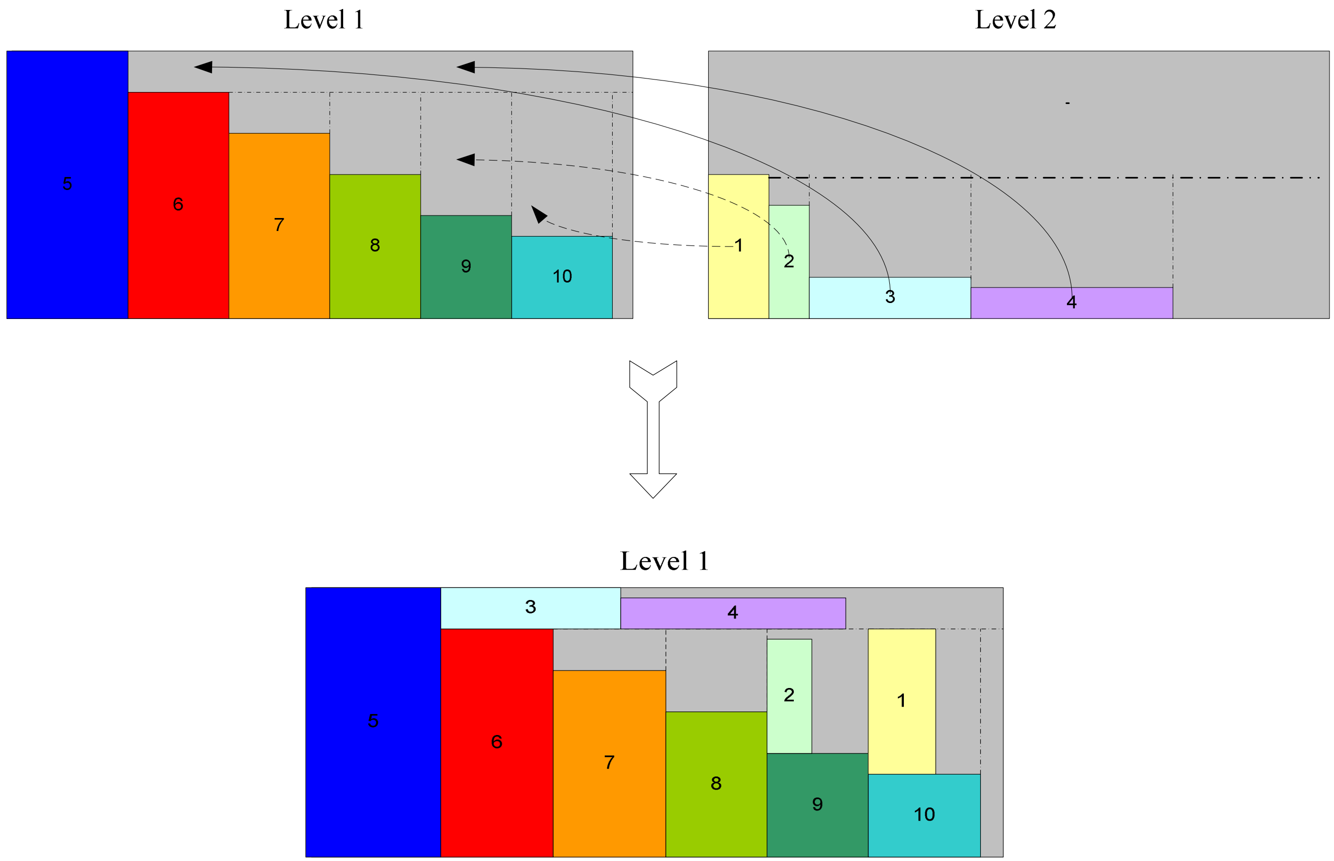
m_i = number of levels in bin i

q = number of bins

Weight Annealing Algorithm(2BP|O|G)

- Phase Three
  - Filling unused space in each level

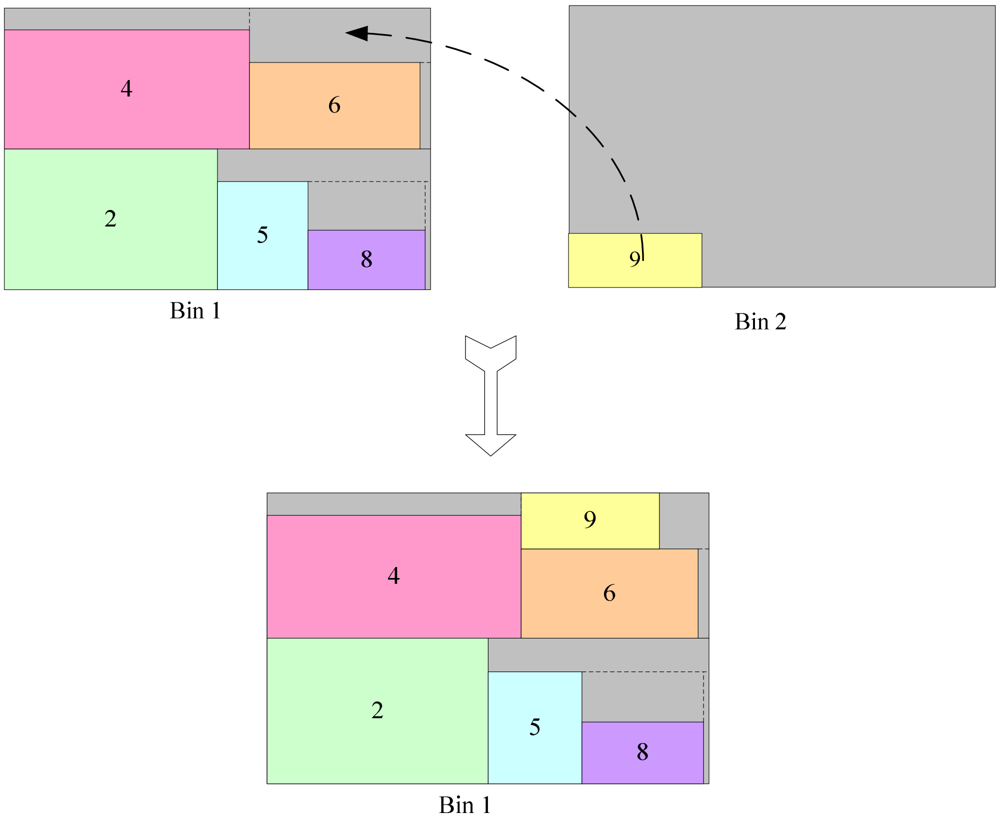
Maximize $f = \sum_{i=1}^{q} (A_i)^2$



Weight Annealing Algorithm(2BP|O|G)

- 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 $\omega_i = 1 + Kr_i$ $r_i = \left(\dfrac{W - b_i}{W}\right)$
  - Phase Two $\omega_i = 1 + Kr_i$ $r_i = \left(\dfrac{H - d_i}{H}\right)$
  - Phase Three $\omega_i = 1 + Kr_i$ $r_i = \left(\dfrac{HW - A_i}{HW}\right)$

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

Procedures	Number of bins	Running Time (sec)
Tabu Search	7364	1436.1
Guided Local Search	7302	-
Exact Algorithm†	7313	524.7
Constructive Algorithm (HBP)	7265	345.9
Set Covering Heuristics	7248	148.5
Weight Annealing	7253	119.3

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