Solving the Maximum Cardinality Bin Packing Problem with a Weight Annealing-Based Algorithm

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

11th INFORMS Computing Society Conference
January 2009

Outline of Presentation

- Maximum Cardinality Bin Packing Problem (MCBPP)
- Weight Annealing
- Solving the MCBPP with Weight Annealing
- Computational Results
- Conclusions

Maximum Cardinality Bin Packing Problem

- Problem statement
  - Assign a subset of n items with sizes t_i to a fixed number of m bins of identical capacity c
  - Maximize the number of items assigned

- Formulation

$$\text{maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{n} t_i x_{ij} \le c \qquad j \in \{1,\ldots,m\}$$

$$\sum_{j=1}^{m} x_{ij} \le 1 \qquad i \in \{1,\ldots,n\}$$

$$x_{ij} = 0 \text{ or } 1 \qquad i \in \{1,\ldots,n\}, \quad j \in \{1,\ldots,m\}$$

where $x_{ij} = 1$ if item i is assigned to bin j and 0 otherwise

Maximal Cardinality Bin Packing Problem

- Practical applications
  - Computing (Labbé, Laporte, and Martello 2003)
    - Assign variable-length records to a fixed amount of storage
    - Maximize the number of records stored in fast memory to ensure a minimum access time to the records
  - Managing real-time multi-processors (Coffman, Leung, and Ting 1978)
    - Maximize the number of completed tasks with varying job durations before a given deadline
  - Computer design (Ferreira, Martin, and Weismantel 1996)
    - Designing processors for mainframe computers
    - Layout of electronic circuits

Weight Annealing

- Allow a greedy heuristic to escape from a poor local optimum by changing the problem landscape and making use of the history of each optimization run
  - Ninio and Schneider (2005)
  - Elidan et al. (2002)
- Assign different weights to different parts of the solution space to guide the computational effort to poorly solved regions
- Allow both uphill and downhill moves
- Track changes in objective function value, as well as how well every region is being solved

Weight Annealing

- Ninio and Schneider (2005) provide an outline of weight annealing

  Step 1. Start with an initial configuration from a greedy heuristic solution using the original problem landscape

  Step 2. Determine a new set of weights based on the previous optimization run and insight into the problem

  Step 3. Perform a new run of the greedy heuristic using the new weights

  Step 4. Return to Step 2 until a stopping criterion is met

- Ninio and Schneider (2005) applied weight annealing to the TSP
  - Five problems with 127 to 1379 nodes
  - Results competitive with simulated annealing

One-Dimensional Bin Packing Problem (1BP)

- 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

  Loh, Golden, and Wasil (2008)

Outline of Weight Annealing Algorithm

- Construct an initial solution using first-fit decreasing procedure
- 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 (Fleszar and Hindi 2002)

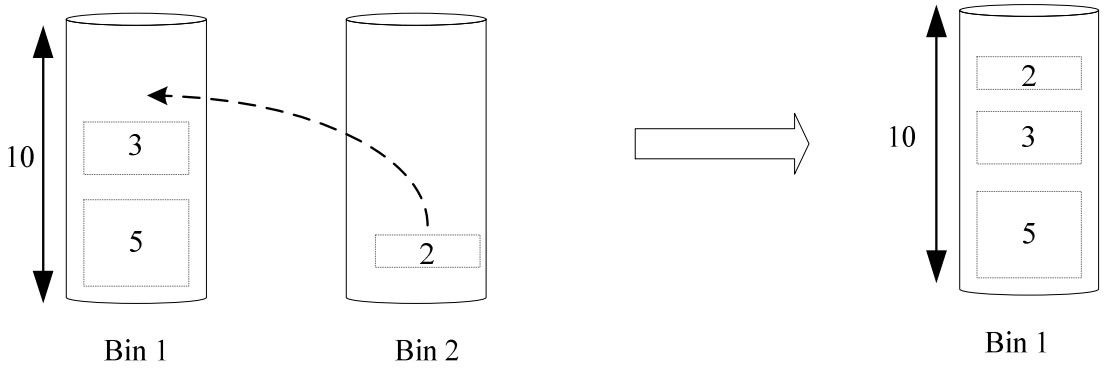
$$\text{Maximize} \quad f = \sum_{i=1}^{p} (l_i)^2$$

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

p = number of bins

q_i = number of items in bin i

t_j = size of item j



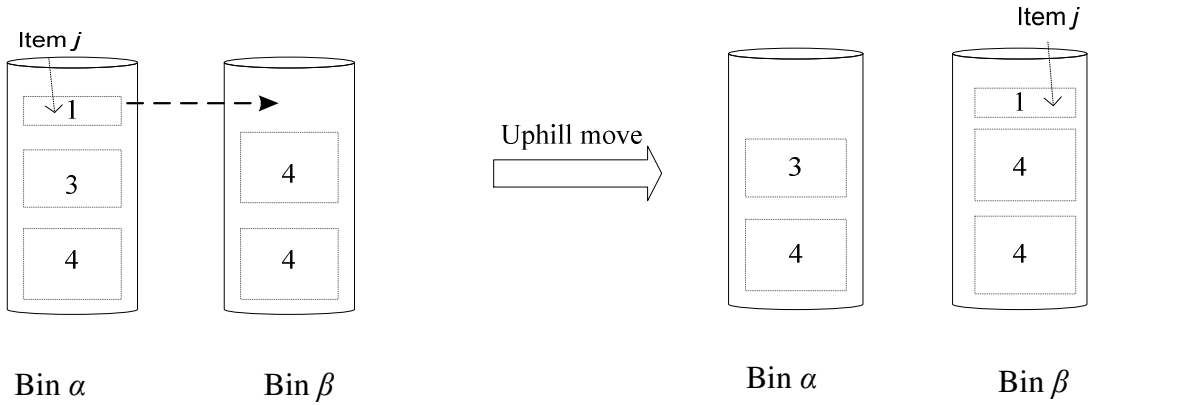
$$f = (5+3)^2 + 2^2 = 68$$

$$f_{new} = (5+3+2)^2 = 100$$

Neighborhood Search for Bin Packing Problem

- Swap schemes
  - Swap items between two bins
  - Carry out Swap (1,0), Swap (1,1), Swap (1,2), Swap (2,2) for all pairs of bins

- Swap (1,0) (Fleszar and Hindi 2002)



$$f = (1+3+4)^2 + (4+4)^2 = 128$$

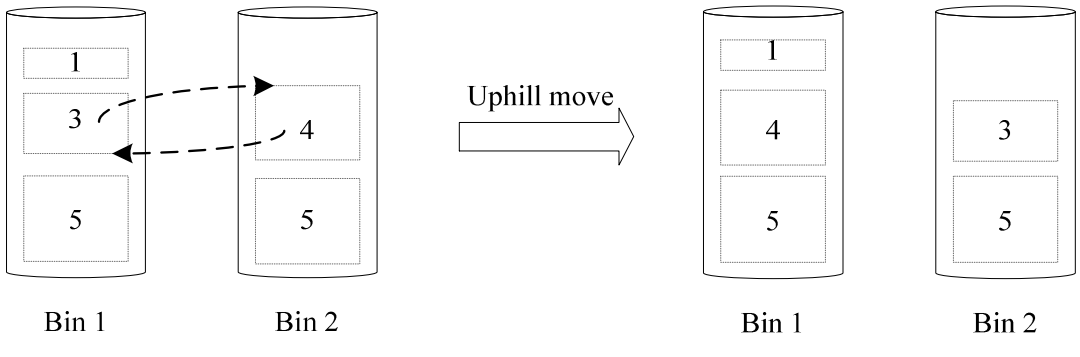
$$f_{new} = (3+4)^2 + (1+4+4)^2 = 130$$

- Need to evaluate only the change in the objective function value

$$\Delta f = (l_\alpha - t_j)^2 + (l_\beta + t_j)^2 - l_\alpha^2 - l_\beta^2$$

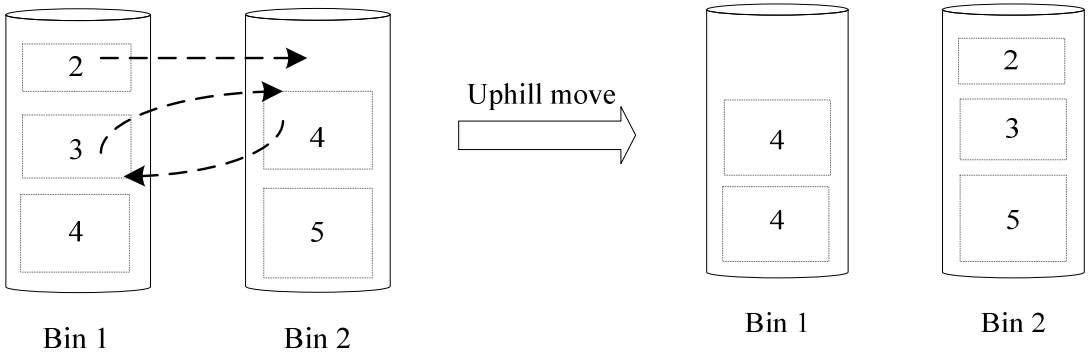
Neighborhood Search for Bin Packing Problem

- Swap (1,1)



(f = 162) (f_{new} = 164)

- Swap (1,2)



(f = 162) (f_{new} = 164)

Weight Annealing for Bin Packing Problem

- Weight of item i

$$w_i = 1 + K\, r_i$$

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

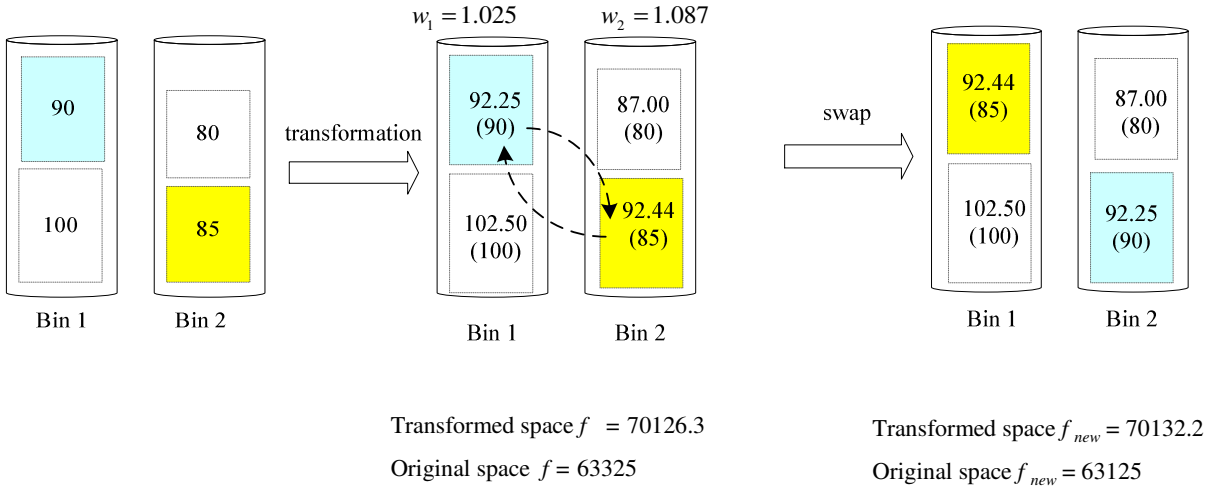
C = capacity

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



Transformed space f = 70126.3

Original space f = 63325

Transformed space f_{new} = 70132.2

Original space f_{new} = 63125

- Transformed space - uphill move
- Original space - downhill move

Bounds for MCBPP

- We use the three upper bounds on the optimal number of items developed by Labbé, Laporte, and Martello (2003): $\overline{U}_0$, $\overline{U}_1$, and $\overline{U}_2$.

- We use the two lower bounds on the minimal number of bins developed by Martello and Toth (1990): L_2 and L_3.

Outline of Weight Annealing Algorithm (WAMC)

- Arrange items in the order of non-decreasing size
- Compute a priori upper bound on the optimal number of items (U^*).
  - $U^* = \min\{\overline{U}_0, \overline{U}_1, \overline{U}_2\}$.
  - Update ordered list by removing item i with size t_i for which $i > U^*$

    (Optimal solution z^* is obtained by selecting the first z^* smallest items)
- Improve the upper bound U^*
  - Find lower bound on the minimum number of bins required (L_3)
  - If $L_3 > m$, reduce U^* by 1.
  - Update ordered list by removing item i with size t_i for which $i > U^*$
  - Iterate until $L_3 = m$
- Find feasible packing solution for the ordered list with the weight annealing algorithm for 1BP
- Output results

WAMC for the MCBPP

Step 0. Initialization
Parameters are K (scaling parameter), *nloop1*, *nloop2*, T (temperature), and *Tred*.
Set $K = 0.05$, *nloop1* = 20, *nloop2* = 50, $T = 1$, and *Tred* = 0.95.
Inputs are number of items (n), the item size ordered list, bin capacity (c),
and number of bins (m).

Step 1. Compute the upper bound $U^* = \min\{\overline{U}_0, \overline{U}_1, \overline{U}_2\}$.

Step 2. Set $n = U^*$.
Remove item $i > U^*$ from the ordered list.

Step 3. Improve the upper bound.
While ($L_3 > m$) do{
$U^* = U^* - 1$.
Remove item $i > U^*$ from the ordered list.
Compute L_3. }

Step 4. For $j = 1$ to *nloop1*

Step 4.1 Construct initial solution with the ordered list and modified FFD algorithm.

Step 4.2 Improve the current solution.

Set $T = 1$.

Compute residual capacity r_i of bin i.

WAMC for the MCBPP

Step 4. (continued)

For k =1 to *nloop2*

Compute weights $w_i^T = (1+ K\, r_i)^T$.

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 j loop and k loop if m is reached.

Exit Swap (1,0) if no feasible move with $\Delta f_{(1,0)} \geq 0$ is found.

Perform Swap (1,1); Swap (1,2); Swap (2,2).

T: = $T \times Tred$

End of k loop

End of j loop

Step 5. Outputs are the number of bins used and the final distribution of items.

Test Problems

- Labbé, Laporte, and Martello (2003)
  - 180 combinations of three parameters
    - number of bins m = 2, 3, 5, 10, 15, 20
    - capacity c = 100, 120, 150, 200, 300, 400, 500, 600, 700, 800
    - size interval $[t_{min}, 99]$; t_{min} = 1, 20, 50
  - Total of 180 × 10 = 1800 randomly generated instances

- Peeters and Degraeve (2006)
  - 270 combinations of three parameters
    - capacity c = 1000, 1200, 1500, 2000, 3000, 4000, 5000, 6000, 7000, 8000
    - size interval $[t_{min}, 999]$; t_{min} = 1, 20, 50
    - desired number of items

$$\bar{n} = 100, 150, 200, 250, 300, 350, 400, 450, 500$$

$$m = \bar{n}(t_{\min} + 999)/2c$$

  - Total of 270 × 10 = 2700 randomly generated instances

Solution Procedures for MCBPP

- Enumeration algorithm (LLM) by Labbé, Laporte, and Martello (2003)
  - Compute a priori upper bounds
  - Embed the upper bounds into an enumeration algorithm
- Branch-and-price algorithm (BP) by Peeters and Degraeve (2006)
  - Compute a priori and LP upper bounds
  - Solve the problem with heuristics in a branch-and-price framework

Computational Results

- Results on the test problems of Labbé, Laporte, and Martello (2003)
  - Generated 1800 problems for testing on WAMC
  - LLM and BP used *different* sets of 1800 problems
  - Number of instances solved to optimality
    - BP 1800
    - WAMC 1793
    - LLM 1759
  - Average running time
    - BP < 0.01 sec (500 MHz Intel Pentium III)
    - WAMC 0.03 sec (3 GHz Intel Pentium IV)
    - LLM 3.16 sec (Digital VaxStation 3100)

Computational Results

- Generated 2700 problems for testing on WAMC; BP used a *different* set of 2700 problems
- Computational Results

Solution Procedures	Number of Instances Solved to Optimality	Average Running Time (sec)
WAMC	2665	0.20
BP	2519	2.85

- WAMC outperformed BP
  - BP had difficulties solving instances with
    - Large bin capacities (5000-8000)
    - Large number of items (350-500)
  - WAMC solved all instances with bin capacities ≥ 2000
  - WAMC was faster

Computational Results

Desired	Bin Capacity									
	1000		1200		1500		2000		3000	
Number of items	BP	WA	BP	WA	BP	WA	BP	WA	BP	WA
100	30	30	30	30	30	27	30	30	30	30
150	30	30	30	29	30	29	30	30	30	30
200	30	28	30	28	30	29	30	30	30	30
250	30	29	30	29	30	29	30	30	30	30
300	30	29	30	28	30	28	30	30	30	30
350	30	29	30	28	30	30	30	30	30	30
400	30	28	30	30	30	28	30	30	30	30
450	30	29	30	27	30	26	30	30	29	30
500	30	29	30	30	30	29	30	30	29	30
Total	270	261	270	259	270	255	270	270	268	270

Computational Results

Desired	Bin Capacity									
	4000		5000		6000		7000		8000	
Number of items	BP	WA	BP	WA	BP	WA	BP	WA	BP	WA
100	30	30	30	30	30	30	30	30	30	30
150	30	30	30	30	30	30	30	30	30	30
200	30	30	30	30	30	30	30	30	30	30
250	30	30	30	30	30	30	30	30	30	30
300	30	30	21	30	23	30	23	30	30	30
350	30	30	20	30	20	30	20	30	29	30
400	30	30	20	30	20	30	20	30	20	30
450	30	30	20	30	20	30	20	30	19	30
500	27	30	20	30	20	30	20	30	19	30
Total	267	270	221	270	223	270	223	270	237	270

Conclusions

- WAMC is easy to understand and simple to code
  - Weight annealing has wide applicability(1BP, 2BP)
- WAMC produced high-quality solutions to the maximum cardinality bin packing problem
- WAMC solved 99% (4458/4500) of the test instances to optimality with an average time of a few tenths of a second
- WAMC is a promising approach that deserves further computational study