A Graph Theoretic Approach for Minimizing Storage Space using Bin Packing Heuristics

Debajit Sensarma
Dept. of Computer Science & Engineering
University of Calcutta
Kolkata, India

Samar Sen Sarma
Dept. of Computer Science & Engineering
University of Calcutta
Kolkata, India

Abstract—**In the age of Big Data the problem of storing huge volume of data in a minimum storage space by utilizing available resources properly is an open problem and an important research aspect in recent days. This problem has a close relationship with the famous classical NP-Hard combinatorial optimization problem namely the "Bin Packing Problem" where bins represent available storage space and the problem is to store the items or data in minimum number of bins. This research work mainly focuses on to find a near optimal solution of the offline one dimensional Bin Packing Problem based on two heuristics by taking the advantages of graph. Additionally, extreme computational results on some benchmark instances are reported and compared with the best known solution and solution produced by the four other well-known bin oriented heuristics. Also some future directions of the proposed work have been depicted.**

Keywords—Bin Packing; Combinatorial Optimization; Graph Theory; Heuristics; Operational Research

I. Introduction

The storage space minimization problem is an open problem of now-a-days as the sizes as well as the dimension of data are increasing day by day. So, there is a need to produce a near optimal solution in less amount of time. To tackle with the problem the author have considered the storage minimization problem as the famous one dimensional Bin Packing Problem where storage space can be represented as bins and the problem is to store the items or data in minimum number of bins. This problem arises in a wide variety of contexts and this popular combinatorial optimization problem has been extensively studied during past few years. The authors [1] called the problem as "The Problem That Wouldn't Go Away". The study of classical one dimensional Bin Packing Problem first begins in the early 1970's [2]. The problem states that, an unlimited number of bins with integer capacity $C>0$ each, a set of items with their weights, w_i, $0< w_i \leq C$ are given. The goal is to assign each item to one bin, such that total weight of the items in each bin does not exceed the capacity C and the number of bins used for packing all items is minimized. The problem is known to be NP-Hard is strong sense [3]. Thus, in this case the satisfying solution is to design an approximation algorithm which will construct near-optimal packing.

One dimensional Bin Packing Problem has several applications in real world, among them resource and storage space minimization is one facet. Some formulations of real world storage minimization problem using Bin Packing Problem are as follows: i) Placing computer files with specified size into the identical disk with same capacity with constrained that each file must be entirely on one disk [4]. The objective is to minimize the number of disks needed for the set of files. This can be formulated using Bin Packing Problem where items are files, disks are bins and disk capacity is the bin capacity which is fixed. The problem is to minimize the number of bins. ii) Server Consolidation [5] is an approach to the efficient usage of computer server resources in order to reduce the total number of servers or server locations that an organization requires. In this case, existing servers can be treated as items, resource utilizations are item sizes, bins are destination servers and the bin capacity is the utilization threshold of the destination servers. The goal is to minimize the destination servers and maximizing resource utilization. With one resource the problem is same as one dimension Bin Packing Problem. Additionally, with more than one resource (e.g. CPU, disk, memory and computer network) the problem dimension increases. iii) Also the Bin Packing Problem can be used to minimize the cost of storing data (items) in the cloud storage [6]. As buying hard-drive in bulk is much cheaper than buying them individually, the goal of solving the problem becomes minimizing the hard-drives (bins) to store the data (items). Besides this, there are other storage minimization problems where Bin Packing has a major role, but are not discussed in this paper.

Not only Bin Packing Problem but also graph theory has vast real world applications. Graph algorithm provides unified solution approach to many classical and modern application areas by taking graph as an omnipotent mathematical tool. In view of storage minimization problem, there exists various graph compression mechanism which can be used to store data compactly [7].

This paper mainly focused on the solution of one dimensional Bin packing problem in polynomial time, and for this an algorithm depending on two offline bin oriented heuristics has been proposed taking the advantages of graph theory. Firstly, a vertex weighted graph is constructed from the set of item weights where for each item weights one vertex is created. Then, the first heuristic chooses the subset of vertices according to the maximum total weight criteria and the second one is based on maximum average weight criteria, which ultimately produces the minimal clique partition of the graph with each clique having weight not exceeding the capacity of each bin. The total number of partition gives the total number of bins. The algorithm runs in polynomial time.

Most of the existing algorithms not completely based on graph algorithm rather hybridization of graph algorithms but this work is completely based on a graph algorithm to find minimum clique Partition with weight constraint and can compete with existing algorithms. Also it can open a new direction for solving multi-dimensional Bin Packing Problem. The detailed description of the algorithm can be found in subsequent sections.

The article is organized as follows: section II contains some preliminary concepts related to the work. Some existing work to tackle Bin Packing problem with graph is described in section III. Section IV gives the detailed description of proposed algorithm. Section V contains computational results. Finally, section VI concludes the article giving some future scopes in section VII.

II. Preliminaries

This section contains some preliminary concepts related to the topic, taken from [8, 9, 10, 11].

Definition 2.1: A **Graph** G is a triple consisting of a vertex set V (G), an edge set E(G), and a relation that associates with each edge two vertices (not necessarily distinct) called its endpoints.

Definition 2.2: vertex weighted graph is a graph where each vertex has been assigned a positive weight.

Definition 2.3: A **Null Graph** is a graph whose edge set is null.

Definition 2.4: A **Clique** in a graph G is a set of pairwise adjacent vertices.

Definition 2.5: A vertex x of a graph G is **Simplical Vertex** if its adjacency set Adj(x) induces a complete subgraph of G, i.e. Adj(x) is a clique (not necessarily maximal).

Definition 2.6: An ordering δ = [v_1, v_2, …, v_n] where n is the number of vertices of an undirected graph G is **Perfect Elimination Ordering** iff each v_i is a simplical vertex of the induced sub graph $G_{\{vi,\ldots vn\}}$.

Definition 2.7: Chordal Graph is a simple graph in which every cycle of length four and greater has a cycle chord.

Definition 2.8: Given a vertex weighted graph G = (V, E; W}, having weight of vertices $w_1, w_2, \ldots, w_{|V|}$ respectively and a bound C′, the **Minimum Clique Partition with Constrained Weight (MCPCW)** problem [12] is to find a partition of these |v| vertices into smallest number of cliques such that each clique has its weight not beyond C′.

Theorem 2.9: Minimum Clique Partition with Constrained Weight (MCPCW) problem is NP-Hard.

Proof. The proof is done by transforming an instance of 3-Partition problem to an instance of **MCPCW**.

Consider an instance P of 3-Partition problem: Given the set S = {$a_1, a_2, \ldots, a_{3k}$}of 3k integers satisfying $C'/4 < a_j < C'/2$ for each $1 \le j \le 3k$ and $\sum_{j=1}^{3k} a_j = kC'$. The problem asks whether S can be partitioned into k subsets $s_1, s_2, \ldots, s_k$, such that for each i= 1, 2,…, k, s_i contains exactly three elements of S and $\sum_{a \in s_i} a = C'$.

Now we will construct a polynomial time reduction Q for P of the 3-partition problem to an instance Q(P) of the **MCPCW** problem i.e. a vertex weighted graph with weight of each vertices $w_1, w_2, \ldots, w_{3k}$ respectively where $w_i = a_i$ for each i= 1,… 3k and the bound $C' = (\sum_{j=1}^{3k} w_j)/k$.

We now prove the claim that there exists a feasible solution to an instance P of the 3-partition problem iff instance Q(P) of the **MCPCW** problem has its optimal solution. So, the feasible partition of the instance Q(P) can be constructed in the following way: for each $s_i = \{a_{i_1}, a_{i_2}, \ldots, a_{i_l}\}$, select the clique $c_1 = \{w_{i_1}, w_{i_2}, \ldots, w_{i_l}\}$ and then obtains a partition of these 3k weights of |V| vertices into k cliques having weight exactly C′. Conversely, if the instance Q(P) of the problem **MCPCW** has an optimal clique partition { $c_1, c_2, \ldots, c_k$} with the smallest integer k having $\sum_{w \in c_i} w \le C'$ for each i=1,…,k. By the facts, $\sum_{j=1}^{3k} a_j = kC'$ and $C'/4 < a_j < C'/2$ for each $1 \le j \le 3k$, we obtain $\sum_{w \in c_i} w = C'$ (as $w_j = a_j$ for each j= 1,… 3k) for each j= 1, 2,…, k and the clique c_j contains exactly three elements from S, i.e. $s_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$ and $\sum_{a \in c_j} a = C'$ j= 1,… k. So, the instance p of the 3-partition problem has the partition $s_1, s_2, \ldots, s_k$.

Definition 2.10: A **bin oriented heuristic** for Bin Packing Problem constructs solution bin by bin i.e. while unpacked items remain it is packed with the maximal subset of unpacked items, e.g. First Fit Decreasing (FFD), Best-Two-Fit (B2F), Minimum Bin Slack (MBS), MBS′ etc. [13, 14].

Definition 2.11: Offline algorithms have all the items available before the packing starts, e.g. First Fit Decreasing (FFD) [4].

III. Related Works

This section consists of some related works to solve one dimensional Bin Packing Problem based on graphs. Firstly, in [15] the authors consider time constrained scheduling problem. For a set of jobs J with execution time $t(j) \in (0, 1]$ and an undirected graph (the conflict graph) G =(J, E), they consider to find schedule of the jobs that are adjacent and they are assigned different machines (bins) with total execution time of each machine at most 1. The objective is to assign all jobs into minimum number of machines maintaining the time constraint. To tackle the problem, they have proposed six different algorithms based on different principles. The first three algorithms are the modification of classical NF, FF, FFD algorithms. Next algorithm depends on optimal coloring algorithm which finds a minimum partition of the item set into independent sets which is equal to the chromatic number of G and applies one of the NF, FF and FFD packing to each independent set. Fifth and sixth algorithm is same like above but the main difference is fifth one is based on pre-coloring method and sixth one is based on general coloring method that works for co-graph and k-trees. Next, the authors of [16] consider the problem namely Bin Packing with Conflict (**BPC**) using conflict graph and it's online, offline versions. They mainly improve the upper bounds of BPC on perfect graphs, interval graph and bipartite graphs. Most of the recent results follow from the adaptation of weighting systems to enable analysis of algorithms for BPC and new algorithms which carefully remove small sub-graph of items causing problematic instances. In next work [17] authors considered a restricted problem called Bin Packing with Clique Graph Conflicts. They have designed a polynomial time approximation algorithm for constant item size analyzing its performance in the more general case of bounded item sizes. In [18] authors investigated the following problem: the items to be packed are structured as the leaves of a tree and it is called as Structured Bin Packing Problem. The objective is to pack the items in the same bin whose lowest common ancestor has low height. Next, authors of [19] have proposed a problem to pack a graph G with lower and upper bound on its edges and weights on its vertices into a host graph I and called the problem as Graph Bin Packing Problem. The vertices of G are items to be packed and vertices of I are bins. The host vertex can accommodate at most L weight in total and if two items are adjacent in G, then the distance of their host vertices in I must be between the lower and upper bounds on the edge joining the two items.

Most of the above algorithms not completely based on graph algorithm rather hybridization of graph algorithms and exiting heuristics for solving Bin Packing Problem. Our work is simple and purely based on a graph algorithm namely finding minimum clique Partition with weight constraint and can compete with existing algorithms. Also it can open a new direction for solving multi-dimensional Bin Packing Problem.

IV. The Proposed Algorithm

Let, W= {w_1, w_2, …,w_n} be the given sequence of weights of the items. The items are numbered 1 through n, from the left to right of the list, labeling their positions in W, i.e. w_1 is the weight of first item in W, w_2 is the weight of second item in W and so on.

In this section an algorithms based on two bin oriented heuristics has been formulated based on graph to cope with the one dimensional Bin Packing Problem.

In this algorithm firstly items are sorted in non decreasing order with respect to their weight. Next, a vertex weighted graph is constructed from the sequence of items. Here, for each item a weighted vertex are introduced. Hence, firstly the graph consists of 'n' isolated vertices {v_1,…,v_n} with their weight {w_1, w_2, …,w_n} respectively. Now, for introducing edges to the graph, the following procedure is being followed. For any pair of items with weight w_i and w_j that are in the position i and j, respectively in W, an edge is introduced between the corresponding vertices of w_i and w_j, only if (i-j)(w_i-(C-w_j)) ≥ 0, where C is the capacity of each bin. In other words, an edge is introduced between the corresponding vertices in the graph if they satisfy the condition $w_i + w_j \leq C$. This is explained with an example below.

Example 4.1: Suppose, W = {8, 11, 10, 4, 7, 9, 3}, C = 15.

After sorting the sequence is W′ = {11, 10, 9, 8, 7, 4, 3}.

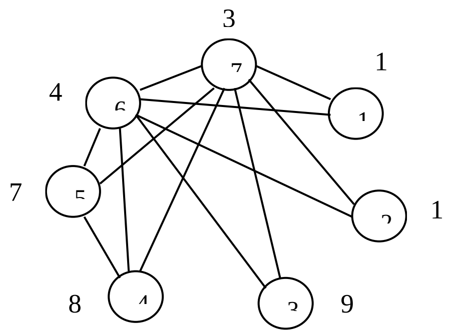


Fig. 1. Vertex weighted graph for the sequence W = {8, 11, 10, 4, 7, 9, 3} and bin capacity C=15. Vertex 1 has weight 11, vertex 2 has weight 10 and so on. An edge {vi, vj} indicates that wi + wj ≤ C.

Lemma 4.2: The Graph produced from the sequence W′ after sorting the sequence W in non increasing order (i.e. $w_1 \geq w_2 \geq \ldots \geq w_n$), has a Perfect Elimination Ordering.

Suppose, $W_i \geq W_j \geq W_k$ and vertex i and j are connected. The following equations are satisfied.

$W_i + W_j \leq C$ … (1)
$W_i + W_k \leq C$ … (2)

If the ordering is the perfect elimination ordering then, vertex j and k will also be connected and $W_j + W_k \leq C$.

Adding (1) and (2)

$2W_i + (W_j + W_k) \leq 2C$

Or, $W_j + W_k \leq 2(C-W_i)$… (3)

As, $W_j \leq W_i$

Putting $W_j = W_i$ we get from (1)

$2W_i \leq C$

From, (3) we get,

$W_j + W_k \leq C$

This condition is applicable for the whole ordering. So, the ordering is the perfect elimination ordering.

Claim 4.5: There exists a feasible solution to an instance I of one dimensional Bin Packing Problem if and

only if the instance τ (I) of the MCPCW problem for Chordal Graph has its optimal solution with value k.

For any feasible solution of an instance I of Bin Packing Problem, the set of items B is partitioned into k bins $\{B_1, B_2, \ldots, B_k\}$, $\forall$ i=1, …, k, such that each B_i contains items of B and $\sum_{a \in B_i} a \le C$ (C=capacity of each bin). Then a feasible partition of the instance τ (I) of the MCPCW problem can be constructed in the following way: for each $B_i = \{a_{i_1}, a_{i_2}, \ldots, a_{i_l}\}$, select the clique $C_i = \{w_{i_1}, w_{i_2}, \ldots, w_{i_l}\}$ having total weight of the vertices not exceeding C. Likewise obtain the partition of total items into k cliques each having total weight not exceeding C.

Conversely, if the instance τ (I) of the MCPBW problem has optimal clique partition $\{C_1, C_2, \ldots, C_k\}$, $\forall$ i=1, …, k, with smallest integer k and having $\sum_{a \in c_i} a \le C$. Then each clique contains items from the set B, i.e. $B_j = \{a_{i_1}, a_{i_2}, \ldots, a_{i_l}\}$ and $\sum_{a \in c_j} a \le C$, $\forall$ i=1, …, k. So, the instance I of Bin Packing Problem has the partition $\{B_1, B_2, \ldots, B_k\}$.

Algorithm 4.1: Counting Bins

Input: List of vertices (**n**) with their weights $\{\mathbf{w_1, w_2, \ldots, w_n}\}$, Capacity (**C**).
Output: Number of Bins (**B**).

```
Begin
Step 1: If (n! = 0) then go to step 2 else goto step 7.
Step 2: Sort the vertices according to non-increasing order of their weight.
Step 3: Call Algorithm 4.1.1.
Step 4: Call Algorithm 4.1.2.
Step 5: Assign clique partition number (obtained from step 4) of vertex weighted
graph (G) (produced by step 3) with total weight of each clique ≤ C to B (i.e. B ⟵ CC (Clique Count)).
Step 6: Print B.
Step 7: End.
```

Algorithm 4.1.1: Construct_ Graph

Input: List of vertices with weights $\{\mathbf{w_1, w_2, \ldots, w_n}\}$, Capacity (**C**).
Output: Vertex weighted Graph (**G**).

```
Begin
Step 1: Set i ⟵ 1, j ⟵ 1.
Step 2: If ( i ≤ n) then goto step 3 else goto step 9.
Step 3: If (j ≤ n) then goto step 4 else goto step 8.
Step 4: If ( i ≠ j ) goto step 5 else goto step 7.
Step 5: If (w_i + w_j ≤ C) then goto step 6 else goto step 7.
Step 6: Connect item i and j.
Step 7: Set j ⟵ j+1, goto step 3.
Step 8: Set i ⟵ i+1, goto step 2.
Step 9: End
```

Algorithm 4.1.2: Minimum Clique Partition with Constrained Weight (MCPCW)

Input: Adjacency List of the Vertex weighted Graph (**G**), Capacity (**C**).
Output: Clique Count (**CC**).

```
Begin
Step 1: CC ⟵ 0;
Step 2: If (n!= 0) then goto step 3 else goto step 7;
Step 3: i ⟵ 1;
Step 4: If vertex i has zero or one neighbor, then delete the vertex along with its
neighbor (if any) from the Graph (G), CC ⟵ CC + 1 and goto step 2 else goto step 5;
Step 5: Select subset of vertices consisting of vertex i and its neighbor vertices based on Selection criteria 1 or Selection Criteria 2.
Step 6: Delete the subset produced from step 4, CC ⟵ CC + 1, goto step 2;
Step 7: End
```

As the subsets are the cliques, so algorithm 4.1.2 returns the number of clique partition with each partition weight not exceeding the capacity. The critical part of the algorithm 4.1 is step 4 of the algorithm 4.1.2 where subset of the vertices consisting of the current vertex and its neighbors has to be selected. Here, we have adopted two heuristics for selection of the subset. The selection criteria are depicted below:

A. *Selection Criteria 1 (A1):*

This criterion selects the subset of the current vertex along with its neighbor vertices which gives maximum total weight not exceeding the capacity (**C**).

B. *Selection Criteria 2 (A2):*

This criterion selects the subset of the current vertex along with its neighbor vertices which gives maximum average weight not exceeding the capacity (**C**). Here, firstly the total average weight ($\mathbf{T_a}$) of the vertex set is calculated. Suppose, average weight of current subset is $\mathbf{C_a}$ and average weight of its previous subset is $\mathbf{P_a}$, then if $C_a \ge P_a$ or $C_a \ge T_a$ and also total sum of current subset is greater than the previous one, current subset is selected as the final subset, otherwise previous subset is selected as the final subset and this process continues for all possible subsets.

Theorem 4.3: The graph G formulated by the Algorithm 4.1, is a Chordal Graph.

Proof. Let, there is a chordless cycle $v_1, v_2, \ldots, v_l$, with $l \geq 4$ in G. According to lemma 4.2, the graph G has perfect vertex elimination ordering. Suppose, v_i is the vertex in the cycle that occurs first in the perfect elimination ordering and v_{i+1}, v_{i+2} are neighbors of v_i occur later in the ordering. So, there must be an edge between v_{i+1} and v_{i+2}. But this contradicts the assumption that the cycle is chordless. So, the graph G is a Chordal Graph.

Claim 4.4: Any induced subgraph of the graph G produced by the Algorithm 4.1, is Chordal.

As the graph G produced by the Algorithm 4.1 is Chordal and any induced subgraph of a Chordal Graph is Chordal [20], so the above claim is also true for the graph produced by the Algorithm 4.1.

Lemma 4.6: Minimum Clique Partition Problem with Constrained Weight (MCPCW) for the Chordal Graph can be solved in O(|V| +|E|) time where V is the vertex set and E is the edge set.

According to lemma 4.2, the ordering $w_1 \geq w_2 \geq \ldots \geq w_n$ is a perfect vertex elimination ordering. Suppose, processing starts with vertex v_1 with weight w_1. It is added to the first partition. Next the adjacent vertices of v_1 are checked and the vertices are added along with v_1 to the partition with total weight $\leq C$, based on one of the two above selection criteria. If the first partition is $\{v_1, v_{2, \ldots,} v_k\}$, then after deletion of the vertices in the partition Algorithm 4.1 continues the execution with the remaining graph G´, which is also a Chordal Graph according to the lemma 4.4. The execution continues until vertex set is empty. In each iteration, Algorithm 4.1 checks the vertex and its neighbors. So, the overall complexity of the implementation is $O(|v|) + O\sum_{v\in V}(|\text{Adj}(v)|)$, which is roughly equivalent to O (|V| + |E|).

Theorem 4.7: Number of Bins Produced by the Algorithm 4.1 is $K \leq 3/2$ OPT +1 and time complexity is O $(|V|^2)$.

Proof. Assume, partition of the ordered list of vertex weights has to be done where the weights $\{\mathbf{w_1, w_2, \ldots, w_n}\}$ are distributed in the following sets:

$X = \{w_i \mid w_i > 2L/3\}$ {L=capacity of each Bin}

$Y = \{w_i \mid L/2 < w_i \leq 2L/3\}$

$T = \{w_i \mid L/3 < w_i \leq L/2\}$

$Z = \{w_i \mid w_i \leq L/3\}$

Case 1: There is one Clique Partition with all vertices from set Z.

1) In this case all partitions except the last one have used more than 2C/3 of the total capacities. Otherwise an item from set Z can put into them.

2) It has to be the last partition.

Suppose, required number of bins = K.

$\therefore 2L(K-1)/3 + (C_K) \leq \sum_{i=1}^{n} w_i$ [C_K = total weight of partition K]

$\Rightarrow 2(K-1)/3 + (C_K)/L \leq \left\lceil \sum_{i=1}^{n} w_i/L \right\rceil \leq$ OPT [$\because$ OPT= $\left\lceil \sum_{i=1}^{n} w_i/L \right\rceil$]

$\Rightarrow K \leq 3/2$ OPT + 1 -3/2. C_K/L [Clique Partition contains one vertex with weight= L/3]

$\Rightarrow K \leq 3/2$ OPT + 1 -1/2

$\Rightarrow K \leq 3/2$ OPT + 1

Case 2: There is no Clique Partition with all vertices from Z.

In this case all vertices from set Z can be thrown out without changing total number of partitions and below cases arise.

1) No partition has more than 2 items.

2) Any partition with one vertex from X cannot accommodate any other vertices.

3) Any partition with one vertex from Y can accommodate only another vertex from T.

4) Any Partition with one vertex from T can accommodate either one vertex from Y or one vertex from T but not both.

From the conclusion above we know that know that our algorithm will put at most 2 vertices in a bin. So, it put each vertex in a partition with maximum total weight (criteria 1) and maximum average weight (criteria 2). So, in this case the solution of proposed algorithm is optimal.

For the second part, it can be seen from the algorithm that for V number of vertices algorithm 4.1.1 construct the graph in $O(|V|^2)$ time and from Lemma 3.5 it can be concluded that algorithm 4.1.2 requires O(|V| + |E|) time. As time complexity of algorithm 4.1.1 dominates time complexity of algorithm 4.1.2; total time required by the algorithm 4.1 is $O(|V|^2)$.

C. Illustration of Algorithm 4.1 (Counting Bins) with examples

Suppose, set W is the set of vertex weights organized in non-increasing order of their sizes and $w_i \in Z^+ \ \forall$ i=1, 2 …, n. $W = \{w_1 \geq w_2 \geq \ldots \geq w_s > C/2 > w_{s+1} \geq w_{s+2} \geq \ldots \geq w_n\}$ and capacity=C. The optimal number of bins is calculated as $\left\lceil \sum_{i=1}^{n} w_i/C \right\rceil$.

Case 1: Input sequence: $y_1 \geq y_2 \geq \ldots \geq y_s > C/2$

The graph can be viewed as a null graph and in that case the number of cliques is the cardinality of the vertex weight set which is |W|.

Example 4.8: W = {15, 14, 12, 11, 10, 9, 8} and C=15.Optimal No. Bins= 6.

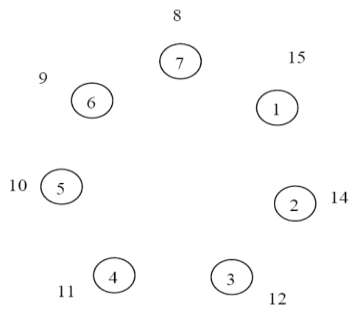


Fig. 2. Vertex weighted graph for case 1

So, with selection criteria 1 and selection criteria 2, the number of Bins=7.

Case2: Input sequence: $w_1 \geq w_2 \geq \ldots \geq w_s > C/2 > w_{s+1} \geq w_{s+2} \geq \ldots \geq w_n$

This graph is a Chordal graph. Number of bins can be found by finding minimum clique partition with total weight of each clique not exceeding C.

Example 4.9: W = {11, 10, 9, 8, 7, 4, 3} and C=15. Optimal no. of Bins=4

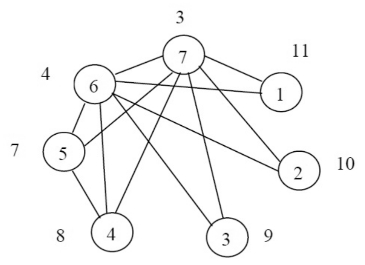
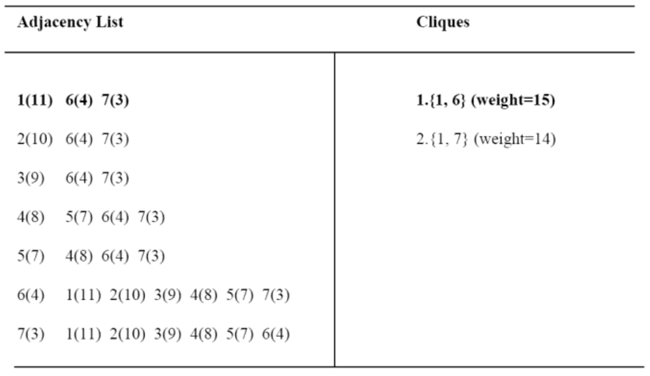


Fig. 3. Vertex weighted graph for case 2

Selection Criteria 1:



Adjacency List	Cliques
1(11) 6(4) 7(3)	**1.{1, 6} (weight=15)**
2(10) 6(4) 7(3)	2.{1, 7} (weight=14)
3(9) 6(4) 7(3)	
4(8) 5(7) 6(4) 7(3)	
5(7) 4(8) 6(4) 7(3)	
6(4) 1(11) 2(10) 3(9) 4(8) 5(7) 7(3)	
7(3) 1(11) 2(10) 3(9) 4(8) 5(7) 6(4)	

Fig. 4. Adjacency list of the graph in Figure.3 and possible cliques

Clique (C_1) = (11, 4).

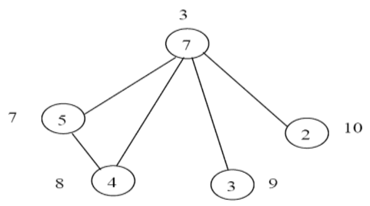
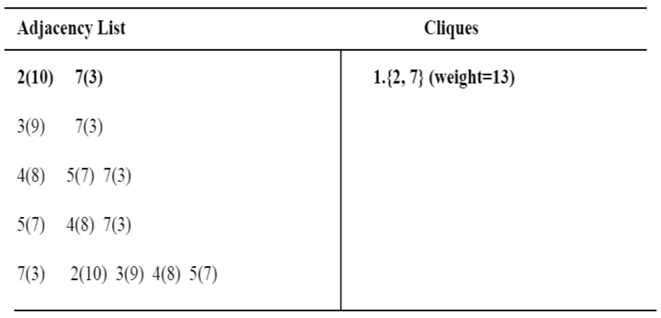


Fig. 5. Vertex weighted graph after deletion of vertex labeled 1 and 6



Adjacency List	Cliques
2(10) 7(3)	**1.{2, 7} (weight=13)**
3(9) 7(3)	
4(8) 5(7) 7(3)	
5(7) 4(8) 7(3)	
7(3) 2(10) 3(9) 4(8) 5(7)	

Fig. 6. Adjacency list of the graph in Figure.5 and possible cliques

Clique (C_2) = (10, 3).

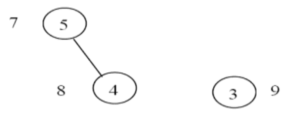
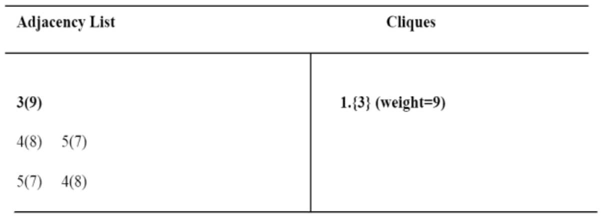


Fig. 7. Vertex weighted graph after deletion of vertex labeled 2 and 7



Adjacency List	Cliques
3(9)	**1.{3} (weight=9)**
4(8) 5(7)	
5(7) 4(8)	

Fig. 8. Adjacency list of the graph in Figure.7 and possible clique

Clique (C_3) = (9).

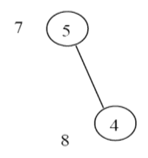


Fig. 9. Vertex weighted graph after deletion of vertex labeled 3

Adjacency List	Cliques
4(8) 5(7)	**1.{4, 5} (weight=15)**
5(7) 4(8)	

Fig. 10. Adjacency list of the graph in Figure.9 and possible clique

Clique (C_4) = (8, 7).

So, total number of bins C_1, C_2, C_3, C_4 = 4.

Selection Criteria 2:

Total average weight of the vertices T_a = (11+ 10+ 9+ 8+ 7+ 4+ 3)/7 =7.43.

Adjacency List	Cliques
1(11) 6(4) 7(3)	**1.{1, 6} (Average weight = (11+4)/2 = 7.5 > T_a and weight=15)**
2(10) 6(4) 7(3)	2.{1, 7} (Average weight= (11+3)/2 = 7 <T_a and weight=14)
3(9) 6(4) 7(3)	
4(8) 5(7) 6(4) 7(3)	
5(7) 4(8) 6(4) 7(3)	
6(4) 1(11) 2(10) 3(9) 4(8) 5(7) 7(3)	
7(3) 1(11) 2(10) 3(9) 4(8) 5(7) 6(4)	

Fig. 11. Adjacency list of the graph in Figure.3 and possible cliques

Clique (C_1) = (11, 4).

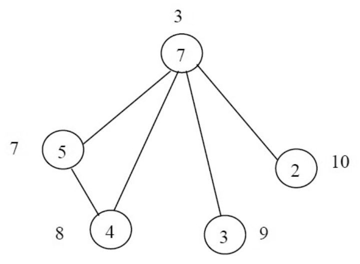


Fig. 12. Vertex weighted graph after deletion of vertex labeled 1 and 6

Adjacency List	Cliques
2(10) 7(3)	**1.{2, 7} (Average weight=(10+3)/2 =6.5 < T_a and weight=13)**
3(9) 7(3)	
4(8) 5(7) 7(3)	
5(7) 4(8) 7(3)	
7(3) 2(10) 3(9) 4(8) 5(7)	

Fig. 13. Adjacency list of the graph in Figure.12 and possible clique

Clique (C_2) = (10, 3).

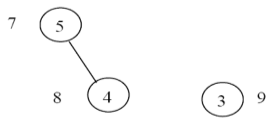


Fig. 14. Vertex weighted graph after deletion of vertex labeled 2 and 7

Adjacency List	Cliques
3(9)	**1.{3} (Average weight 9 > T_a and weight=9)**
4(8) 5(7)	
5(7) 4(8)	

Fig. 15. Adjacency list of the graph in Figure.14 and possible clique

Clique (C_3) = (9).

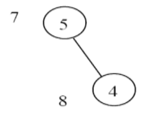
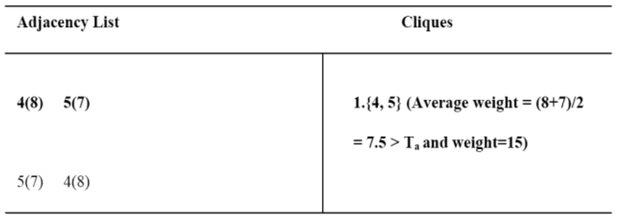


Fig. 16. Vertex weighted graph after deletion of vertex labeled 3



Adjacency List	Cliques
4(8) 5(7)	1.{4, 5} (Average weight = (8+7)/2 = 7.5 > T_a and weight=15)
5(7) 4(8)	

Fig. 17. Adjacency list of the graph in Figure.16 and possible clique

Clique (C_4) = (8, 7).

Here also total number of bins= 4.

Case 3: Input sequence: $C/2 > x_{s+1} \geq x_{s+2} \geq \ldots \geq x_n$

In this case the graph can be viewed as a clique, which is also a Chordal graph.

Here also the number of bin is the number of minimal clique partition with total weight of each clique $\leq$ C.

Example 4.10: W= {10, 9, 9, 9, 8, 8, 7, 7} and C=24. Optimal number of Bins = 3.

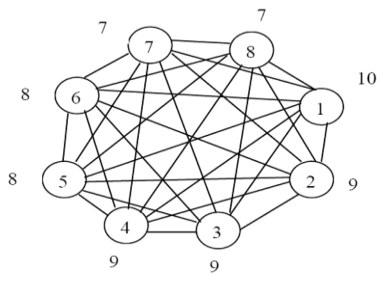


Fig. 18. Vertex weighted graph for case 3

Applying selection criteria 1 (A1) we get the number of bins= 4, i.e. clique (C_1) = (10, 7, 7), clique (C_2) = (9, 9), clique (C_3) = (9, 8) and clique (C_4) = (8), but with selection criteria 2 (A2), the number of bins= 3, i.e. clique (C_1) = (10, 9), clique (C_2) = (9, 8, 7), clique (C_3) = (9, 8, 7) which is improved than former.

V. Computational Results

The proposed algorithm was coded in C, compiled using Borland C++ 5.0 compiler in Win32 mode and in Intel® Atom^{TM} 1.60 Hz Processor with 1.0 GB DDR2 RAM.

The algorithms were tested on six classes of benchmark problem instances, all of which can be downloaded from the web page of EURO Special Interest Group on Cutting and Packing (ESICUP) (http://paginas.fe.up.pt/~esicup/). The propose algorithm with heuristic criteria 1 is named as A1 and with heuristic criteria 2 is named as A2.

The first two, the u class and t class, were developed by [21] and named instance ‘a’ in table I. The u class has item weights drawn from an integer uniform distribution on (20, 100) and bin capacity c= 150. There are four sets in this class, namely u_120, u_250, u_500 and u_1000; each consisting of 20 instances with n= 120, 250, 500 and 1000 items, respectively. The t class has item weights drawn from a uniform distribution on (25, 50) and c= 100. Item weights in this class are real numbers. There are also four sets in this class, namely t_60, t_120, t_249 and t_501; each consisting of 20 instances with n= 60, 120, 249 and 501 items, respectively. The t class is considered difficult, because in an optimal solution of each instance, each bin contains 3 items with zero slack (hence the name ‘triplets class’). All problem instances in both the u and t classes have been solved to optimality with the exact algorithm of [22]. It can be seen from table I that, proposed A1 and A2 finds the solution better than FFD heuristic and A1 is giving better solution than A2.

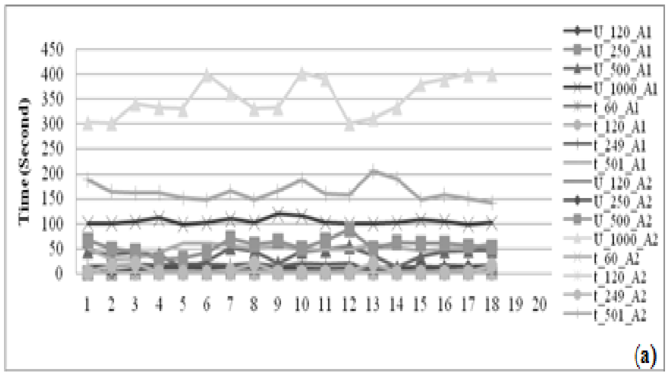
A third class of benchmark problem instances, developed by [23], contains two sets, was_1 and was_2 and named instance ‘b’ in table I. Each set has 100 instances with c= 1000 and item weights from (150,200). Was_1 has n= 100 items in each instance, while was_2 has n= 120 items. For all instances in this class, optimal solutions are known. Solution produced by the proposed A1 and A2 are better than FFD heuristic but A2 has better solution than all other heuristics in table I.

A fourth class of benchmark problem instances, developed by [24], is called gau_1 and contains 17 problem instances with c= 10,000 and various values of n and item weights. It is named instance ‘c’ in table I. For all instances the optimality gap is one bin. Solutions produced by proposed A1 and A2 are same and are better than FFD and B2F heuristics.

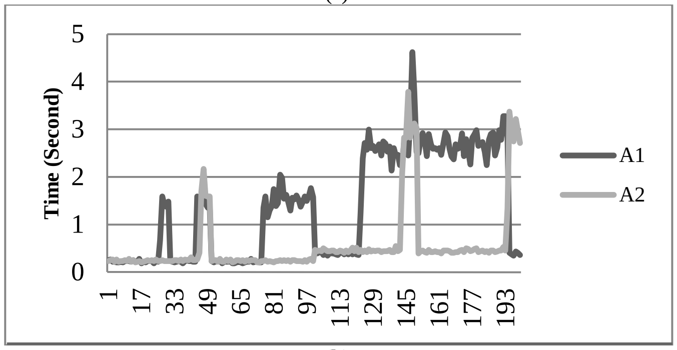
Next, the test on the data set of difficult problem instances has been performed, called hard28, used for example by [25] and named instance ‘d’ in table I. This set has 28 instances with n∈{160,180, 200}, c= 1000, and items weights drawn from (1,800). The simplest heuristic, FFD, finds optimal solutions for five instances and solutions worse than optimal by one bin for all the remaining instances. None of the other heuristics including A1, is able to improve these solutions. In fact, B2F, MBS, MBS´, A1 find worse solutions for some instances. But proposed A2 finds the same solutions as FFD.

A sixth class of benchmark problem instances, developed by [26], consists of set_1, set_2, and set_3 and named instance ‘e_1’, ‘e_2’, ‘e_3’ in table I respectively. Set_1 has 720 instances with c= 100, 120, 150, n= 50, 100, 200, 500, and item weights drawn from an integer uniform distribution on (1,100), (20, 100), and (30, 100). Set_2 has 480 instances with c= 1000, n= 50, 100, 200, 500 and item weights such that each bin has on average 3 to 9 items. Set_3 has 10 instances with c= 100,000, n= 200, and item weights drawn from a uniform distribution on (20,000, 35,000). Optimal solutions for 1184 instances in this class have been found in [26]. For the remaining 26 instances, optimal solutions were found by [27]. From Table I it can be seen that solution produced by proposed A2 is better than other heuristics.

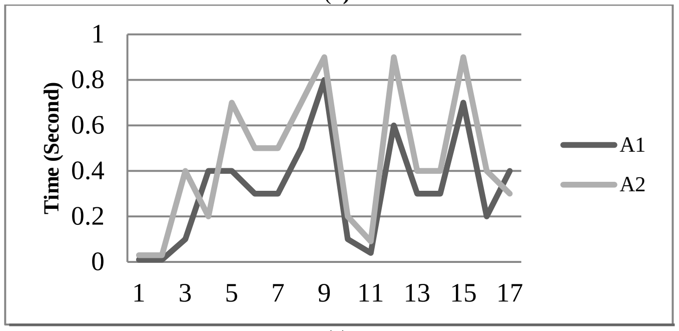
Additionally, in Figure.19 time comparison between two proposed heuristics A1 and A2 for above instances are shown.



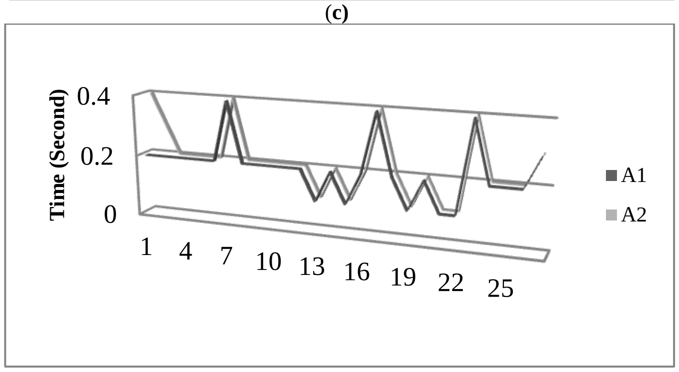
(a)



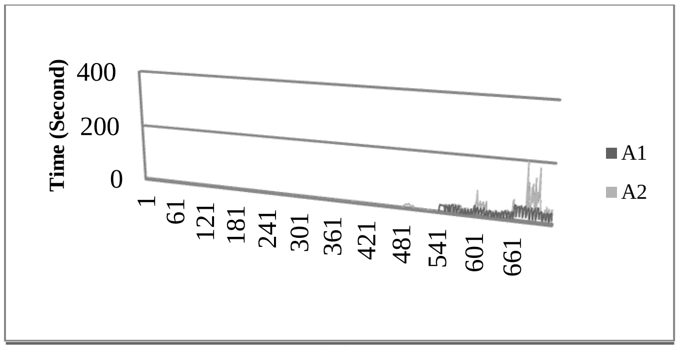
(b)



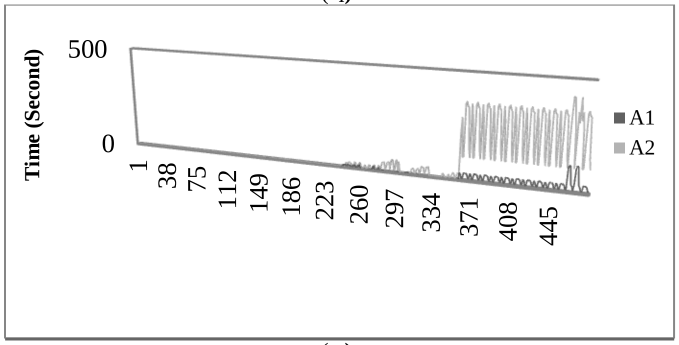
(c)



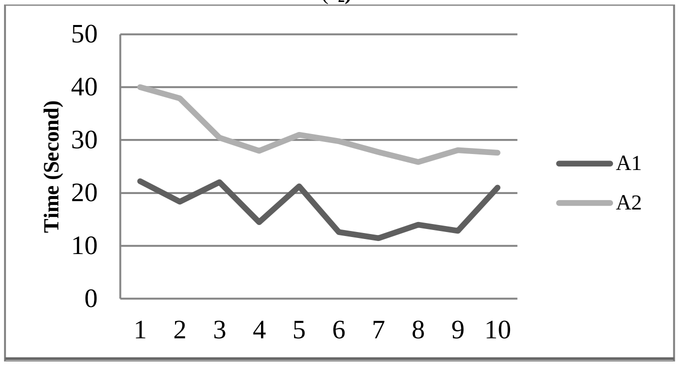
(d)



(e_1)



(e_2)



(e_3)

Fig. 19. Time Comparison between two proposed heuristics A1 and A2 for instances a, b, c, d, e1, e2, e3 respectively. X-axes represent time and Y-axes represent number of instances

VI. Conclusions

It is the very beginning stage of the solution of the problem to store large amount of data in a minimum storage space. In this paper, mainly one dimensional Bin Packing Problem has been treated by a graph algorithm. The proposed algorithm is based on two heuristics; one is depending on maximum total weight criteria of the vertices not exceeding the bin capacity and second is based on maximum average weight criteria of the vertices also not exceeding the capacity of the bin. The algorithm takes polynomial time and finds near-optimal solution which is shown in computational results. It can also be seen that, no algorithm is better for all the instances, the algorithm with second criteria (A2) outperforms other heuristics for one benchmark instance and in other cases, solution with two heuristics (A1 and A2) deviates small from the best known solutions and solutions produced by the four heuristics.

VII. Future Work

Several further research scopes can be outlined as follows. Firstly, along with the volume of the data its dimension is also increasing. To tackle this problem good and efficient algorithms are needed for storing high dimensional data. In this case, multidimensional network graph concept can be used or vertex weighted graph for each dimension can be created and the proposed algorithm can be applied to the graph produced from the intersection of graphs of each dimension. But it needs further investigations. Secondly, online partition problem or semi-online partition problem [28] can be applied to generate cliques with constrained weight of the vertex weighted graph. This concept can be used to tackle online or semi-online Bin Packing Problem but needs further detailed study. Third, running time of the algorithm can be improved from O ($|V|^2$) to O (|V|log|V|) by using the appropriate data structure namely red black tree [29] which is under the investigation of the author and last but not the least, though this paper contains a study of the bound of maximum number of bins needed by the proposed algorithm, the proof of the upper bound involves an extremely detailed case analysis which can be investigated in future.

Acknowledgements

The authors would like to thank University Of Calcutta, West Bengal, India, Department of Science & Technology (DST- under Grant [**No. DST/ INSPIRE Fellowship/2013/943**]), 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] G. Michael R., and D. S. Johnson, “Approximation algorithms for bin packing problems: A survey,” Analysis and design of algorithms in combinatorial optimization 266 (1981): 147-172.

[2] G. Michael R., R. L. Graham, and J. D. Ullman, “Worst-case analysis of memory allocation algorithms,” In Proceedings of the fourth annual ACM symposium on Theory of computing, pp. 143-150. ACM, 1972.

[3] G. Michael R., and D. S. Johnson, Computers and intractability. Vol. 29. New York: wh freeman, 2002.

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

[5] R. Gupta, S. K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti, “A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin item incompatibility constraints,” In Services Computing, 2008. SCC'08. IEEE International Conference on, vol. 2, pp. 39-46. IEEE, 2008.

[6] D. Bein, W. Bein, and S. Venigella, “Cloud storage and online bin packing,” In Intelligent Distributed Computing V, pp. 63-68. Springer Berlin Heidelberg, 2011.

[7] D. Sensarma, and S. S. Sarma, “A Unified Framework for Security and Storage of Information,” Internationa Journal of Advance Engineering and Research Development, Vol.2, No.1, 2015.

[8] D. B. West, Introduction to graph theory, Vol. 2. Upper Saddle River: Prentice hall, 2001.

[9] N. Deo, Graph theory with applications to engineering and computer science. (1994).

[10] M. C. Golumbic, Algorithmic graph theory and perfect graphs, Vol. 57, Elsevier, 2004.

[11] A. Shapira, R. Yuster, and U. Zwick., “All-pairs bottleneck paths in vertex weighted graphs,” In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 978-985. Society for Industrial and Applied Mathematics, 2007.

[12] J. Li, M. Chen, J. Li, and W. Li., “Minimum clique partition problem with constrained weight for interval graphs,” In Computing and Combinatorics, pp. 459-468. Springer Berlin Heidelberg, 2006.

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

[14] K. Fleszar, and C. Charalambous, “Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem,” European Journal of Operational Research 210, no. 2 (2011): 176-184.

[15] K. Jansen, and S. Öhring., “Approximation algorithms for time constrained scheduling,” Information and Computation 132, no. 2 (1997): 85-108.

[16] L. Epstein, and A. Levin, “On bin packing with conflicts,” In Approximation and Online Algorithms, pp. 160-173. Springer Berlin Heidelberg, 2006.

[17] B. McCloskey, and A. J. Shankar, “Approaches to bin packing with clique-graph conflicts,” Computer Science Division, University of California, 2005.

[18] B. Codenotti, G. D. Marco, M. Leoncini, M. Montangero, and M. Santini, “Approximation algorithms for a hierarchically structured bin packing problem,” Information processing letters 89, no. 5 (2004): 215-221.

[19] C. Bujtás, G. Dósa, C. Imreh, J. Nagy-GYörgy, and Z. Tuza., “The graph-bin packing problem,” International Journal of Foundations of Computer Science 22, no. 08 (2011): 1971-1993.

[20] Klein, P. Nathan, “Parallel algorithms for chordal graphs,” Brown University, Department of Computer Science, 1991.

[21] E. Falkenauer, “A hybrid grouping genetic algorithm for bin packing,” Journal of heuristics 2, no. 1 (1996): 5-30.

[22] JV. De Carvalho, “LP models for bin packing and cutting stock problems,” European Journal of Operational Research. 2002 Sep 1;141(2):253-73.

[23] P. Schwerin, and G. Wäscher, “The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP,” International Transactions in Operational Research 4, no. 5-6 (1997): 377-389.

[24] G. Wäscher, and T. Gau, “Heuristics for the integer one-dimensional cutting stock problem: A computational study,” Operations-Research-Spektrum 18, no. 3 (1996): 131-144.

[25] G. Belov, and G. Scheithauer, “A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting,” European journal of operational research 171, no. 1 (2006): 85-106.

[26] A. Scholl, R. Klein, and C. Jürgens, “Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem,” Computers & Operations Research 24, no. 7 (1997): 627-645.

[27] A. C. Alvim, C. C. Ribeiro, F. Glover, and D. J. Aloise, “A hybrid improvement heuristic for the one-dimensional bin packing problem,” Journal of Heuristics 10, no. 2 (2004): 205-229.

[28] S. Albers, and H. Matthias, “Semi-online scheduling revisited,” Theoretical Computer Science 443 (2012): 1-9.

[29] T. H. Cormen, C. E. Leiserson, and R. R. Rivest, Introduction to Algorithms. MIT Press (1990).

TABLE I. COMPARISON BETWEEN PROPOSED HEURISTICS A1, A2 AND FOUR EXISTING HEURISTICS WITH 1615 INSTANCES

	(a)			**(b)**			**(c)**			**(d)**			**(e1)**			**(e2)**			**(e3)**		
	Best	**Avg. Dev.**	**Max Dev.**	**Best**	**Avg. Dev.**	**Max Dev.**	**Best**	**Avg. Dev.**	**Max Dev.**	**Best**	**Avg. Dev.**	**Max Dev.**	**Best**	**Avg. Dev.**	**Max Dev.**	**Best**	**Avg. Dev.**	**Max Dev.**	**Best**	**Avg. Dev.**	**Max Dev.**
FFD	6	6.73	24	0	1.04	2	3	0.82	1	5	0.82	1	546	0.39	5	236	1.56	21	0	3.4	4
B2F	41	1.23	4	64	0.68	1	4	0.76	1	4	1.07	2	639	0.13	3	363	0.44	9	7	0.3	1
MBS	40	0.98	3	33	0.84	1	13	0.24	1	1	3.32	10	252	1.47	3	387	0.27	5	0	2.6	3
MBS´	41	1.24	7	36	0.82	1	13	0.24	1	2	1.39	3	633	0.14	3	381	0.34	6	0	3.3	4
A1	31	1.45	7	33	0.84	1	8	0.53	1	2	1.43	3	624	0.16	3	367	0.40	6	0	3.4	4
A2	23	1.28	4	75	0.63	1	8	0.53	1	5	0.82	1	631	0.15	3	381	0.34	6	4	0.6	1