Volume 5, No. 8, August 2014

Journal of Global Research in Computer Science

RESEARCH PAPER

Available Online at www.jgrcs.info

ISSN-2229-371X

A NOVEL GRAPH BASED ALGORITHM FOR ONE DIMENSIONAL BIN PACKING PROBLEM

Debajit Sensarma[*1], Samar Sen Sarma[*2]

[*1]Department of Computer Science & Engineering, University of Calcutta, Kolkata, West Bengal, India
debajit.sensarma2008@gmail.com[1]

[*2] Department of Computer Science & Engineering, University of Calcutta, Kolkata, West Bengal, India
sssarma2001@yahoo.com[2]

Abstract: The Bin Packing Problem (BPP) is one of the most known combinatorial optimization problems. The main objective of the problem is to minimize the number of bins used and pack the items with different sizes in finite number of bins efficiently. This paper introduces a new graph based algorithm for one dimensional bin packing problem. The proposed algorithm is implemented and tested with the well known benchmark instances and a comparison with existing First-Fit Decreasing (FFD) algorithm is given with respect to number of bins and waste space. In most of the cases the new algorithm produces near optimal solutions and performs better than FFD.

Keywords: Graph, Bin Packing Problem, Combinatorial Optimization, Heuristics

1. INTRODUCTION

In one-dimensional Bin Packing Problem, a sequence of items L = (a_1, a_2, …, a_n) is given, each of size S(a_i), i=1,.., n and the goal is to pack them into minimum number of bins with pre specified capacity C (i.e. partition them into a minimum number m of subsets B_1, B_2, …, B_m such that $\sum_{a_i \in b_j} s(a_i) \le C$, $1 \le j \le m$). In the view of constraint satisfaction problem Bin Packing problem is a problem of partitioning the set L under a sum constraint i.e. divide L into a minimum number of blocks, called bins such that the sum of sizes of the items in each bin is at most a given capacity C>0. There are various real world applications of bin packing problem e.g. stock cutting problem, Television programming problem, computer storage allocation problem, Transportation problem. In many of the problems Bin Packing has been presented as the primary combinatorial optimization problem, secondary problem or an embedded special case. For example, in container loading problem Bin Packing is embedded.

Bin packing can be defined in several ways. One classification is based on their dimension. There is one-dimensional, two-dimensional, three-dimensional and multi-dimensional Bin packing. On the other hand the problem has two types of packing, fixed sized Bin packing problem and variable sized Bin packing problem. In case of the fixed sized, the bin capacity is fixed and it may not have different capacity. The objectives are:

i) To minimize the number of bins that will not exceed its capacity.

ii) To minimize the Waste Space.

iii) To minimize the time of execution.

In case of variable sized packing problem the capacity of the bins varies. The objective is to pack the items with above constraints and minimizing the cost associated with the chosen bins.

The paper is organized as follows. Section 2 gives the problem formulation. The existing algorithms of Bin Packing Problem are depicted in section 3. In section 4 the proposed algorithm is described. The experimental results are given in section 5 and section 6 concludes the paper.

2. PROBLEM FORMULATION

Given 'B' identical bins of capacity 'C' and a list of n items with sizes a_1, a_2,…, a_n to pack and a compatibility graph G (V, E), where V is the set of size of the items and E is the set of edges such that (i, j) $\in$ E if item i and j are compatible (i.e. S (i) + S (j) $\le$V). The problem is to assign items to bins, using a minimum number of bins, while ensuring that the total size of the items assigned to a bin does not exceed the bin capacity C and that two items that are compatible are assigned to the same bin. The number B is assumed to be large enough to guarantee feasibility; more precisely it is a valid upper bound on the number of bins in an optimal solution (note that B $\le$ n). The possible integer programming formulation of the problem is:

$$\text{Minimize } B=\sum_{k=1}^{B} y_k \qquad \text{……. (i)}$$

$$\text{Such that, } \sum_{i=1}^{n} a_i \; x_{ik} \le C\, y_i \qquad k=1,\dots,B \qquad \text{…….. (ii)}$$

$$\sum_{k=1}^{B} x_{ik} = 1 \quad i=1,\dots,n \qquad \text{……..(iii)}$$

$$x_{ik} + y_{jk} \le y_k \quad (i,j) \notin E,\ k=1,\dots,B \qquad \text{……(iv)}$$

$$y_k \in \{0,1\} \qquad k=1,\dots,B \qquad \text{….(V)}$$

$$x_{ik} \in \{0,1\} \qquad i=1,\dots,n,\ k=1,\dots,B \qquad \text{….(vi)}$$

Where y_k =1 if bin k is used, 0 otherwise and x_{ik} =1 if item i is put into bin k, 0 otherwise.

In the Constraint (i) the objective function is to minimize the total number of bins and pack all the items with identical capacity. Constraint (ii) guarantees that the size of items (a_i) filled in the bin k do not exceed the bin capacity. Constraint (iii) ensures that each item is placed only in one bin. Constraint (iv) formulate the compatibility.

3. BIN PACKING ALGORITHMS

Bin packing is a NP-Hard problem [1, 2]. Many heuristic and approximation algorithms have been proposed to reach the near optimal solution. Mainly the algorithms are online and offline.
On-line algorithms permanently assign the objects to a bin in the sequence they arrive. There is an initial condition for all the on-line heuristics that the first object is already packed bin. There are various online algorithms. Some of them are:

A. Next Fit Heuristic .

In this algorithm items are placed in the order in which they arrive. The task is to place the next item as well as it arrives into the current bin if it fits, otherwise close that bin and start a new bin.

B. First Fit Heuristic .

In this algorithm items are placed in the order in which they arrive. As soon as the item arrives, the algorithm places the item into the lowest numbered bin in which it fits. If the item does not fit into any open bin, start a new bin.

C. Best Fit Heuristic .

This algorithm place the items in the order in which they arrive. Place the next item into that bin which will leave smallest residual capacity after the item is placed in the bin. If item does not fit in any bin, start a new bin.

D. Worst Fit Heuristic .

This algorithm places the items in the order in which they arrive and place the next item into that bin which will leave the largest residual capacity after the item is placed in the bin. If it does not fit in any bin, start a new bin.

Off-line algorithms have all the objects available before the packing starts. Two well known off-line algorithms are described below:

E. First Fit Decreasing Heuristic .

The algorithm first Sort the items in decreasing order then place the next item into the lowest numbered bin in which it fits. If it does not fit into any open bin, start a new bin.

F. Best Fit Decreasing Heuristic .

The algorithm first sort the items in decreasing order and place the next item into that bin which will leave the smallest residual capacity after the item is placed in the bin. If it does not fit in any bin, start a new bin.

Besides the development of approximation algorithms for the classical problem [4], there exist some variants of the problem. (a) Bin Packing Problem with number of items maximized. [5] (b) Bin Packing Problem with a bound on number of items can be packed in each bin. [6] (c) Class constrained Bin Packing Problem. [8] (d) Dynamic Bin Packing Problem. [7] (e) Bin Packing with constrained on data. [9, 10] (f) Bin stretching problem. [11]

4. PROPOSED ALGORITHM

The algorithm proposed here is graph [3] based. It is designed for offline one dimensional bin packing problem. The algorithm is a collection of two algorithms. (a) Algorithm B: Bin Counting and (b) Algorithm C: Construct Compatibility graph. Algorithm C creates the compatibility graph from the given set of sizes of the input items and Algorithm B count the number of bins required. Waste Space per bin is calculated as difference between total capacity of the bin and sum of sizes of selected item set. The algorithm finds minimal number of bins in reasonable amount of time and space. The two algorithms are depicted below.

Algorithm B: Bin Counting

Input: Compatibility Graph, n= Number of items, deg= Degree of each item.

Output: B= Number of Bins.

B1: [Initialize] Set i $\leftarrow$ 1, Count $\leftarrow$ 0, B $\leftarrow$ 0;

B2: [Sort items] Sort the input items in non-increasing order of their sizes.

B3: [Compatibility graph construction] Construct Compatibility graph and compute degree (deg) of each items.

B4: [Check if (i $\le$n)?] If (i $\le$n) then goto step B5 else goto step B20.

B5: [Scan adjacent items] Scan for adjacent items of i.

B6: [Check if $\deg_i$ =0?] If i contains no adjacent elements, goto step B17 else goto step B7.

B7: [Check if $\deg_i$ =1?] If i contains one adjacent element, goto step B17 else goto step B8.

B8: [Check if Count $\le \deg_i$] If (Count $\le \deg_i$) goto step B9 else goto step B11.

B9: [Item selection] Take the adjacent item with better fitted in the bin and will left the bin with minimum waste space.

B10: [Increment Count] Count $\leftarrow$ Count+1, goto step B8.

B11: [Check if waste space =0?] If (waste space $\leftarrow$ 0) then goto step B16 else goto step B12.

B12: [Compute of all possible sum] Compute all possible sum of subsets of adjacent elements of i by taking 3, 4,…,q items at a time and store the items which will left the bin with minimum Waste Space.

B13: [Check if waste space =0?] If (waste space $\leftarrow$ 0) then goto step B17 else goto step B14.

B14: [Compare waste space] Compare waste space of step B9 and Step B12.

B15: [Item set selection] Choose the items which will best fit in the bin and left the bin with minimum waste space.

B16: [Reset Count] Set Count $\leftarrow$ 0.

B17: [Delete item/ item set] If ($\deg_i \leftarrow 0$) then delete the item i else delete the item i and its selected adjacent item/items from the adjacency list.

B18: [Set B, n] Set B $\leftarrow$B+1, n $\leftarrow$n- e /* e number of item/ items chosen */

B19: [Set i] Set i $\leftarrow$1, goto step 4.

B20: [Terminate] Stop.

Algorithm C: Construct Compatibility graph

Input: Array= Set of sizes in non-increasing order, n= Number of items.

Output: Compatibility graph, deg= Degree of each item.

C1: [Initialize] Set i$\leftarrow$ 1, j $\leftarrow$ 1, deg $\leftarrow$ 0.

C2: [Check if i $\leq$ n?] If (i $\leq$n) then goto step C3 else goto step C10.

C3: [Check if j $\leq$n?] If (j $\leq$ n) then goto step C4 else goto step C9.

C4: [Check if i$\neq$ j?] If (i$\neq$ j) goto step C5 else goto step C7.

C5: [Check sum of items less than or equal to Capacity?] If (Array[i] + Array[j] $\leq$Capacity) then goto step C6 else goto step C7.

C6: [Connect items] Connect item i and j.

C7: [Increase $\deg_i$] $\deg_i \leftarrow \deg_i$ +1.

C8: [Reset j] Set j$\leftarrow$ j+1, goto step C3.

C9: [Reset i] Set i$\leftarrow$ i+1, $\deg_i \leftarrow$ 0, goto step C2.

C10: [Terminate] Stop.

5. EXPERIMENTAL RESULTS

The program is done on Intel® Atom™ Processor, 1.60 GHz, 1.0 GB DDR2 RAM and with Borland C++ 5.5 compiler.

This section evaluates results of FFD algorithm and the proposed algorithm with respect to i) Number of bins ii) Waste Space (%) = (($\sum_{j=1}^{B} C - y_j$) / (C*Number of bins))*100 . Here y_j is sum of sizes of items in bin j, C= capacity of each bins. The test instances are taken from the OR-library [12]. The benchmark datasets are divide into three classes; easy, medium and hard class instances. Table-I contains the results of easy instances. Out of 5 instances the proposed algorithm produces optimal solution for all but FFD produces for 3 instances. Table-II contains results for the medium class instances. Out of 5 instances proposed algorithm produces optimal solution for 4 instances and near optimal solution for 1 instance but FFD produces optimal solution for only 1 instance. Lastly results of Hard class instances are shown in Table-III where out of 5 instances proposed algorithm produces near optimal solution for all 5 instances and which is better than FFD. From the above three tables its can be seen that the Waste Space (%) of the proposed algorithm is comparatively less than FFD. Here, N represents number of items, C represents bin capacity. We have chosen q= 2, 3, 4, 5.

Table I. Comparative results for easy instances

			Number of Bins			***Waste Space (%)***	
Instance (Easy)	***N***	***C***	***FFD***	***Propo sed method***	***Minimal number of Bins***	***FFD***	***Propsed method***
N1C2W4_D	50	120	**28**	**28**	28	9.88	9.88
N1C1W1_N	50	120	26	**25**	25	6.88	3.16
N3C1W1_A	50	100	106	**105**	105	4.83	3.92
N2C3W2_P	50	150	**41**	**41**	41	2.05	2.05
N3C1W4_R	50	100	**145**	**145**	145	11.48	11.48

Table II. Comparative results for medium instances

			Number of Bins			***Waste Space (%)***	
Instance (Medium)	***N***	***C***	***FFD***	***Pro Posed metho d***	***Mini mal num ber of Bins***	***FFD***	***Pro posed metho d***
N1W1B1_R0	50	1000	20	**18**	18	15.33	5.92
N1W1B1_R1	50	1000	20	**18**	18	15.27	5.86
N1W1B1_R7	50	1000	19	**17**	17	12.69	2.42
N2W1B1_R0	100	1000	37	35	34	11.10	6.02
N3W4B3_R0	200	1000	**24**	**24**	24	0.28	0.28

Table III. Comparative results for hard instances

			Number of Bins			***Waste Space (%)***	

Instance (Hard)	*N*	*C*	*FFD*	*Prpo Sed method*	*Minimal number of Bins*	*FFD*	*Proposed method*
Hard0	200	100000	59	56	57	7.79	4.56
Hard1	200	100000	60	58	57	7.68	4.50
Hard2	200	100000	60	58	56	7.40	4.21
Hard3	200	100000	59	57	55	7.77	4.53
Hard4	200	100000	60	58	57	7.48	4.30

6. CONCLUSION

In this paper a heuristic is proposed to tackle one-dimensional bin packing problem. The proposed algorithm is a graph based offline algorithm. A compatibility graph is constructed from the set of item sizes where item sizes are acts as nodes of the graph and two nodes as connected if they are compatible with respect to a capacity constraint. Experiment on some problem instances show the supremacy over existing offline FFD algorithm with respect to number of bins and total waste space.
In future we will experiment the proposed algorithm with other instances and will try to apply the algorithm in real life problem solving.

ACKNOWLEDGMENT

The authors would like to thank University Of Calcutta, West Bengal, India, Department of Science & Technology (DST), New Delhi, for financial support and the reviewers for their constructive and helpful comments and specially the Computer without which no work was possible.

REFERENCES

[1] Garey, Michael R., and David S. Johnson. "*Computers and intractability*." Vol. 29. wh freeman, 2002.

[2] Basu, S. K. "*Design methods and analysis of algorithms*." PHI Learning Pvt. Ltd., 2013.

[3] Deo, Narsingh. "*Graph theory with applications to engineering and computer science*." PHI Learning Pvt. Ltd., 2004.

[4] Gonzalez, Teofilo F., ed. "*Handbook of approximation algorithms and metaheuristics*." CRC Press, 2007.

[5] Coffman Jr, Edward G., JY-T. Leung, and D. W. Ting. "*Bin packing: maximizing the number of pieces packed*." Acta Informatica 9.3 (1978): 263-271.

[6] Krause, K. L., V. Y. Shen, and Herbert D. Schwetman. "*Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems.*" Journal of the ACM (JACM) 22.4 (1975): 522-550.

[7] Coffman, Jr, Edward G., Michael R. Garey, and David S. Johnson. "*Dynamic bin packing.*" SIAM Journal on Computing 12.2 (1983): 227-258.

[8] Shachnai, Hadas, and Tami Tamir. "*Tight bounds for online class-constrained packing.*" Theoretical Computer Science 321.1 (2004): 103-123.

[9] Liu, Wei-Ping, and Jeffrey B. Sidney. "*Bin packing using semi-ordinal data.*" Operations Research Letters 19.3 (1996): 101-104.

[10] Mandal, C. A., P. P. Chakrabarti, and S. Ghose. "*Complexity of fragmentable object bin packing and an application.*" Computers & Mathematics with Applications 35.11 (1998): 91-97.

[11] Azar, Yossi, and Oded Regev. "*On-line bin-stretching.*" Theoretical Computer Science 268.1 (2001): 17-41.

[12] http://www.wiwi.uni-jena.de/entscheidung/binpp/

BIODATA

Debajit Sensarma is presently pursuing his PhD degree from the department of Computer Science and Engineering, University of Calcutta, Kolkata, India with DST INSPIRE Fellowship. He has published several papers in International journals and conferences.

Dr. Samar Sen Sarma is presently working as the Professor of the department of Computer Science and Engineering, University Of Calcutta, Kolkata, India. He has published several papers in International journals and conferences.