Fall 2008

A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach

Zhou-Jing Wang

Kevin W. Li Dr.
University of Windsor

Jason K. Levy

A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach

Zhoujing Wang [a], Kevin W. Li [b*], and Jason K. Levy [c]

[a]*Department of Automation, Xiamen University, Xiamen, Fujian 361005, China*
[b]*Odette School of Business, University of Windsor, Windsor, Ontario N9B 3P4, Canada*
[c]*Huxley College of the Environment, Western Washington University, Bellingham, Washington 98225, USA*

Abstract

Increasing fuel costs, post-911 security concerns, and economic globalization provide a strong incentive for container carriers to use available container space more efficiently, thereby minimizing the number of container trips and reducing socio-economic vulnerability. A heuristic algorithm based on a tertiary tree model is proposed to handle the container loading problem (CLP) with weakly heterogeneous boxes. A dynamic space decomposition method based on the tertiary tree structure is developed to partition the remaining container space after a block of homogeneous rectangular boxes is loaded into a container. This decomposition approach, together with an optimal-fitting sequencing and an inner-right-corner-occupying placement rule, permits a holistic loading strategy to pack a container. Comparative studies with existing algorithms and an illustrative example demonstrate the efficiency of this algorithm.

Keywords: Heuristics, container loading problem, dynamic space decomposition, holistic loading, tertiary tree

1. Introduction

The accelerating pace of economic globalization has led to the rapid growth of a vast container shipment transportation network, encompassing sea, land, rail, and air linkages. In the post-911 era, increasing emphasis has been placed on the vulnerability of this intermodal container shipping transportation conduit to a possible terrorist attack. Accordingly, the movement of containers across all modes of transportation are now susceptible to significant delays due to enhanced security requirements such as the maritime diversion, inspection, or interdiction of containers suspected of carrying

[*] Corresponding author. Tel: +1 519 2533000 ext. 3456; fax: +1 519 9737073. Email: kwli@uwindsor.ca

smuggled arms or concealed Chemical, Biological, Radiological or Nuclear Weapons (CBRN).In addition to possible delays in shipping the containers, rising fuel costs now provide a strong incentive for container carriers to maximize available container space, thereby minimizing the number of required trips across the global container transportation system. Accordingly, utilizing container space in an efficient manner has assumed increasing economic significance [Bortfelt et al., 2003, Mack et al., 2004], especially in light of growing competition from container carriers in Asian nations.

The container loading problem (CLP) typically involves arranging rectangular cardboard cargo boxes in a container so that the volume usage, subject to other practical constraints, of the container is maximized [Moura and Oliveira, 2005]. A number of CLP categories have been considered [Gehring and Bortfeldt, 1997]. One approach is based on container quantity (single or multiple containers). Completely loading a large quantity of goods may necessitate multiple containers [Takahara, 2005], while a single container is typically involved when some goods can be left behind, [Dyckhoff, 1990]. Another approach differentiates CLPs according to the mix of box types to be loaded. In this respect, CLPs vary from a completely homogeneous problem, where boxes have identical dimensions and orientations, to the extreme heterogeneous case, where many different sized items are present. The CLP between these two extremes (with relatively few box types) is often referred to as the weakly heterogeneous case [Bischoff and Ratcliff, 1995].

The CLP is notoriously NP hard [Scheithauer, 1992], meaning that the solution is too complex to be solved exactly in polynomial time. Accordingly, heuristics are often the only viable option [George and Robinson, 1980; Mack et al., 2004; Ngoi et al. 1994]. Recent years have witnessed significant advances in algorithm development for handling CLPs [Eley, 2002; Pisinger, 2002], and increased attention is now focused on intelligent algorithms, such as genetic algorithms [Gehring and Bortfeldt, 1997; Bortfeldt and Gehring, 2001], simulated annealing [Jin et al., 2004], and tabu search algorithms [Bortfeldt et al, 2003]. Furthermore, researchers have begun paying more attention to additional practical constraints. For instance, Davies and Bischoff [1999], Eley [2002], and Gehring and Bortfeldt [1997] take into account the weight distribution of cargo within a container. Bischoff [2006] examines the impact of varying load bearing strength. Several studies consider loading stability, including Bortfeldt and Gehring [2001],

Bortfeldt et al. [2003], and Terno et al. [2000]. Other container factors have also been considered, such as orientation constraints [Gehring and Bortfeldt, 2002] and the grouping of boxes [Bischoff and Ratcliff, 1995; Takahara, 2005].

As pointed out by Bischoff and Ratcliff [1995], many existing CLP algorithms are based on a "wall-building" approach: layers are formed either vertically across the width of the container [George and Robinson, 1980] or horizontally from bottom to top [Loh and Nee 1992]. This "wall-building" or "layer-building" method proves to be effective for accommodating other practical container loading constraints such as limited bearing strength [Bischoff, 2006] and weight distributions [Davies and Bischoff, 1999] as well as handling pallet loading problems [Bischoff et al. 1995]. Generally speaking, the "wall-building" technique treats the remaining space after a wall or layer is loaded as a reduced-sized container with some procedures to fill the trapped or unused space within a layer or across layers. Therefore, the remaining space is essentially pre-determined in a static manner. However, for approaches that do not follow the "wall-building" philosophy [Ngoi et al., 1994], characterizing the remaining space may be complicated, depending on the local arrangement of recently-loaded boxes [Bortfeldt et al., 2003]. As our approach falls within this latter category, it is necessary to consider the decomposition and representation of remaining space as different decompositions lead to distinct searching paths and solutions. One of the key novelties in this algorithm is to develop an explicit procedure for decomposing the remaining container space in a dynamic fashion.

The heuristic algorithm herein presented tackles the CLP with weakly heterogeneous items. This approach employs a tertiary tree structure to represent the container space and develops a dynamic decomposition method to partition the unfilled space after a block of identical items is loaded. This dynamic decomposition, assisted by an optimal-fitting sequencing and an inner-right-corner-occupying rule, is designed to search for an optimal partition of the remaining space for next-step packing. When the cargo is loaded, identical boxes are arranged as a large rectangular block to allow for a holistic loading into the container. An earlier version of this paper was presented at a conference and published in the proceedings [Wang et al., 2006]. This article significantly expands the conference paper by more clearly describing the CLP algorithm, providing new examples

about block formation, adding the algorithm pseudo-code, and discussing more details on the holistic loading scheme.

The rest of the paper is structured as follows. Section 2 briefly describes the problem, followed by an introduction to the tertiary tree model in Section 3. The heuristic algorithm is presented in Section 4. Comparative studies with other algorithms and an illustrative example are given in Sections 5 and 6. Some concluding remarks are provided in Section 7.

2. The Problem Statement

This heuristic is designed to load the cargo packed in heterogeneous rectangular cardboard boxes into a single standard rectangular container. The dimensions of the rectangular container are given as L (length), W (width), and H (height). The boxes to-be-loaded are categorized into K groups as per their three dimensions, and each group k contains N_k cuboids with a length, width, and height of l_k, w_k, and $h_k (k = 1, 2, ..., K)$, respectively. The objective of the algorithm is to determine a loading scheme that will maximize the space usage of the container.

$$\eta = \max\left\{\frac{\sum (l_k \cdot w_k \cdot h_k) N_k}{L \cdot W \cdot H}\right\}$$

This algorithm takes a holistic approach to the CLP: homogeneous boxes will first be grouped into a larger block with a rectangular shape and then simultaneously loaded into the container. When the algorithm converges, a detailed loading scheme will be generated to specify the optimal sequence and placement of each group.

Similar to the approach of Ngoi et al. [1994], without a loss of generality, this algorithm assumes that

1) The cargo is packed in rectangular cardboard boxes only. Other item shapes are not considered.
2) All boxes are sufficiently strong and firm to bear other boxes placed on top of them. The limited bearing strength constraint [Bischoff, 2006] is excluded.
3) Boxes are weakly heterogeneous, i.e., there are relatively few sizes and each size has many boxes. This assumption permits a holistic loading approach.
4) All boxes are shipped to the same destination so that prioritizing the box loading

sequence is not required.

5) There is no restriction on the orientation of the box and all boxes may be rotated about *x*-, *y*-, and/or *z*-axis if necessary. For instance, in the case of loading many boxes of shoes or cigarettes into a container, no problems arise from interchanging the three box dimensions.

These assumptions enhance CLP tractability and apply to many practical container loading situations. As this heuristic employs a tertiary tree structure to describe the container space, the basics of the tertiary tree model are furnished next.

3. Basics of the Tertiary Tree Model

Trees are one of the most important computer science data structures for representing nonlinear information and facilitating software development [Knuth, 1997]. A tree is often employed to represent hierarchical information, which is defined recursively as a non-empty finite set of nodes,

$$T = \{r\} \cup T_1 \cup T_2 \cup ... \cup T_m$$

where node *r* is designated to be the root of the tree, and the other nodes are partitioned into subsets $T_1, T_2, ..., T_m$, $m \geq 0$ and each of these subsets is also a tree [Knuth, 1997]. Tertiary trees are a special tree structure where either the node set is empty or the set consists of a root, *r*, and exactly three distinct tertiary trees. In a tertiary tree, a node has either none or three nodes below it, often referred to as child nodes.

Consider the situation in which a box or a block of homogeneous boxes forming a large rectangular parallelepiped is placed in the inner right corner of a container as shown in Fig. 1. When the first box or block G(1) is loaded into the container and placed in the corner (labeled as Vol), the remaining space is partitioned into three subspaces, Vol(1), Vol(2), and Vol(3). Similarly, if the next box or block G(2) is put in Vol(1), the space Vol(1) will be partitioned into three parts accordingly. This partition process continues until all spaces are occupied (it is possible that a space becomes too small to accommodate any available box, in this case, the unused space is discarded and treated as if it were occupied) or all boxes are loaded into the container.

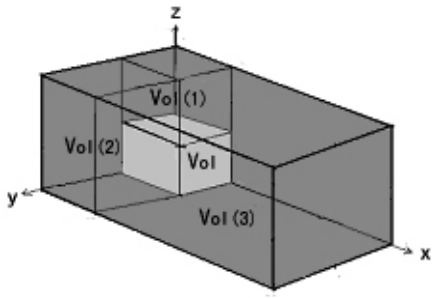

Fig. 1 Partition of the container space after a box/block is loaded.

Given this observation, it is natural to employ a tertiary tree structure to represent the progressive loading of the cargo [He et al., 2000]. Let each box/block be a node in a tertiary tree, the three partitioned subspaces due to the loading of this box/block can be treated as the child nodes of this node. Given that the container and each box/block have certain volumes, the recursive partition process stops after a finite number of steps. Therefore, the resulting nodes are finite and the overall structure is indeed a tertiary tree. An illustration of the tertiary tree is given in Fig. 2.

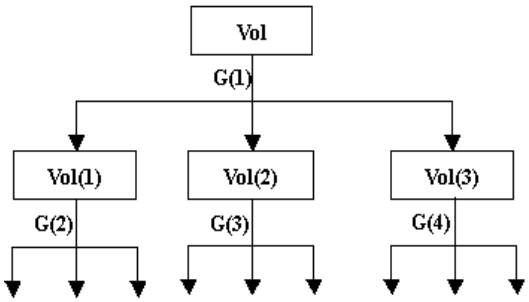

Fig. 2. A tertiary tree model for the space partition.

Next, our heuristic algorithm is formulated based on this tertiary tree structure.

4. A Heuristic Algorithm

In contrast to other heuristic algorithms in the current literature [Davies and Bischoff, 1999], where the container space is usually partitioned in a pre-determined manner, our proposed algorithm decomposes the container space in a dynamic fashion based on the tertiary tree model as shown in Section 3.

4.1 Dynamic space decomposition

The process starts from the initial full space of the container, which is set as the

current space and corresponds to the root of the tertiary tree. According to a unique *optimal-fitting* sequencing rule proposed in this paper, a group of homogeneous boxes that is to be loaded and fits the current space best will be chosen first (see the *optimal-fitting* sequencing rule below for details). A corner-occupying placement rule is adopted to put the cargo in the inner-right corner of the current space. After a block of boxes is loaded, the remaining empty volume in the current space can be divided into three mutually exclusive subspaces, corresponding to the left (*L*), middle (*M*), and right (*R*) child nodes of the root in the tertiary tree. Each of the three subspaces (child nodes) is then set to be the current space sequentially from the left to the middle and then to the right node, and the same decomposition procedure is repeated for each new current space until no unused space is available in the container, all unused spaces are too small to fit any box, or all boxes are loaded.

One way to partition the remaining space is as shown in Fig. 1. However, there exist several other possible partitions. For instance, in Fig. 3, the initial current space is uniquely determined by its two space diagonal vertices, *A* and *N*, thus denoted by $S(A,N)$. Using the similar space notation, the partition given in Fig. 1 corresponds to $\{S(A,I),\ S(B,M),\ S(C,N)\}$. Other five possible partitions are given as { *S*(*A*, *K*), *S*(*F*, *M*), *S*(*C*, *N*)}, { *S*(*A*, *J*), *S*(*F*, *M*), *S*(*E*, *N*)}, { *S*(*A*, *J*), *S*(*F*, *N*), *S*(*E*, *G*)}, { *S*(*A*, *O*), *S*(*B*, *N*), *S*(*E*, *G*)}, and {*S*(*A*, *I*), *S*(*B*, *N*), *S*(*C*, *G*)} (See Fig. 3 for details).

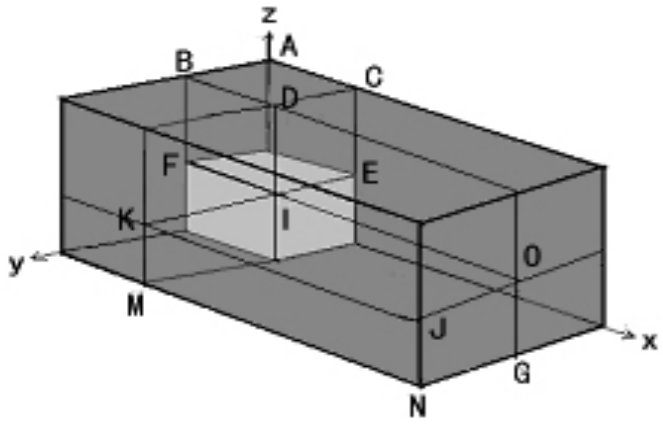

Fig. 3. Different decomposition scenarios for the remaining empty space.

It is obvious that different space decomposition scenarios lead to distinct searching

paths, which directly affect the efficiency and quality of the solution. The essence of the proposed dynamic space decomposition approach is to minimize the size of most likely unusable space(s) and maximize the size of the most likely usable partitioned space(s). When the cargo is placed in the current space, it takes the following sequence: first, from the bottom to the top (along the *z*-axis in Fig. 3); second, from the right to left (along the *y*-axis in Fig. 3); and last, from the back to the front (along the *x*-axis in Fig. 3). For the resulting three partitioned spaces, the order to set each of them as the new current space follows the same mechanism and this sequential information is encoded in the tertiary tree where the left child node is set first and, then the middle, and last, the right.

To facilitate the formulation of the space decomposition strategy, let L, W, and H denote the length, width, and height of the current space, respectively, and l_0, w_0, and h_0 represent the length (along x), width (along y), and height (along z) of the currently loaded cargo, which is either a single box or a rectangular block of homogeneous boxes. Furthermore, for all boxes to be loaded, assume that $B_{\min}$ is the minimum of any of the three dimensions and $V_{\min}$ is the minimum volume. Given the aforesaid notation, the width of the space $S(B,M)$ is $W_m = W - w_0$ with a volume of $V_m = l_0(W - w_0)H$. Similarly, the space $S(C,N)$ has a length of $L_r = L - l_0$ and a volume of $V_r = (L - l_0)WH$. Now, our dynamic space decomposition can be described as the following four rules.

1) If ($W_m \geq B_{\min}$ and $V_m \geq V_{\min}$) and ($L_r \geq B_{\min}$ and $V_r \geq V_{\min}$), the remaining empty space is decomposed into {*S*(*A*, *I*), *S*(*B*, *M*), *S*(*C*, *N*)};
2) If ($W_m \geq B_{\min}$ and $V_m \geq V_{\min}$) and ($L_r < B_{\min}$ or $V_r < V_{\min}$), the remaining empty space is decomposed into {*S*(*A*, *O*), *S*(*B*, *N*), *S*(*E*, *G*)};
3) If ($W_m < B_{\min}$ or $V_m < V_{\min}$) and ($L_r \geq B_{\min}$ and $V_r \geq V_{\min}$), the remaining empty space is decomposed into {*S*(*A*, *K*), *S*(*F*, *M*), *S*(*C*, *N*)};
4) Otherwise, the remaining space is decomposed into {*S*(*A*, *J*), *S*(*F*, *M*), *S*(*E*, *N*)};

After the first block of boxes is loaded as shown in Fig. 3, rule 1) indicates that space *S*(*C*, *N*) is most likely usable given the two conditions are satisfied, so this partition provides the maximum possible volume for the space to the front (along the *x*-axis). As per the order of loading the cargo into the container, space *S*(*A*, *I*) is most likely unusable and, hence, is allocated the minimum possible volume to the top (along the *z*-axis). While

space $S(B, M)$ may or may not be usable, depending on the length l_0 as the other two dimensions, W_m and H, are both greater than $B_{\min}$ and $V_m \geq V_{\min}$. By comparing this partition with the other five possibilities, we can verify that this is the best decomposition in terms of maximizing the volume usage of the current space. Similarly, for the other three rules, each provides the best available partition of the unfilled space given the respective conditions.

4.2 Optimal-fitting sequencing

After the unfilled space is decomposed, the next question is to find out the loading sequence of remaining boxes. A variety of sequencing rules has been proposed [George and Robinson, 1980; Ngoi et al., 1994]. The rank of boxes may be determined in a descending order according to their volumes, base areas, the longest of their three dimensions, or aggregate volumes of unloaded homogeneous boxes. This paper proposes a unique *optimal-fitting* rule to rank boxes, which integrates the aggregate volume of homogeneous boxes and the orientation of the boxes together to determine a group of boxes that make the best use of the current space.

To describe the *optimal-fitting* sequencing rule, assume that the cargo to be loaded is packed into boxes with K distinct specifications, $\{1, 2, ..., k, ..., K\}$, and each type k has N_k identical boxes and the corresponding length, width, and height of each box of type k are $l_k, w_k,$ and $h_k,$ respectively, k = 1, 2, …, K. When a box is placed in a container, there exist six distinct orientations if the "front" and "back", "top" and "bottom", and "left" and "right" positions are not differentiated. Using face notation W-H for the rectangle face consisting of the two sides with the width and height and L-W for the face with the length and width sides (other faces can be described similarly), the six orientations can be expressed as: 1. face W-H towards the container door and face L-W towards the top; 2. face L-W towards the door and face W-H towards the top; 3. face L-H towards the door and face L-W towards the top; 4. face L-W towards the door and face L-H towards the top; 5. face W-H towards the door and face L-H towards the top; 6. L-H towards the door and W-H towards the top. For the current space (initially, the full container space), the optimal-fitting sequencing rule selects the cargo according to the following procedure.

1) In the current space, for each specification k with the quantity N_k, calculate the

maximum number of boxes that can be loaded as per each aforesaid orientation i, denoted by N_{ik}, $i = 1, 2, \ldots, 6$; $k = 1, 2, \ldots, K$. It is obvious that $0 \le N_{ik} \le N_k$ and the corresponding aggregate volume for each group under each orientation is given as $v(i,k) = (l_k w_k h_k) N_{ik}, \quad i = 1,2,...,6; \; k = 1,2,...,K$. In total, $6K$ possible aggregate volumes are obtained. We call the i^* and k^* that maximize $v(i,k)$ as the optimal-fitting orientation and specification, respectively. If such i^* and k^* are unique, then $N_{i^*k^*}$ boxes of k^* will be chosen and grouped together with an orientation of i^* for possibly loading into the current space. Otherwise, go to 2).

2) For all i^* and k^*'s that maximize $v(i,k)$, calculate the aggregate length along the x-axis, width along the y-axis, and height along the z-axis (see Fig. 3) of the large block by stacking the identical boxes together. The block with the minimal length (x) is selected first, followed by that with the minimal width (y), and that with the minimal height (z). In the case that all the three dimensions are the same for all optimal blocks, it does not matter which block is chosen first as they are identical in terms of maximizing the volume usage. Therefore, a block will be randomly selected.

This sequencing rule, as a matter of fact, dictates the dynamic space decomposition strategy. For instance, after the first group of boxes is loaded as shown in Fig. 3, the second rule here ascertains that the usable possibility of the three subspaces, to the top (z), to the side (y), and to the front (x), is in an ascending order if the conditions of decomposition rule 1) are satisfied. In this case, the decomposition {$S(A, I)$, $S(B, M)$, $S(C, N)$} maximizes the subspace to the front of the current cargo and minimizes the subspace to the top.

4.3 Holistic loading

As per the optimal-fitting sequencing rule described in Section 4.2, it has been determined which type (k^*) of boxes is to be loaded and what orientation of each box (i^*) is to be placed as well as the maximum number of boxes ($N_{i^*k^*}$) that may be loaded next. The exact number of boxes is to be finalized at the holistic loading stage. At the loading time, homogeneous boxes are first grouped together as a large rectangular parallelepiped and, then, loaded into the current space simultaneously as a block (see Fig.

4 for an illustration).

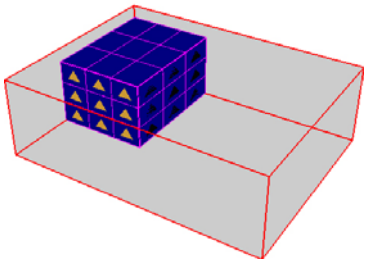

Fig. 4. An illustration of the holistic loading scheme.

Given the optimal choice of type k^* and orientation i^*, at the holistic loading step, the key concern is how to arrange boxes of type k^* as a large cuboid. It is possible that the maximum number of type k^* boxes, $N_{i^*k^*}$, cannot be stacked together to form a regular cuboid as per orientation i^*. Instead, multiple cuboids may be formed. In this case, some boxes may have to be left behind and re-considered at a later stage. For an illustration, see the example after the pseudo code below.

To formulate the holistic loading scheme in pseudo codes, let *SL*, *SW*, *SH* be the three dimensions of the current space along the three axes, *x*, *y*, and *z* as shown in Fig. 3, and each box of type k^* with orientation i^* takes up *BL*, *BW*, and *BH* along the *x*, *y*, and *z* axis, respectively. For notational convenience, define $Q = N_{i^*k^*}$, and let *QX*, *QY* and *QZ* be the numbers of boxes arranged as a large cuboid along the *x*, *y*, and *z* axis, respectively. Next, we seek to find QX^*, QY^*, and QZ^*, such that $QX^* \cdot QY^* \cdot QZ^*$ is closest to Q (or $Q - QX^* \cdot QY^* \cdot QZ^*$ is minimized). Using this notation, the procedure for the holistic loading scheme is given as the following pseudo code.

```
max_temp = 0;
MAX_QX=0;  // The output for the optimal number of boxes along x.
MAX_QY=0;  // The output for the optimal number of boxes along y.
MAX_QZ=0;  // The output for the optimal number of boxes along z.
for QX = 1 to min(Q, int(SL/BL)
   for QY = 1 to min(int(Q/QX), int(SW/BW))
         for QZ=1 to min(int(Q/(QX*QY)),int(SH/BH))
            if (QX*QY*QZ)<=Q then
```

```
            if  (QX*QY*QZ)>max_temp then
                begin
                   max_temp = QX*QY*QZ;
                  MAX_QX=QX;
                  MAX_QY=QY;
                  MAX_QZ=QZ;
                end
        output(MAX_QX,MAX_QY,MAX_QZ)
```

The output of this scheme (MAX_QX = QX^*, MAX_QY = QY^*, MAX_QZ = QZ^*) clearly indicates how the boxes are to be arranged together to form a large cuboid: $QX^* \times QY^* \times QZ^*$ along the three axes, *x*, *y*, and *z*, respectively. For any remaining boxes with the same dimensions, the quantity is updated accordingly.

To illustrate how a large cuboid is formed and when multiple cuboids may arise, an example is furnished here.

Example. Assume that the three dimensions of the current space are $SL \times SW \times SH = 215.5\text{cm} \times 185\text{cm} \times 195.5\text{cm}$ along the *x*, *y*, and *z* axis, respectively, and the optimal-fitting sequencing rule has determined that the type and orientation of boxes to be loaded are $BL \times BW \times BH = 35\text{cm} \times 40\text{cm} \times 45\text{cm}$ along the *x*, *y*, and *z* axis, respectively.

1. $Q = 4$. As min(*Q*, int(*SL/BL*) = min(4, 6) = 4, int(*SW/BW*) = 4, and int(*SH/BH*) = 4, the first optimal solution is achieved at $QX = 1$, $QY = 1$, and $QZ = 4$ during the first outer loop $QX = 1$ and max_temp is set to be 4. When the outer loop continues, another potential optimal solution, $QX = 2$, $QY = 1$, and $QZ = 2$, is obtained. However, the output values, *MAX_QX*, *MAX_QY*, and *MAX_QZ*, will not be changed as *QX*QY*QZ* = 2*1*2 > max_temp = 4 is not satisfied. Similarly, another potential optimal solution, $QX = 2$, $QY = 2$, and $QZ = 1$, will not alter the output values, either. Therefore, the final output optimal solution is $MAX\_QX = 1$, $MAX\_QY = 1$, and $MAX\_QZ = 4$, corresponding to a block of $1 \times 1 \times 4$, one box along *x* (*SL*), one box along *y* (*SW*), and four boxes along *z* (*SH*).
2. $Q = 6$. As min(*Q*, int(*SL/BL*) = min(6, 6) = 6, int(*SW/BW*) = 4, and int(*SH/BH*) = 4, the first maximum is attained at $QX = 1$, $QY = 2$, $QZ = 3$. Five other maxima

are $QX \times QY \times QZ = 1 \times 3 \times 2;\ 2 \times 1 \times 3;\ 2 \times 3 \times 1;\ 3 \times 1 \times 2;\ 3 \times 2 \times 1$. Similar reasoning leads to a final output optimal solution of $1 \times 2 \times 3$, corresponding to a block of one box along *x* (*SL*), two boxes along *y* (*SW*), and three boxes along *z* (*SH*).

3. *Q* = 7. Similarly, min (*Q*, int(*SL/BL*) = min(7, 6) = 6, int(*SW/BW*) = 4, and int(*SH/BH*) = 4, and the first maximum is arrived at $QX \times QY \times QZ = 1 \times 2 \times 3$. The same five other maxima are derived as those in case 2. Therefore, a same block of $1 \times 2 \times 3$ is obtained as the optimal solution to form a large cuboid and one box is left behind for loading at a later step.

In case 3, it is obvious that the current space is large enough to hold all seven boxes. However, due to the holistic loading scheme, these boxes are divided into two blocks. The three cases also demonstrate the key sequencing idea in this algorithm: bottom to top first (case 1) and, then, right to left (case 2 and 3), and, lastly, back to front.

4.4 Representations of the heuristic algorithm

A pseudo code representation of this heuristic is provided below. In this set of codes, a record type, TPackTreeNode, is defined to depict the information for the current space, a linked list, TTree, is employed to represent tertiary trees, and a class, TTreeManager, is introduced to manage tertiary trees.

```
TPackTreeNode=record     // Record the node information
    SL,SW,SH: Double      // Length, width, and height of the current space
    SLC,SWC,SHC: Double      // Coordinates of the current space
    BL,BW,BH:Double      // Length, width, and height of boxes
  end;
TTree=^CPackTreeNode;      // Tertiary tree
CPackTreeNode=record
    Data: TPackTreeNode;   // Root node information
    LChild,MChild,RChild: TTtree;   // L, M, and R child nodes
  end;

TTreeManager=class
  public
    constructor Create;
```

```
        destructor Destroy;   override;
        function CanPack:Boolean;  // Can any box be packed into current space?
        procedure SelBoxes;       // Select boxes as per sequencing rule.
        procedure CalPlace;       // Place the boxes as per placement rule.
        procedure PlaceBox;     // Holistically load boxes into the container as a cuboid.
        procedure UpdateBoxInfo;   // Update the quantity of remaining boxes.
        procedure DecoSpace;      // Dynamically decompose the remaining space.
        procedure Loading(Tree:TTree);      // Loading scheme.
    end;
procedure TTreeManager. Loading(Tree:TTree);
      begin
        if CanPacking then
          begin
              SelBoxes;
              CalPlace;
              PlaceBox;
              UpdateBoxInfo;
               DecoSpace;
               Loading(Tree^.LChild);
               Loading(Tree^.MChild);
               Loading(Tree^.RChild);
          end;
        end;
```

The algorithm expressed in the above pseudo code is inherently a recursive procedure. To represent the heuristic in a flowchart, the data structure of a stack [Preiss, 1999] is employed to transform it to a non-recursive process (See Fig. 5). Note that the stack structure has the property of first-in-last-out. The flowchart guarantees that the sequence of further decomposing the three subspaces starts from the left, to the middle, and then to the right child node.

Figure 5 is about here

This algorithm has been programmed using Delphi 6.0 and OpenGL on a Windows

2000 operating system Pentium PC (The computer program is available upon request). This program allows us to demonstrate this algorithm by using third-party test data and facilitates the comparative studies with other heuristic algorithms in the current literature in the next section.

5. Comparative studies

The CLP addressed in this article is similar to that described in Loh and Nee [1992], where 15 sets of test data are provided. Subsequently, Ngoi et al. [1994], Bischoff et al. [1995], Bischoff and Ratcliff [1995], and Gehring and Bortfeldt [1997] tested their algorithms using these benchmark data sets. To demonstrate the efficiency of our algorithm, we use these same test data and the results are shown in Table I along with the results from other algorithms. For the original test data LN01, LN02, …, LN15, readers are referred to Loh and Nee [20].

Table I is about here

The test result illustrates the algorithm and demonstrates its efficiency. The second last column of Table I indicates that only two out of the fifteen sets of test data cannot be completely loaded for our algorithm while the other four algorithms have to leave some boxes behind for three to five problems. In the case that other algorithms can completely load the boxes, our algorithm always achieves the same result. For LN07, only our algorithm completely packs all boxes. The last column in Table I shows that our algorithm consistently accomplishes the highest volume utilization whenever none of the five algorithms can completely load all boxes or only this algorithm loads all. Therefore, the average volume utilization of this algorithm is slightly higher than those of the other four algorithms based on these test data.

Furthermore, to demonstrate that the proposed dynamic space decomposition rule is superior to any of the four predetermined static schemes as given earlier, these 15 sets of test data are also employed to compare the loading results as given in Table II. In the table, Rule 1 means that the remaining space is always decomposed into three subspaces $\{S(A, I), S(B, M), S(C, N)\}$ after a box/block is loaded. Similarly, Rule 2, 3, and 4 refer to the fixed decompositions of $\{S(A, O), S(B, N), S(E, G)\}$, $\{S(A, K), S(F, M), S(C, N)\}$, and $\{S(A, J), S(F, M), S(E, N)\}$, respectively. It is clear that the dynamic decomposition rule consistently performs better than any of the four static decompositions whenever there

are boxes left. This result clearly confirms the superiority of the dynamic space decomposition rule.

Table II is about here

When the loading results are compared for different space decomposition rules in Table II, the optimal-fitting sequencing rule is applied. To examine the efficiency of this optimal-fitting sequencing rule, another comparative study is conducted by fixing the decomposition scheme to be dynamic but varying the sequencing rule from descending volume, to descending aggregate volume of homogeneous boxes, and to the optimal-fitting. Results are shown in Table III and demonstrate that the optimal-fitting sequencing rule always achieves higher volume usage than the other two rules if any box is left.

Table III is about here

For all of the aforementioned numerical experiments, our computer program completes calculations within 20 seconds after appropriate data are entered, demonstrating the computational efficiency of this heuristic algorithm. The program's output information includes the sequence of blocks, the orientation of each box and the stacking structure within each block, as well as each block's origin coordinates. This efficient computer implementation allows practitioners to apply this algorithm in real-world container loading practice.

6. An illustrative example

This heuristic is also applied to a real-world situation. For reasons of business confidentiality, the firm's identity is omitted. This plastic and rubber producer has contracted to export a batch of its product packed in six different types of carton boxes. The dimensions and quantities of each type of boxes are given in Table IV. The objective is to load as many boxes as possible into a standard 20-foot container with three dimensions of $590.5\text{cm} \times 235\text{cm} \times 239.2\text{cm}$. Traditionally, the loading process is carried out by using an empirical approach. According to the empirical calculation, all boxes of type 1 through 5 can be loaded, but for type 6, only 57 of the 129 boxes can be packed into the container. The overall container volume utilization rate is 85.74% as per this loading scheme.

Table IV is about here

By employing the algorithm proposed in this paper, the whole batch of the cargo can be fully loaded with a total volume utilization of 90.86%. The computer program based on this algorithm can visually demonstrate the loading pattern of different types of boxes as shown in Fig. 6. This windows-based software package allows a user to rotate the loaded container so that the loading status in the container may be viewed from different angles. Furthermore, the user may zoom in and out to make the container and boxes larger or smaller in order to obtain a better view. The “animation”, “next” and “back” buttons facilitate the user to envision the loading scheme in a progressive manner, thereby furnishing an invaluable guide for practitioners in practice.

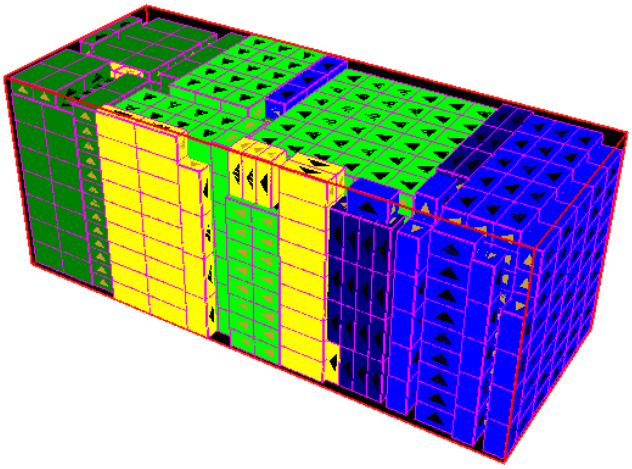

Fig. 6. Visual illustration of the loading scheme.

7. Conclusions

A heuristic algorithm is put forward to handle the container loading problem with weakly heterogeneous items packed in rectangular boxes. This algorithm exploits the tertiary tree structure to dynamically decompose the empty space into three subspaces and devises a unique optimal-fitting sequencing rule to rank remaining boxes. Along with the inner-right-corner-occupying placement rule and a holistic loading strategy, this algorithm is able to find a loading scheme that maximizes container volume utilization. Experiments with the commonly used 15 sets of test data and an illustrative example demonstrate the efficiency of this algorithm and suggest that this approach may be applied in real-world container-loading situations.

This algorithm assumes that there is no restriction on the orientation of boxes, that is, all boxes may be rotated around any of the three axes. However, it is trivial to relax this constraint and fix a particular direction. For instance, if a box must be face-up, it may still

rotate about the height axis (*z*-axis in Fig. 3), but not about the *x*- or *y*-axis. In this case, only two instead of six orientations are available and our algorithm can be slightly modified to reflect this change.

Significant problems remain open and deserve further investigations. For instance, the algorithm does not consider the weight distribution in the container [Davies and Bischoff, 1999] or limited weight bearing strength for any box [Bischoff, 2006]. These practical constraints dramatically complicate CLPs, and this heuristic is likely required to be substantially re-designed to accommodate them. For example, the loading sequence of certain boxes may have to be prioritized and the center of gravity has to be considered during the loading process. Further research is expected to address these issues.

Acknowledgements

The authors wish to express their appreciation to the Editor, Dr. Roman Slowinski, and the three anonymous reviewers for their constructive comments which significantly improve the quality of the paper. Kevin W. Li is also grateful for the financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) under the Discovery Grant program.

REFERENCES

Bischoff, E.E., 2006. Three-dimensional packing of items with limited load bearing strength. European Journal of Operational Research 168(3): 952-966.

Bischoff, E.E., Janetz, F., and Ratcliff, M.S.W., 1995. Loading pallets with non-identical items. European Journal of Operational Research 84(3): 681-692.

Bischoff, E.E. and Ratcliff, M.S.W., 1995. Issues in the development of approaches to container loading. Omega – International Journal of Management Science 23(4): 377-390.

Bortfeldt A. and Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research 131(1): 143-161.

Bortfeldt, A., Gehring, H., Mack, D., 2003. A parallel tabu search algorithm for solving the container loading problem. Parallel Computing 29, 641-662.

Davies, A.P. and Bischoff, E.E., 1999. Weight distribution considerations in container loading. European Journal of Operational Research 114(3): 509-527.

Dyckhoff, H., 1990. A topology of cutting and packing problems. European Journal of Operational Research 44(2): 145-159.

Eley, M., 2002. Solving container loading problems by block arrangement. European Journal of Operational Research 141(2): 393-409.

Gehring, H., and Bortfeldt, A., 1997. A genetic algorithm for solving the container loading problem. International Transactions in Operational Research 4(5/6): 401-418.

Gehring, H. and Bortfeldt, A., 2002. A parallel genetic algorithm for solving the container loading problem. International Transactions in Operational Research 9(4): 497-511.

George, J.A., and Robinson, D.F., 1980. A heuristic for packing boxes into a container. Computer and Operations Research 7(3): 147-156.

He, D., E, M., Cha, J., Wang, C., and Jiang, Y., 2000. A heuristic approach to container loading problem based on space decomposition and a rule for usage rate of packing space. Journal of Computer Aided Design and Computer Graphics, 12(5): 367-370 (in Chinese).

Jin, Z., Ohno, K., and Du, J., 2004. An efficient approach for the three-dimensional container packing problem with practical constraints. Asia-Pacific Journal of Operational Research 21(3): 279-295.

Knuth, D.E., *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, 3rd Edition, Addison-Wesley: Reader, 1997.

Loh, T.H. and Nee, A.Y.C., 1992. A packing algorithm for hexahedral boxes. In: Proceedings of the Conference of Industrial Automation, 115-126, Singapore, 1992.

Mack, D., Bortfeldt, A., and Gehring, H., 2004. A parallel hybrid local search algorithm for the container loading problem. International Transactions in Operational Research 11, 511-533.

Moura, A. and Oliveira, J.F., 2005. A GRASP approach to the container-loading problem. IEEE Intelligent Systems 20(4): 50-57.

Ngoi, B.K.A., Tay, M.L., and Chua, E.S., 1994. Applying spatial representation techniques to the container packing problem. International Journal of Production Research 32(1): 111-123.

Pisinger, D., 2002. Heuristics for the container loading problem. European Journal of Operational Research 141(2): 382-392.

Preiss, B.R., *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*, John Wiley & Sons, Inc.: New York, 1999.

Scheithauer, G., 1992. Algorithm for the container loading problem. In: *Operational Research Proceedings 1991*, 445-452, Springer: Berlin, 1992.

Takahara, S., 2005. Loading problem in multiple containers and pallets using strategic search method, In: Modeling Decisions for Artificial Intelligence, Proceedings Lecture Notes in Artificial Intelligence, Vol. 3558, 448-456, Springer-Verlag: Berlin, Heidelberg, 2005.

Terno, J., Scheithauer, G., Sommerweiss, U., and Riehme, J., 2000. An efficient approach for the multi-pallet loading problem. European Journal of Operational Research 123(2): 372-381.

Wang, Z., Li, K.W., and Zhang, X., 2006. A heuristic algorithm for the container loading problem with heterogeneous boxes, In: Proceedings of 2006 IEEE International Conference on Systems, Man, and Cybernetics, 5240-5245, IEEE: New Jersey, 2006.

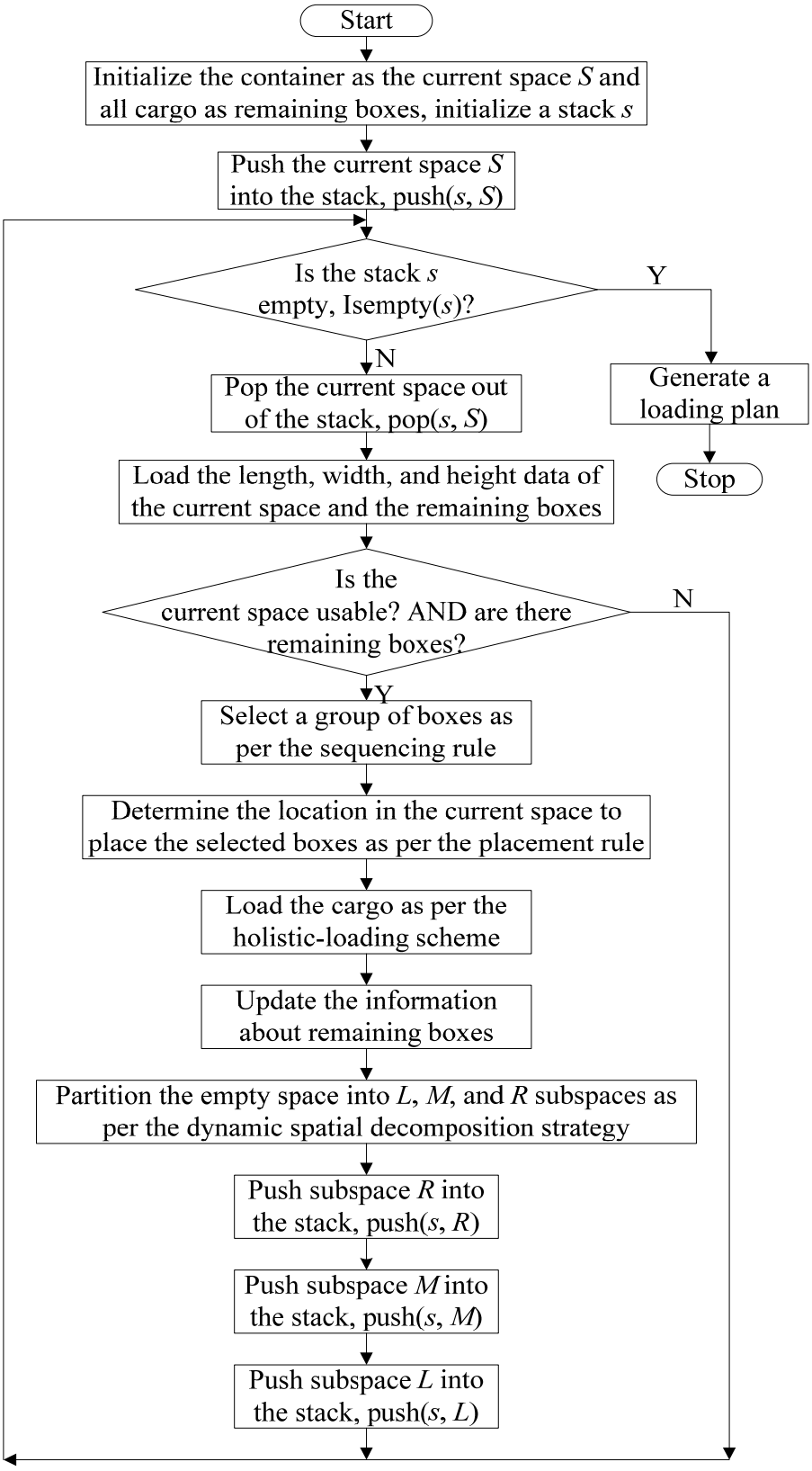

Fig. 5. The flowchart of the heuristic algorithm

TABLE I

COMPARATIVE RESULT WITH OTHER ALGORITHMS

Problem	Ngoi et al. [1994]		Bischoff et al. [1995]		Bischoff and Ratcliff [1995]		Gehring and Bortfeldt [1997]		This algorithm	
	BL[a]	VU[b]	BL	VU	BL	VU	BL	VU	BL	VU
LN01	0	62.5	0	62.5	0	62.5	0	62.5	0	62.5
LN02	54	80.7	23	89.7	35	90.0	39	89.5	35	90.7
LN03	0	53.4	0	53.4	0	53.4	0	53.4	0	53.4
LN04	0	55.0	0	55.0	0	55.0	0	55.0	0	55.0
LN05	0	77.2	0	77.2	0	77.2	0	77.2	0	77.2
LN06	48	88.7	24	89.5	77	83.1	32	91.1	37	92.9
LN07	10	81.8	1	83.9	18	78.7	7	83.3	0	84.7
LN08	0	59.4	0	59.4	0	59.4	0	59.4	0	59.4
LN09	0	61.9	0	61.9	0	61.9	0	61.9	0	61.9
LN10	0	67.3	0	67.3	0	67.3	0	67.3	0	67.3
LN11	0	62.2	0	62.2	0	62.2	0	62.2	0	62.2
LN12	0	78.5	3	76.5	0	78.5	0	78.5	0	78.5
LN13	2	84.1	5	82.3	20	78.1	0	85.6	0	85.6
LN14	0	62.8	0	62.8	0	62.8	0	62.8	0	62.8
LN15	0	59.5	0	59.5	0	59.5	0	59.5	0	59.5
Average		69.0		69.5		68.6		69.9		70.2

[a]BL = No. of boxes left, [b]VU = Volume utilization (%).

Table II. Comparisons of static and dynamic space decomposition rules

Problem	Rule 1		Rule 2		Rule 3		Rule 4		Dynamic rule	
	BL[a]	VU[b]	BL	VU	BL	VU	BL	VU	BL	VU
LN01	0	62.5	0	62.5	0	62.5	0	62.5	0	62.5
LN02	37	89.2	72	85.5	35	90.7	51	87.3	35	90.7
LN03	0	53.4	0	53.4	0	53.4	0	53.4	0	53.4
LN04	0	55.0	0	55.0	0	55.0	0	55.0	0	55.0
LN05	0	77.2	0	77.2	0	77.2	0	77.2	0	77.2
LN06	36	86.8	34	90.5	32	88.3	34	91.4	37	92.9
LN07	0	84.7	17	74.8	2	83.3	7	81.4	0	84.7
LN08	0	59.4	0	59.4	0	59.4	0	59.4	0	59.4
LN09	0	61.9	0	61.9	0	61.9	0	61.9	0	61.9
LN10	0	67.3	0	67.3	0	67.3	0	67.3	0	67.3
LN11	0	62.2	0	62.2	0	62.2	0	62.2	0	62.2
LN12	2	76.8	10	70.7	1	78.0	1	77.4	0	78.5
LN13	2	84.4	8	81.6	1	85.1	3	83.8	0	85.6
LN14	0	62.8	0	62.8	0	62.8	0	62.8	0	62.8
LN15	0	59.5	0	59.5	0	59.5	0	59.5	0	59.5
Average		69.7		68.3		69.8		69.5		70.2

[a]BL = No. of boxes left, [b]VU = Volume utilization (%).

Table III. Comparisons of different sequencing rules

Problem	Descending aggregate volume		Descending volume		Optimal-fitting	
	BL[a]	VU[b]	BL	VU	BL	VU
LN01	0	62.5	0	62.5	0	62.5
LN02	41	89.2	61	89.2	35	90.7
LN03	0	53.4	0	53.4	0	53.4
LN04	0	55.0	0	55.0	0	55.0
LN05	0	77.2	0	77.2	0	77.2
LN06	37	92.9	62	87.2	37	92.9
LN07	0	84.7	10	82.4	0	84.7
LN08	0	59.4	0	59.4	0	59.4
LN09	0	61.9	0	61.9	0	61.9
LN10	0	67.3	0	67.3	0	67.3
LN11	0	62.2	0	62.2	0	62.2
LN12	1	78.0	3	77.0	0	78.5
LN13	0	85.6	0	85.6	0	85.6
LN14	0	62.8	0	62.8	0	62.8
LN15	0	59.5	0	59.5	0	59.5
Average		70.1		69.5		70.2

[a]BL = No. of boxes left, [b]VU = Volume utilization (%).

TABLE IV
DIMENSIONS AND QUANTITIES OF BOXES

Type	Length (cm)	Width (cm)	Height (cm)	# of boxes
1	51.0	26.0	15.9	47
2	43.0	31.0	17.5	360
3	32.9	22.7	30.6	485
4	51.0	26.0	15.9	69
5	41.0	21.0	24.3	248
6	43.5	31.0	17.5	129