# An Efficient Algorithm for Solving the Container Loading Problem

**Conference Paper** · January 2007

DOI: 10.1007/978-3-540-74450-4_36 · Source: DBLP



**2 authors**, including:

Kun He
Huazhong University of Science and Technology
**46** PUBLICATIONS **244** CITATIONS

SEE PROFILE

# An Efficient Algorithm for Solving the Container Loading Problem

Wenqi Huang and Kun He*

Theoretical Computer Science Institute, College of Computer Science,
Huazhong University of Science and Technology, Wuhan 430074, China
wqhuang@mail.hust.edu.cn, brooklet60@gmail.com

**Abstract.** We present a new algorithm for the three-dimensional packing problem with a single container to be loaded. Deviates from the traditional approaches, it uses a principle —— "largest caving degree" such that items are packed as closely as possible. Then, it incorporates a backtracking strategy that gains a good trade-off between solution quality and computation time. We tested our algorithm using all the 47 related benchmarks from the OR-Library that have no orientation constraints. Experiments indicate the algorithm's good quality. The container volume utilization, achieved within comparable time, is 94.6% on average. This significantly improves current best-known results in the literature by 3.6%.

**Keywords:** Heuristic; Cargo Loading; Cutting and Packing.

## 1 Introduction

The cutting and packing (C&P) problem is a well-motivated research issue in combinatorial optimization area. Theoretically, it is NP-hard in the strong sense and is very difficult to solve [1]. Practically, it has a wide variety of applications in the real world. E.g., the efficient cargo transportation is of high economic value in logistics, and near optimal cargo loading leads to lower costs all over the distribution system. A C&P problem usually corresponds to a job scheduling problem. E.g., the two-dimensional (2D) bin packing problem is equivalent to a parallel job scheduling problem on identical machines located on a 2D mesh [2]. Therefore, methods for the C&P problem are also useful for the scheduling problem, which is another kind of important problem both in theory and practice.

In this paper, we consider a typical three-dimensional (3D) C&P problem: the container loading problem. We assume there is no orientation constraint, i.e. the item may be rotated by $90^o$ before it is packed. This complicates the problem since the number of possible packing is now a factor of $6^n$ larger for $n$ items. Yet, it may allow better packing, and small modification on algorithms could easily eliminate this scenario.

Heuristic approaches are often the feasible options for the 3D C&P problem. Basic methods are wall building [1,3,4], layering [5], guillotine cutting [6], block arrangement [7,8,9], etc. Tree-search [1,7], tabu search [8], simulated annealing [8],

* Corresponding author.

B. Chen, M. Paterson, and G. Zhang (Eds.): ESCAPE 2007, LNCS 4614, pp. 396–407, 2007.

or genetic search [9] is incorporated to improve the solution's quality. Sometimes, parallel methods are used to shorten the computation time [8,9]. Based on the wall building and layering methods, Lim et al. proposed a basic algorithm MFB and a strengthened algorithm MFB_L [4]. They tested on 760 benchmarks from the OR-Library [10]. The results of MFB_L were about 3% better on average than that in [3], which was the only available work they found containing packing results for the frontier 700 benchmarks.

We proposed a new approach for the 3D packing problem. The inspiration originates from an old adage "gold corner, silver side and grass belly" in Chinese I-go. The adage has concentrated human's experience and wisdom formed in the last thousand years. It indicates that corner worth most while center worth dirt on the chessboard. Moreover, we improved the idea with "diamond cave". Based on a principle of "largest caving degree" [11], a packing item should always occupy a corner, or even a cave, bounded by surfaces of the packed items or the container. In this way, items are packed as closely as possible. We tested our algorithm using all the 47 benchmarks from the OR-Library [10] that allow rotations. Results generated are compared with those in [4] which tested the same data. Experiments indicate that we are able to achieve an average packing utilization of 94.6%. To our knowledge, this significantly improves the best-known results by 3.6%.

## 2 Problem Specification

The container loading problem is to pack a subset of rectangular items (e.g. boxes) into a single rectangular container with fixed dimensions. The objective is to find a feasible solution that maximizes the total volume of the packed items, i.e. maximizes the container's volume utilization. A solution is feasible if: ① each item is packed completely in the container; ② there is no overlapping between any two items; ③ each item is placed parallel to the surfaces of the container.

Given a container with dimensions $(L,W,H)$ and a set of $n$ items $S=\{(l_1,w_1,h_1),...,(l_n,w_n,h_n)\}$. Variables $l_i$, $w_i$ and $h_i$ are the three dimensions of item $i$. Without loss of generality, suppose all the variables are plus integer numbers. Consider the container embedded into a three-dimensional Cartesian reference frame. The lower-left-near corner coincides with the origin and the upper-right-far corner coincides with point $(L,W,H)$, as Fig. 1 shows. Let $\delta_i\in\{0,1\}$ denotes whether item $i$ is packed into the container. If item $i$ has been packed, let $(x_{i1},y_{i1},z_{i1})$ and $(x_{i2},y_{i2},z_{i2})$ denote the coordinates of its lower-left-near corner and upper-right-far corner respectively. Then, the problem can be formulized as follows:

$$\max \sum_{i=1}^{n} l_i w_i h_i \delta_i$$

Subject to

(1) $(x_{i2}-x_{i1},y_{i2}-y_{i1},z_{i2}-z_{i1})\in\{(l_i,w_i,h_i),(w_i,l_i,h_i),(l_i,h_i,w_i),(h_i,l_i,w_i),(h_i,w_i,l_i),(w_i,h_i,l_i)\}$;

(2) $\max(\ \max(x_{i1},x_{j1})-\min(x_{i2},x_{j2}),\max(y_{i1},y_{j1})-\min(y_{i2},y_{j2}),$
$\max(z_{i1},z_{j1})-\min(z_{i2},z_{j2})\ )\delta_i\delta_j\geq 0$;

(3) $0 \le x_{i1} < x_{i2} \le L$ , $0 \le y_{i1} < y_{i2} \le W$ , $0 \le z_{i1} < z_{i2} \le H$ ;

(4) $\delta_i \in \{0,1\}$ ;

Where $i, j = 1, 2, ..., n$ , $i \neq j$ . Item $i$ has six orientations whose dimensions on x-, y-, and z-axis are $(l_i, w_i, h_i)$, $(w_i, l_i, h_i)$, $(l_i, h_i, w_i)$, $(h_i, l_i, w_i)$, $(h_i, w_i, l_i)$ and $(w_i, h_i, l_i)$, with their orientation number from one to six respectively. Constraints (1) to (3) are corresponded to the three conditions of a feasible solution.

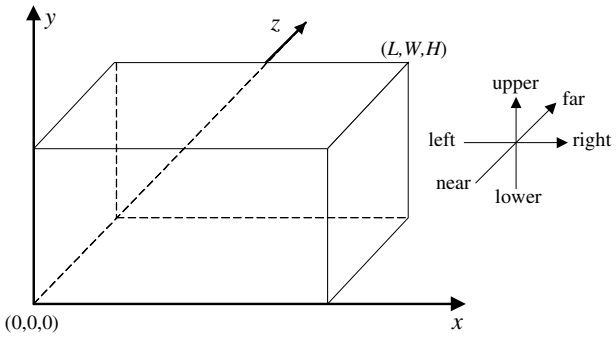


**Fig. 1.** Coordinate system

## 3 Algorithm Description

The main idea of our algorithm is to always do a Real Corner Occupying Action (RCOA) with the largest caving degree. A RCOA is an action that places an item at a corner where three orthogonal surfaces meet. Caving degree is a value defined to judge how close a packing item is to other items filled in already.

### 3.1 Definitions

**Definition 1 (Real Corner).** It is an unoccupied space where three orthogonal surfaces meet and form a corner. The surface could be an interior surface of the container, or an exterior surface of a packed item. The three surfaces intersect each other on surface or edge.

The six surfaces of the container can be regarded as six items with unit length for the third dimension, and are packed at the corresponding positions in the beginning. The three surfaces of a real corner belong to different items. A corner is denoted by its coordinates $(x,y,z)$ and its corner direction. There are eight different corner directions. They are corresponded with the directions of the eight corners, formed by the coordinate planes in the eight quadrants of the 3D Cartesian reference frame.

**Theorem 1.** There are at most two real corners formed by any three items.

*Proof.* According to the real corner's definition, any two items of a real corner must intersect each other on surface or edge.

① If the two items intersect on surface, as in Fig.2 (a) item *i* and item *k* or item *j* and item *k* show, there are at most four intersecting edges. And at each direction, there are at most two parallel edges. The surface of the third item must be orthogonal with the intersecting edges. The third item could intersect at most two parallel edges to form corners, as in Fig.2 (a) item *j* shows. Or the third item could intersect one edge with its two parallel surfaces, as in Fig.2 (a) item *i* shows. So, there are at most two real corners formed.

② If the two items intersect on edge, as in Fig.2 (b) item *i* and item *j* show, there are at most two pairs of intersecting surfaces. Each pair of surfaces is orthogonal with each other and belongs to the two items respectively. So, there are at most two real corners formed with the third item, as Fig.2 (b) item *k* shows.

So, the theorem is true. □

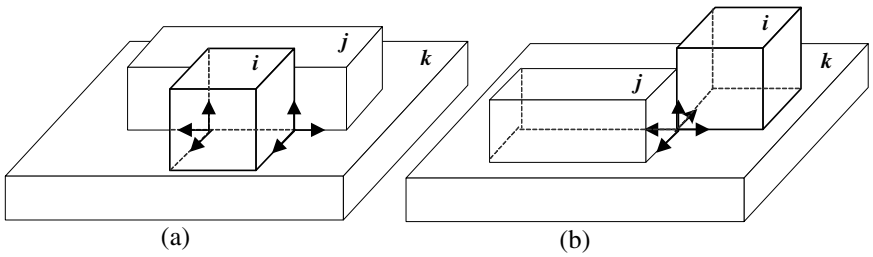


**Fig. 2.** Real corners formed by three items

**Definition 2 (Real Corner Occupying Action, RCOA).** A packing action includes three aspects: which item to be packed, which position to be placed, and which item orientation to be set. An action is called a RCOA if the packing item occupies a real corner.

A RCOA is denoted by the occupied corner's coordinate and direction, as well as the packing item's id and orientation.

**Definition 3 (Distance of Two Point Sets).** It's an infimum on distances of two points chosen from the two sets respectively.

$$d(A,B) = \inf\{d(a,b): a \text{ in } A, b \text{ in } B\}. \quad (1)$$

**Definition 4 (Distance of Two Items, $d_{ij}$).** Regarding each item as a point set, their distance is the Minkowski distance between the two sets.

The distance between item *i* and *j* is $d_{ij}=dx_{ij}+dy_{ij}+dz_{ij}$, as Fig. 3 shows, where

$$dx_{ij}=\max(|x_{ic}-x_{jc}|-\frac{l_i+l_j}{2},0),$$

$dy_{ij}=\max(|y_{ic}-y_{jc}|-\frac{w_i+w_j}{2},0)$,

$dz_{ij}=\max(|z_{ic}-z_{jc}|-\frac{h_i+h_j}{2},0)$,

and $(x_{ic},y_{ic},z_{ic})$, $(x_{jc},y_{jc},z_{jc})$ are the central coordinates of item $i$ and item $j$ respectively.

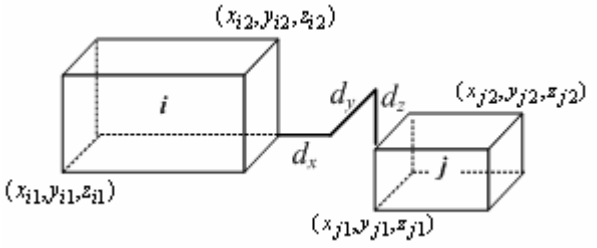


**Fig. 3.** Distance of two items

**Definition 5 (Paste Number, $k_i$ ).** It's the number of pasted surfaces for a RCOA packing item. $k_i\in\{3,4,5,6\}$ . A surface is pasted if the intersect area with another surface is larger than 0.

**Definition 6 (Paste Ratio, $p_i$ ).** It's the total pasted area of a RCOA packing item, divided by the total surface area of the item. $p_i\in(0,1]$

**Definition 7 (Distance of a RCOA, $d_i$ ).** Aligned with the $k$ pasted plane and their normal directions (pointing to the space the packing item is in), a space $\Omega$ is enclosed with 6-$k$ dimensions. Distance of a RCOA is the minimum distance between the packing item and a set of packed items. An item, including the six items forming the container, is considered in the set if its intersection with space $\Omega$ is not null, the intersect area or volume is larger than 0, and it does not paste the packing item. Let $d_i=0$ when $k$=6.

$$d_i=\begin{cases}\min(d_{ij}, j\cap\Omega\neq\varnothing\wedge j\cap i=\varnothing) & k=\{3,4,5\}\\ 0 & k=6\end{cases}. \quad (2)$$

Fig. 4 shows the space $\Omega$ for packing item $i$ with its upper, down, left and far surfaces pasted ( $k$=4 ), and its occupied corner is at the upper-left-far vertex. When computing $d_i$, item $a$ pastes with item $i$ and it is not considered, while item $b$ and item $c$ are considered; the near surface and right surface of the container are considered too.

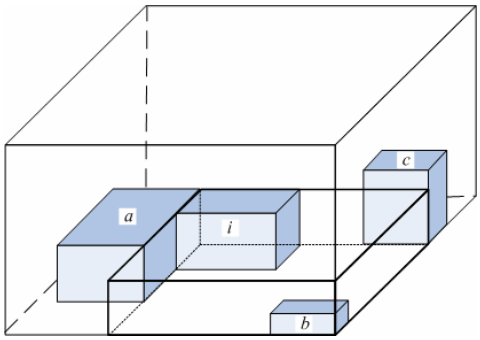


**Fig. 4.** Space Ω

**Definition 8 (Caving Degree, $C_i$ ).** It is defined on a RCOA to judge how close the packing item is with other items filled in already.

$$C_i = \mu(k_i + p_i) - \frac{d_i}{\sqrt[3]{l_i w_i h_i}},\ \mu = \max(L, W, H)\ . \tag{3}$$

The larger the caving degree is, the better a RCOA is. A good caving degree is determined on the largest paste number, the largest paste ratio, and the smallest distance of a RCOA in dictionary order.

### 3.2 Packing Procedure

In the main procedure, a packing method is invoked recursively and items are packed one by one from outside to the container. Suppose at a packing step, some items have been packed and some remain outside. The packing method searches all feasible RCOAs by iterating all free real corners, all free items and six item orientations. Thereafter, based on the largest caving degree principle, an item is selected to pack in an appointed position with an appointed orientation. This action makes the packing item as closely as possible to other packed items.

So, one item is packed at each packing step. The inside items are increasing and the outside items are decreasing until the final packing step comes. At the final step, all items have been packed into the container without overlapping, or none of the remainders can be packed in.

Above procedure is the greedy $B_0$ algorithm. The strengthened $B_1$ and $B_2$ algorithms add backtracking strategy on $B_0$ to get higher solution quality by sacrificing the computation time.

### 3.3 Ranking Scheme

In $B_0$ algorithm, the ranking scheme is employed to select a RCOA at each packing step, which can be formulated as follows:

Main criteria: largest caving degree.
Tiebreaker 1: smallest length of the long dimension.
Tiebreaker 2: smallest length of the middle dimension.

Tiebreaker 3: smallest length of the short dimension.
Tiebreaker 4: smallest coordinate z of the gravity.
Tiebreaker 5: smallest coordinate y of the gravity.
Tiebreaker 6: smallest coordinate x of the gravity.
Tiebreaker 7: smallest orientation number.

Tiebreakers 1 to 3 are aimed at packing potentially awkward items early on. Tiebreakers 4 to 7 primarily serve to produce a definitive selection when the former rules lead to identical scores.

### 3.4 Basic Algorithm

There is no item in the container in the beginning. The six surfaces of the container are regarded as six items, and are packed at the corresponding positions. Then, the packing method is invoked recursively. At the end of each packing step, the corners the packing item consumed are deleted from the free corner map, and the corners the packing item created are added to the free corner map. An update procedure is used to find the consumed and created corners and to update the free corner map. E.g., in Fig. 5, an item is packed at the lower-left-near corner of the container. Its dimensions on x-, y- and z-axis are $l$, $w$ and $h$ respectively. The corner it consumed is $(0,0,0,1)$, and the corners it created are $(l,0,0,1)$, $(0,w,0,1)$ and $(0,0,h,1)$.

Algorithm $B_0$

*Input:* a container denoted by ($L$,$W$,$H$); a free item list with each item denoted by ($l_i$,$w_i$,$h_i$).

```
init container(L,W,H);
 packing(freeItemList){
  if freeItemList is null, return;
  for each free real corner in freeCornerMap{
    for each outside item in freeItemList{
      for each item orientation {
        check current RCOA's validity;
        if it's a feasible RCOA,
          compute its caving degree;
      }
    }
  }
  if current feasible RCOAs are not null {
    choose a best RCOA to do by the ranking scheme;
    move the selected item from freeItemList to
usedItemList;
    update freeCornerMap;
    packing(freeItemList);
  }
}
```

*Output:* a list of packed items in usedItemList, each with coordinates of its lower-left-near and upper-right-far corners; container volume utilization.

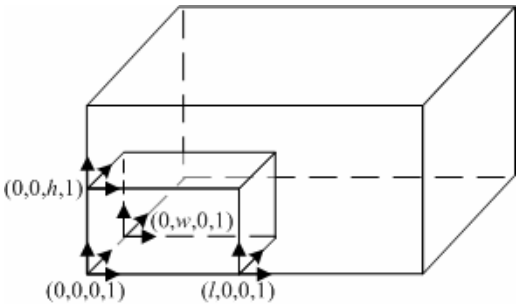


**Fig. 5.** Corners consumed and created

### 3.5 Strengthened Algorithm

Backtracking strategy is incorporated to further improve the packing utilization. At each packing step of algorithm $B_1$, instead of choosing a RCOA with the largest caving degree, we choose a RCOA with the largest volume utilization that pseudo computed by $B_0$. "Pseudo computation" means that the outside items are packed temporarily by $B_0$ so as to get corresponding container volume utilization for the candidate RCOA item, and are removed from the container later. In case of ties, the same tiebreakers as that of algorithm $B_0$ are used. Figure 5 depicts the evolvement of the volume utilization at each packing step, which is produced by algorithm $B_1$ on two cases of test file 9 from the OR-Library.

All feasible RCOAs can be regarded as a neighborhood of current solution, and the neighbor with the largest volume utilization is selected as the next step. Instead of terminating when there is no neighbors in $B_1$, algorithm $B_2$ will terminate when an optimal filling is found, or the volume utilization does not improve at the next step. As can be seen in Fig. 6, case 2 will terminate at step 6, and case 15 will terminate at step 4. This gives a good trade-off between solution quality and computation time.

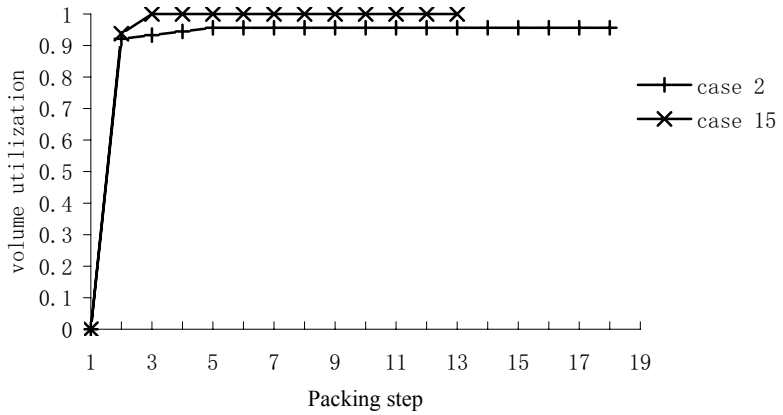


**Fig. 6.** Volume utilization at each packing step

## 4 Experimental Results

We have implemented our algorithms in java and run them on a 1.7GHz IBM notebook. Benchmarks are from OR-Library [10], which is a collection of test data sets for a variety of Operations Research (OR) problems. There are nine sets of test data with 762 test cases for the 3D packing problem. The 47 cases in file 9 allow rotations and are considered here. They are weakly heterogeneous problems, and their item types are from two to five.

Figure 7 refers to the volume utilization of algorithm $B_0$, where x-axis represents the test case index and y-axis represents the container volume utilization. Figure 8 contains a curve graph detailing the computation time of $B_0$ involved in generating the solutions. The average container volume utilization is 87.2% on average. Moreover, algorithm $B_0$ is very fast with average time 1.13 seconds. It takes 0.22 to 2.7 seconds to pack one container, except 4.22 seconds for the 45th case and 11 seconds for the 46th case. Note that in [4], the authors did not compute the 45th to the 47th cases.

Figure 9 refers to the volume utilization of algorithm $B_2$, where x-axis represents the test case index and y-axis represents the container volume utilization. Figure 10 contains a curve graph detailing the computation time of $B_2$ involved in generating the solutions. The average container volume utilization is 94.6% on average. The backtracking strategy enables $B_2$ to improve the average volume utilization by 7.4%. Moreover, algorithm $B_2$ is fairly fast with average time 876 seconds. It takes 0.25 seconds to 48 minutes to pack one container, except 2.66 hours for the 45th case and 3.85 hours for the 46th case. Note that in [4], the authors did not compute the 45th to the 47th cases.

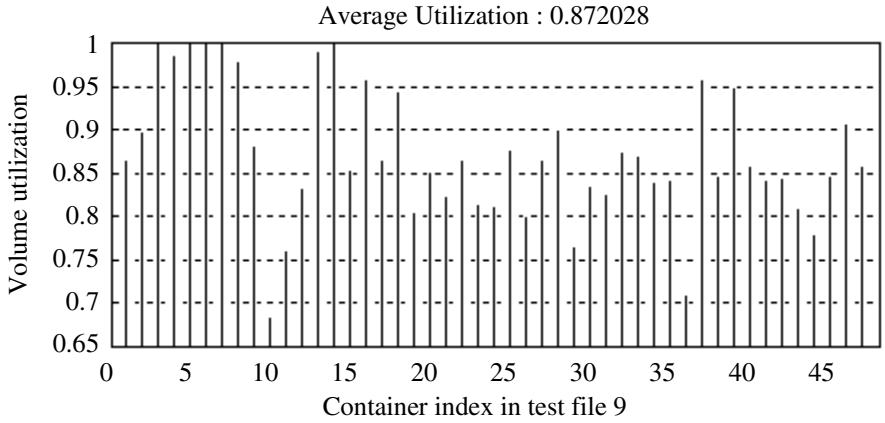


**Fig. 7.** Volume utilization for algorithm $B_0$

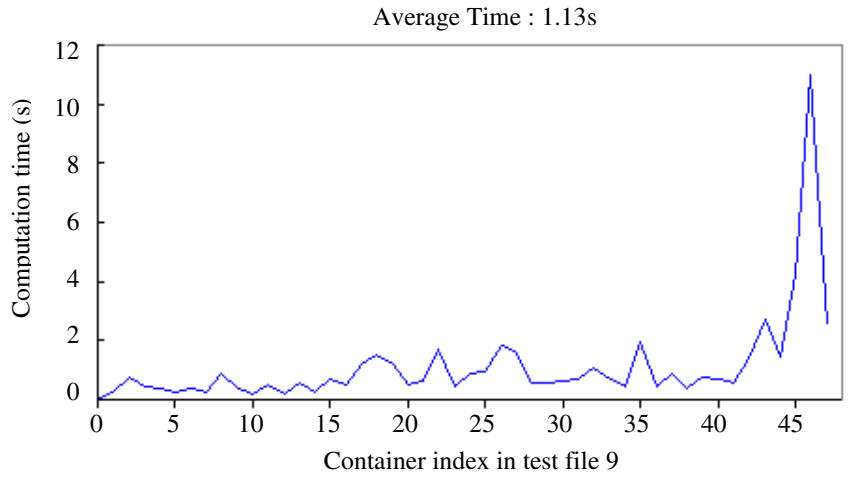


**Fig. 8.** Computation time for algorithm $B_0$

In [4], the authors computed the anterior 44 test cases of file 9. What we did is 3.6% better than theirs judged by these cases. Table 1 shows the comparison. As can be seen, the average volume utilization is 87.21% of $B_0$ as opposed to 83.68% of MFB_L [4] applied, and the average volume utilization is 94.59% of $B_2$ as opposed to 91% of MFB_L [4] applied. The $B_0$ and $B_2$ algorithms perform better than that of MFB and MFB_L respectively.

As for the time, since they used a small-scale computer while we used a personal PC, our algorithms are at least not slower than theirs. And as Fig. 8 and Fig. 9 show, all results are obtained within reasonable times.

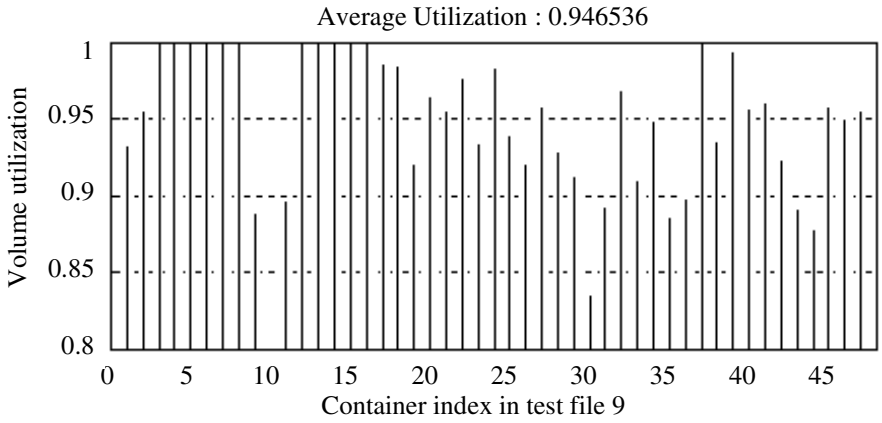


**Fig. 9.** Volume utilization for algorithm $B_2$

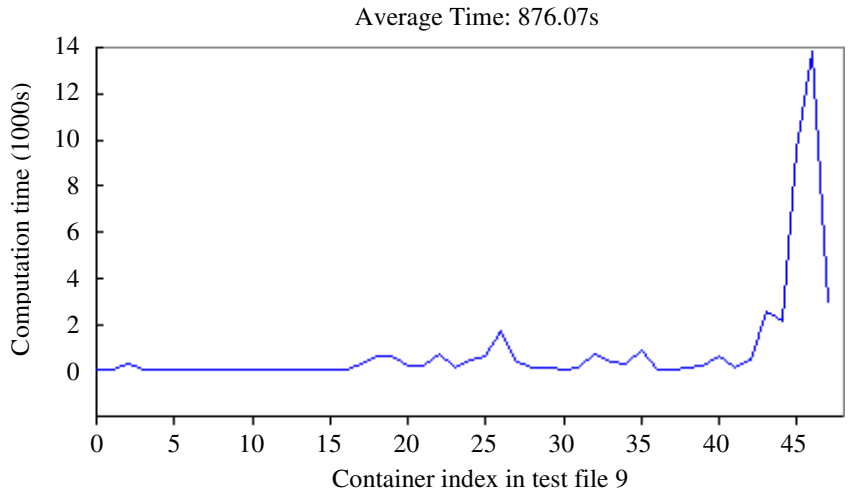


**Fig. 10.** Computation time for algorithm $B_2$

**Table 1.** Comparisons with MFB and MFB_L algorithms

| Comparisons | MFB | $B_0$ | MFB_L | $B_2$ |
|---|---|---|---|---|
| Test computer | Unix Alpha 600 MHz | Windows 1.7GHz | Unix Alpha 600 MHz | Windows 1.7GHz |
| Average utilization | 0.8368 | 0.8721 | 0.91 | 0.94595 |
| Average time | 2-3s | 0.80s | 205s | 337.78s |

## 5 Conclusions and Future Work

We presented a new approach, which is a heuristic in nature, for the three-dimensional container packing problem. The rotation of items is allowed. The approach is efficient yet not complicated. Different from the previous approaches, it uses a conception of caving degree to judge how good a real corner occupying action is. The approach achieved a high average packing utilization of 94.6%, computed within rather short times, using all the 47 related benchmarks from the OR-Library test suite. This is 3.6% better than the best-known results. Tests show that our approach is comparable, in terms of the container volume utilization, to other approaches reported.

There are some factors that can be considered aftertime, such as orientation restrictions, load bearing strength of items, loading stability, grouping of items, and weight distribution. Other variants of the 3D cutting and packing problem, e.g. strip packing and bin packing, may use this approach for reference. More work will be done on these in the future.

**Acknowledgments.** This work was supported by National Natural Science Foundation of China (Grant No. 10471051) and by the NKBRPC (Grant No. G2004CB318000).

## References

1. David, P.: Heuristics for the Container Loading Problem. European Journal of Operational Research 141, 382–392 (2002)
2. Zhang, G.C.: A 3-Approximation Algorithm for Two-Dimensional Bin Packing. Operations Research Letters 33, 121–126 (2005)
3. Bischoff, E.E., Ratcliff, M.S.W.: Issues in the Development of Approaches to Container Loading. OMEGA: The International Journal of Management Science 23, 377–390 (1995)
4. Lim, A., Rodrigues, B., Wang, Y.: A Multi-faced Buildup Algorithm for Three-Dimensional Packing Problems. OMEGA: The International Journal of Management Science 31, 471–481 (2003)
5. Loh, T.H., Nee, A.Y.C.: A Packing Algorithm for Hexahedral Boxes. In: Proceedings of the Conference of Industrial Automation, Singapore, pp. 115–126 (1992)
6. Morabito, R., Arenales, M.: An AND/OR Graph Approach to the Container Loading Problem. International Transactions in Operational Research 1, 59–73 (1994)
7. Eley, M.: Solving Container Loading Problems by Block Arrangements. European Journal of Operational Research 141, 393–409 (2002)
8. Mack, D., Bortfeldt, A., Gehring, H.: A Parallel Local Algorithm for the Container Loading Problem. International Transactions in Operational Research 11, 511–533 (2004)
9. Gehring, H., Bortfeldt, A.: A Parallel Generic Algorithm for Solving the Container Loading Problem. European Journal of Operational Research 131, 143–161 (2001)
10. OR-Library: http://mscmga.ms.ic.ac.uk/jeb/orlib/thpackinfo.html
11. Huang, W.Q., Xu, R.C.: Introduction to the Modern Theory of Computation — Background, Foreground and Solving Method for the NP-hard Problem (in Chinese). Science, Beijing