# Two-dimensional packing for irregular shaped objects

**Conference Paper** · February 2003

DOI: 10.1109/HICSS.2003.1174211 · Source: IEEE Xplore



**4 authors**, including:

Andrew Lim
Red Jasper Holdings
**494** PUBLICATIONS **8,837** CITATIONS

SEE PROFILE

# Two-Dimensional Packing For Irregular Shaped Objects

[1]Ping Chen, [1]Zhaohui Fu, [2]Andrew Lim and [3] Brian Rodrigues

[1]Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
{chenp,fuzh}@comp.nus.edu.sg

[2]Department of IEEM
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
iealim@ust.hk

[3]School of Business
Singapore Management University
469 Bukit Timah Road, Singapore 259756
br@smu.edu.sg

## Abstract

*Packing problems arise in a wide variety of application areas. The basic problem is that of determining an efficient arrangement of different objects in a region without any overlap. The simplest packing problem is rectangular packing, where both the objects and the region are rectangular. Many research works have been done on two and three dimensional rectangular packing. However there are many situations when either objects or the containing region is irregular in shape. In the project, we concentrate on two-dimensional packing problems involving irregular shaped objects (both convex and concave). We have applied several approaches to solve such problems.*

## 1 Introduction

Packing problems arise from a variety of situations including pallet loading, textile cutting, container stuffing and placement problems. Such problems are optimization problems that are concerned with finding a good arrangement of multiple objects (2-D or 3-D) in a larger containing region without overlap. The usual objective of the allocation process is to maximize the material utilization and hence to minimize the wasted area.

The packing problem becomes much simpler when both objects and the containing region are rectangular in shape. Many research works have been done on two and three dimensional rectangular packing problems. However, in many practical applications, objects and containing regions may have irregular shapes. Due to the geometrical complexity introduced by irregular shapes, such problems are not as well studied as rectangular packing.

We can formally define the problem as follows:
Given a set of objects $O$ with irregular shapes and an irregular container $C$, find the set of objects $P \subseteq O$ that can be arranged in $C$ without any overlapping such that the value $\frac{\sum area(P_i)}{area(C)}$ is maximized.

There are relatively few works on two dimensional packing with arbitrary shapes, compared to 2D rectangle packing. They can be classified into two types, namely nesting and packing.

Nesting is an approach in which irregular objects are nested in simpler regular shapes, and these simpler shapes are then packed into an available area.

Rectangle is the most popular shape for nesting. Freeman and Shapira [7] made the first attempt in enclosing irregular shapes in a convex polygon and then finding the required rectangle by iteratively basing the rectangle on each of the edges of the polygon. In practice solutions of this type will only prove satisfactory if the pieces themselves

are close to rectangular so that the wasted area is small. Adamowicz and Albano [1] proposed another approach by nesting more than one piece together when pieces are far from rectangular in shape. They placed a threshold on the amount of waste they are willing to accept in any enclosure. Another alternative using rectangle packing is to nest all the pieces into identical polygons which can be used to tile the plane. These include triangles, quadrilaterals, pentagons and hexagons. Dori and Ben-Bassat [5] based their packing algorithm using hexagons. They nested the required shapes into a polygon which minimizes the wasted area and then surround this with a hexagon which is suitable for tiling the plane. However, their approach is limited to convex polygons.

Straightforward single pass packing strategies involve taking the pieces in order and placing them on the stock-sheet according to a given placement policy. This may be repeated several times for different orderings or different placements and the best solution chosen, or a more intelligent method may be used in a single pass. Qu and Sanders [15] first sorted the pieces in decreasing length order and then placed along two adjacent edges of the stock-sheet. Dowsland and Dowsland [6] used a leftmost placement policy together with a random ordering of the pieces. Albano and Sapuppo [2] were concerned with the more complex problem in which the pieces may be placed in a number of different orientations. There is another approach which has become increasingly popular in recent years. It is to produce an initial layout and then to use small changes in order to improve it. Such an approach may either continuously seek for improvement, or may incorporate meta-heuristic techniques such as simulated annealing or tabu search in order to allow up-hill, or non-improving moves. Blazewicz *et al.*[4] suggested a tabu search algorithm. Sakait and Hae [16] applied genetic algorithms. They discretized the object and represented the object as a linked list of cells. The linking from one cell to the next is given in the form of an eight connectedness. Thus this approach is applicable to any arbitrarily shaped object. The total layout space is divided into a finite number of cells for mapping it into a new 2D genetic algorithm chromosome. The genetic operators of mutation and crossover have been suitably modified to suit this problem.

Keishi, Shigetoshi and Yogi [13] developed an algorithm to handle convex-rectilinear blocks by enhancing the BSG-based packing algorithm. BSG stands for Bounded-Sliceline Grid. Maggie and Wayne [14] made use of the SP (Sequence Pair) structure. Each rectilinear shaped block is partitioned into a set of rectangular sub-blocks, each of them individually handled in a sequence pair as a unit block. The horizontal and vertical graphs are constructed. Packing can be obtained by applying the longest path algorithm on the graphs.

Prior work in this area is largely specific to a particular problem domain. For example, the object pieces from the textile industry usually have similar shapes and sizes. Such restrictions found in the problem domain naturally simplify the problem. This provides the motivation for us to solve a more general class of packing problems, where the objects and even the containers can be arbitrary shaped. Objects can be packed, whether they are convex or concave. In this paper, different approaches are presented in details. We first adopt a rectilinear representation for irregular shapes. Because of the simplicity of this representation (like using pixels to draw an object), we implemented several heuristic methods, like Greedy, Genetic Algorithms(GA) and Tabu Search (TS), to obtain a good packing arrangement. We modified the GA approach first used by Sakait and Hae [16]. We also proposed a Tabu Search based method, which will converge faster than GA. The results turns out to be better than previous works by Sakait and Hae [16]. We have also achieved satisfactory packing when objects have more rectilinear-like shapes. However, if the objects are arbitrarily irregular polygons, the packing can be very poor. This motivated us to adopt a more powerful boundary representation. Such a method will represent the polygon by a vector of its vertices. Such representation is seldom found in previous works. Its simple yet powerful approach empowers us more freedom in manipulating the objects. Meanwhile, it also brings us great difficulty in optimizing the layout. Neither GA or TS can be applied to this system easily. In this paper, we present a relatively simple greedy approach that works surprisingly well under this representation.

## 2 Rectilinear Representation and Heuristics

We first adopted a rectilinear representation for objects to be packed. The major reason is due to its simplicity - like integer coordinates and easy checking for overlapping, etc. After the rectilinearization, objects become more regular (though we will lose some information). Hence it becomes easy to perform advanced heuristic methods, like GA and TS.

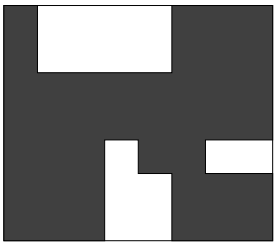
### 2.1 Object Representation

Our objective is to find a good arrangement of a set of arbitrarily shaped objects such that the bounding rectangle of this arrangement is minimized. The objects are represented as a matrix (two-dimensional array), which is a discrete approximation of the actual object shape. This "rectilinearization" process is shown in Figure 1. Note that there will be a digitization error if the object is not rectilinear.

Thus each object has four orientations: $0^\circ$, $90^\circ$, $180^\circ$ and $270^\circ$. Rotation is done by simply modifying the matrix and updating the height and width of the object.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |

Object Representation



Physical Layout

**Figure 1. Rectilinear Representation**

There are three advantages for us to use this representation.

1. Any arbitrarily shaped object, including convex and concave, etc, can be represented easily.
2. Rotation and checks for overlapping are easy.
3. Advanced heuristics (GA & TS) can be applied.

On the other hand, there are also two major disadvantages.

1. Complexity of representation and computation grows quadratically with the size of each object.
2. Only four orientations are possible.

## 2.2 Greedy Method

With rectilinear representation of objects, our heuristic first starts with a greedy approach. For a given set of $n$ objects, the following steps are executed:

```
Array N[1...n] contains all objects;
for each element i in N
  push N[i] into Q;
while(Q is not empty){
  for each element e in Q
    for each unpacked object o in N;
       combine(e, o) and
 record the best combination BC;
  push BC into Q;
}
```

We use the goodness factor for an object (or a set of combined objects) as a measurement for the quality of packing. The goodness factor is dependent on two values. The first value is the total number of edges, which is defined to be the total number of turns when going along the contour of an object, which is shown in Figure 2. The inner holes will

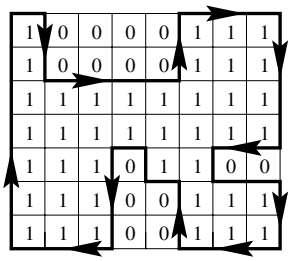


**Figure 2. Counting number of edges**

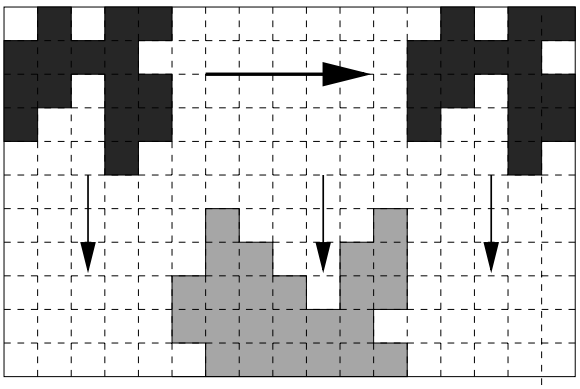


**Figure 3. Combining of objects**

be ignored. The rationale is that the lesser the edge number is, the better the packing is. The rectangle has four edges, which is the smallest. The second is the measure of wasted area, which can be obtained by counting the empty cells of the minimum bound rectangle of such configuration. Since our objective is to improve the utilization of the container, higher weightage may be set for the number of wasted area.

The combined function will return the best combination for two objects. The general idea is likened to the game of Tetris. Firstly, the two objects are rotated. For each one of the sixteen possible combinations, one object is fixed first (we call it $A$), the other object (we call it $B$) is placed above the first object. Then object $B$ is tries to drop down until it touches object $A$ from all possible positions. The Figure 3 illustrates this process. The grey object is $A$ and the black is $B$. For each position leftward, object $B$ will drop down until $A$ and such combination is evaluated.

### 2.2.1 Compaction

We realized that the packing configuration generated by combining can be further improved via compaction. Our compaction routine tries to move every object left-most and down-most, starting with the right-upper-most object. This is as if there are physical forces pushing the objects left and

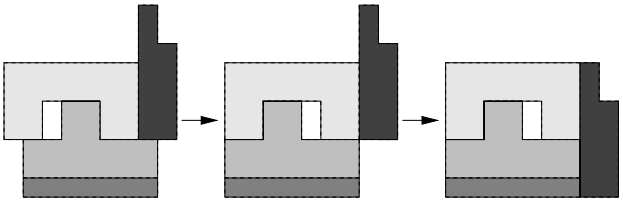


**Figure 4. Compacting of objects**

down. First we start with the right most object. When the object encounters a obstacle, both the object and the obstacle will be linked together as a group and move left together. The size of the group grows as we proceed. The same procedure will be applied for moving objects from top to bottom. Figure 4 shows how the compaction method helps to improve the result.

#### 2.2.2 Experimental Result

The experiments are conducted on sets of objects. Figure 5 demonstrates an example of the packing of $T$ shaped objects. This configuration is in fact a little better than the result obtained by Sakait and Hae [16] using Genetic Algorithms. Note there are four empty spaces in the figure, three of which are together. In Sakait and Hae's work, the four empty spaces are separated.

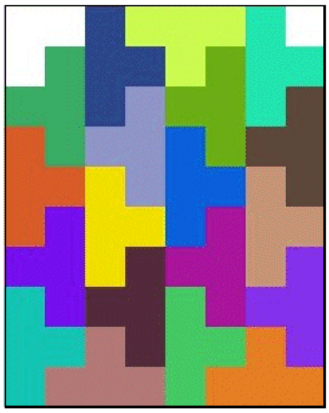


**Figure 5. Packing of $T$ Shapes**

Table 1 compares the results of packing different objects. It turns out that the greedy approach is able to generate a good result in a short time if the objects are simple, like T-shapes or rectangles.

| Test | Description | LB | Result | Util% | Time |
|---|---|---|---|---|---|
| 1 | Rectangles | 120 | 14×10 | 86 | 1,948 |
| 2 | Slicing Rectangles | 42 | 7×6 | 100 | 852 |
| 3 | T-shapes | 76 | 10×8 | 95 | 2,695 |
| 4 | Simple shapes | 345 | 23×18 | 84 | 11,236 |
| 5 | Complex shapes | 280 | 22×18 | 71 | 7,377 |

**Table 1. Some results obtained by the Greedy approach (LB:Lower Bound)**

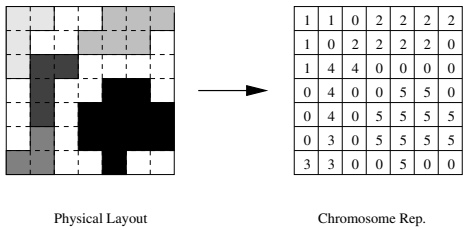



**Figure 6. Chromosome (solution) Representation**

### 2.3 Genetic Algorithms

Genetic Algorithms [8] [3] are heuristic search algorithms based on the mechanics of natural selection. An initial population is randomly generated. A pair of members are randomly chosen from the population for reproduction. The parent chromosomes (solution) are mutated and crossed over to generate the child's chromosomes (solution). All the child chromosomes undergo a natural selection, called "survival of the fittest". This whole process continues until a satisfactory solution is found.

#### 2.3.1 Solution Representation

Obviously, our matrix (2-D array) representation of object arrays can be easily mapped into a 2-D chromosome. (see Figure 6)

Each cell in the chromosome can take a value of 0 up to the number of objects. A 0 means the corresponding pixel is currently empty and a number $x$ indicates that the pixel is occupied by object $x$.

Standard GA operators will not work on this problem. We modified the approaches proposed by Sakait and Hae [16] both in the GA operators and compaction.

#### 2.3.2 Mutation

The idea of mutation is to make an "accidental" change in the chromosome. Hence, in the actual implementation

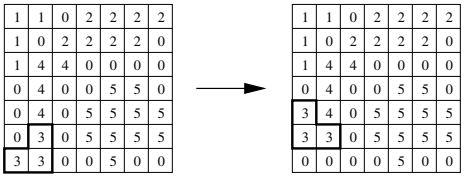


**Figure 7. Mutation**

of our GA, mutation is done by changing the position of one object in the solution. First a randomly selected object is removed from its current position. It will be randomly placed in another position with a randomly chosen orientation without any overlapping with other objects. Figure 7 demonstrates the mutation process. The neighborhood solution is obtained by changing the position and orientation of object 3.

### 2.3.3 Crossover

We believe that good solutions have certain good genetic characteristics. By crossing over two randomly selected parents, the next generation will be able to inherit these good characteristics. Or simply, the crossing over of good solutions is more likely to generate even better solutions.

The implementation of a crossover operator is more complicated than mutation. The general idea is to swap parts of the parents to produce children. Two randomly selected parents are overlaid to select the largest common area that can be swapped. A reconciliation is done after the swapping to ensure the validity of the solution. The process of the crossover operator is shown in Figure 8. The maximum common area (MCA) is surrounded by thick lines. After swapping the MCA, if an object appears twice in a child, a copy of such an object is removed. If an object does not appear in a child, a copy of the object will be randomly added into the child.

### 2.3.4 Objective Function

To measure the quality of one solution during the GA process, we defined the objective function as $\alpha\times$ edges-no $+(1-\alpha)\times$ wasted-area and $\alpha \leq 0.5$. We believe that this measurement is more accurate than the one used by Sakait and Hae [16], which is the sum of area of minimum bounding rectangle and the moment of inertia, in the sense that it also takes the number of edges into consideration. Usually, a highly packed set of objects have less number of edges than the loosely packed ones. Whereas [16] utilizes objects that do not deviate significantly from rectangles, our objects are generalised to be arbitrary rectilinear shapes, thus we have to take into account the number of edges in addition to wasted area.

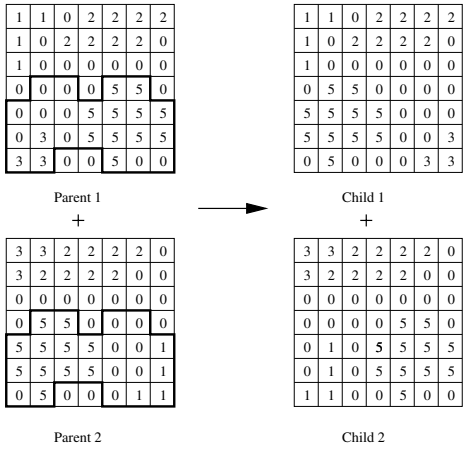



**Figure 8. Crossover**

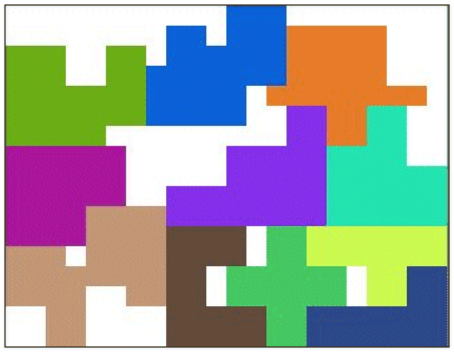


**Figure 9. Result of Genetic Algorithm**

### 2.3.5 Experimental Result

Table 2 shows the solutions generated by GA. GA takes up a large amount of time due to the genetic process. It may not work as well as the greedy approach for smaller test cases but it clearly out-performs the greedy approach for more generalized irregular shapes. Figure 9 illustrates the result of test case 5, which contains complex general shaped objects.

## 2.4 Tabu Search

Tabu Search [12] [10] is a local search heuristic that uses the best neighborhood move that is not in the list of "tabu" moves to move out of local optimum. The algorithm stops when a certain number of non-improving moves have been made. For more details, please refer to [9, 11].

| Test | Description | LB | Result | Util% | Time(s) |
|---|---|---|---|---|---|
| 1 | Rectangles | 120 | 13×10 | 92 | 56,693 |
| 2 | Slicing Rectangles | 42 | 7×6 | 100 | 7,767 |
| 3 | T-shapes | 76 | 12×8 | 83 | 18,695 |
| 4 | Simple shapes | 345 | 25×15 | 92 | 31,708 |
| 5 | Complex shapes | 280 | 22×17 | 74 | 87,280 |

**Table 2. Some results obtained by Genetic Algorithms (LB:Lower Bound)**

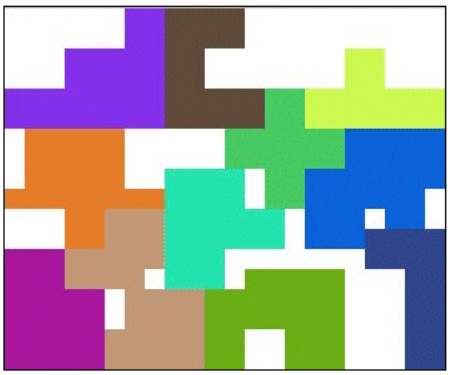


**Figure 10. Result of Tabu Search**

#### 2.4.1 Iterative Steps

In our implementation, the neighboring solution is defined to be any solution that can be obtained by changing the position of one object. For each iteration, every object with all possible new positions and orientations are evaluated. The best one is recorded and performed. The object involved in this operation is placed on the Tabu List, which usually has a length of seven elements in our implementation. The process stops when there are more than five non-improving moves.

#### 2.4.2 Experimental Result

Table 3 shows the results generated by the TS algorithm. The running time of TS changes from case to case due to the non-improvement move termination mechanism. It outperforms the greedy approach for more general cases. The overall performance of TS is comparable to that of GA. As a comparison to GA, Figure 10 illustrates the final configuration of test case 5 generated by TS.

| Test | Description | LB | Result | Util% | Time(s) |
|---|---|---|---|---|---|
| 1 | Rectangles | 120 | 13×10 | 92 | 10,233 |
| 2 | Slicing Rectangles | 42 | 7×6 | 100 | 387 |
| 3 | T-shapes | 76 | 11×9 | 77 | 9,499 |
| 4 | Simple shapes | 345 | 22×17 | 92 | 586 |
| 5 | Complex shapes | 280 | 22×18 | 71 | 8,944 |

**Table 3. Some results obtained by Tabu Search (LB:Lower Bound)**

## 3 Boundary Representation and Heuristics

### 3.1 Introduction of the Boundary Representation

The rectilinear representation we introduced in the previous chapter is based on the discretization of irregular objects. There are two disadvantages of such approaches. Firstly, for small objects, the discrete approximation will not be accurate as there will be obvious changes in the object shape. Secondly, for large objects, the discrete approximation will be more accurate as the changes are negligible to the size of the object. But the computational complexity grows quadratically $O(n^2)$ with the size of the objects, which seriously weakens the usability of such an approach.

The use of polygon boundary representation overcomes the above-mentioned shortcomings. In fact, such an approach is also more accurate due to the use of real number coordinates. However, the trade-off is that object manipulation is more difficult. Table 4 contrasts these two representation schemes. For example, to check whether two objects are overlapped, in the rectilinear representation we only need to check all the pixels. However, in the boundary representation, the computation that is needed to perform a polygon intersection checking involves checking the line segment intersection. There are several cases that need to be considered in line segment intersection checking. A function that handles line segment intersection with all cases exceeds a few hundred lines in code. And this is only the most basic problem in the boundary representation system! Because of its complexity there has been no attempt to solve irregular shaped packing.

### 3.2 The Representation and Operations

The boundary representation is straightforward as it simply stores the coordinates for the vertices of the polygon in a vector in some order. Note that such representation has the ability to represent both convex and concave polygons. Thus in the following, a general polygon can either be convex or concave.

| Operations | Rectilinear Rep. | Boundary Rep. |
|---|---|---|
| Move | pixel by pixel | compute final position and move at one shot |
| Rotation | 4 Orientations | arbitrary angles |
| Computing area | Scan for occupied cells in the matrix | Triangulation and sum up the areas |
| Checking for overlapping | Check for overlapping of two integer matrix | Checking for polygon intersection |

**Table 4. Compare and Contrast for Two Representation**

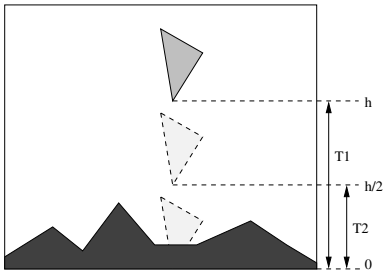



**Figure 11. Binary Search Dropping**

### 3.3 A Binary Move Technique

The move of the object is much more complicated than its rectilinear counterpart. Because the coordinates are real numbers, repeated shifting using a constant distance is not feasible as it is slow and inaccurate.

We proposed a binary search like method to move objects. For example, if object $O$ is at height $h$, to drop it downwards to a position as low as possible, given the bottom is not necessarily flat, we apply the following procedure:

```
Function Drop(H,L){
  while(H-L>10e-5){
    place O in height (H+L)/2;
    check for overlapping;
    if (there is overlapping)
      return Drop(H,(H+L)/2);
    else
      return Drop((H+L)/2,L);
  }
}
```

The function call starts with $Drop(h, 0)$, where 0 is the lowest point of the bottom which is definitely not feasible (see Figure 11). Then the algorithm will try ($T_1$) with the middle point of this interval and branch accordingly. In Figure 11, the middle point is feasible, the algorithm will

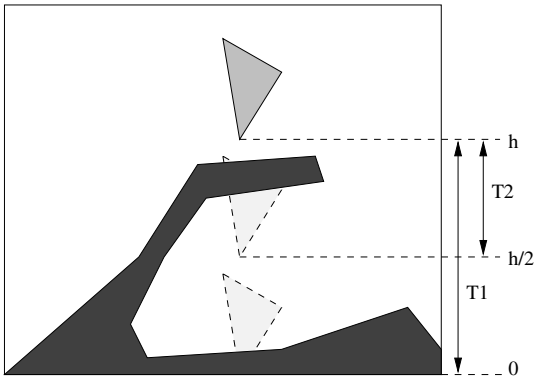



**Figure 12. A Trap During Binary Search Dropping**

then try ($T_2$) with the lower half. Finally, if the distance between $O$ and bottom is within $10^{-5}$ unit, we consider them to "touch" each other. However, we found that this binary search like technique does not work all the time. If there happens to be an obstacle placed in the middle of the dropping interval occurs, the algorithm will just simply ignore the lower half of the interval (See Figure 12). To overcome this problem, we allowed the algorithm to check both half of the interval to perform a more extensive search. In the code segment that follows, we use an additional parameter $R$ to control the number of tries. Another option to overcome the above problem is to divide the interval into more than two parts, for example using ternary search, etc. This approach can be implemented in a similar manner as the binary search based method.

```
Function Drop(H,L,R){
  while(H-L>10e-5){
    place O in height (H+L)/2;
    check for overlapping;
    if(R>0)
      return min(Drop(H,(H+L)/2,R-1),
           Drop((H+L)/2,L,R-1));
    else{
      if (there is overlapping)
        return Drop(H,(H+L)/2,0);
      else
        return Drop((H+L)/2,L,0);
    }
  }
}
```

### 3.4 The Greedy Approach

Advanced heuristic methods, like GA and TS, require an abstract solution representation to perform the search tech-

nique. However, it is not easy to find such abstractions under the boundary representation. Hence, in this section, we use a greedy based algorithm to solve the problem.

Greedy algorithms work in phases. In each phase, a decision is made that appears to be good, without regard for future consequences. Generally, this means that some local optimum is chosen. This "take what you can get now" strategy is the source of the name for this class of algorithms. When the algorithm terminates, we hope that the local optimum is equal to the global optimum. If this is the case, then the algorithm is correct; otherwise, the algorithm has produced a sub-optimal solution. If the optimal answer is not possible because of the NP-Completeness of the problem, then simple greedy algorithms are used as a simple heuristic to generate reasonable solutions.

### 3.4.1 The Algorithm

We have three ways to move to the local optimum, namely shifting down-most, shifting left-most and rotation. The algorithm works as follows:

```
sort the objects;
for each object (O){
  while(a position change of object O){
    shift O downwards as low as possible;
    rotate O around each vertex;
    shift O as left as possible;
  }
}
```

The process will continue until shifting-down, shifting-left or rotation is no longer possible. In other words, the algorithm continues pushing objects downwards and leftwards until the objects reach stable positions. The use of the rotation operator in this process is to ensure that the highest point of the object being pushed is kept as low as possible. In this way, the objects will be compacted along the vertical dimension.

Since the algorithm is greedy in nature, the order of the objects being packed will essentially affect the packing result. We proposed three packing sequences and did experiments for all of them.

1. Pack object with largest area first.
2. Pack object with smallest area first.
3. Pack objects in random order.

Figure 13, 14 and 15 list the packing results for these three orders. It is obvious that packing the largest area first will generate the highest utilization. The major reason is that the empty spaces produced by the big objects are more likely to be filled by the smaller objects later. However, the empty spaces produced by smaller objects will not be used as the objects being packed later become larger and larger.

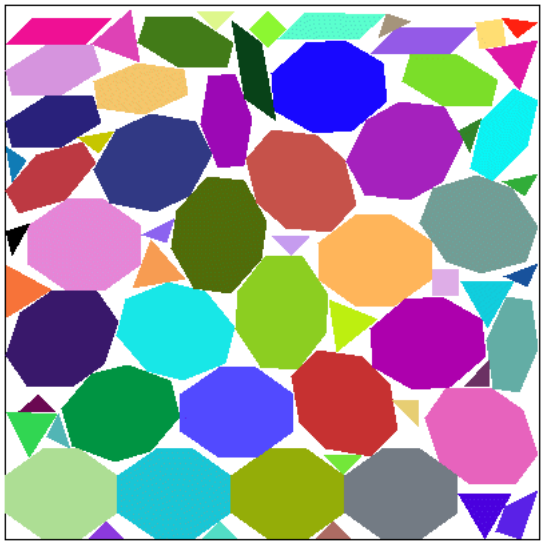


**Figure 13. Largest area first. Utilization: 80%**

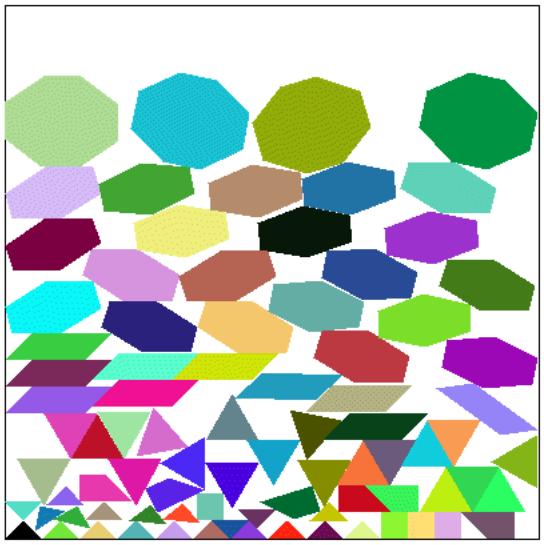


**Figure 14. Smallest area first. Utilization: 60%**

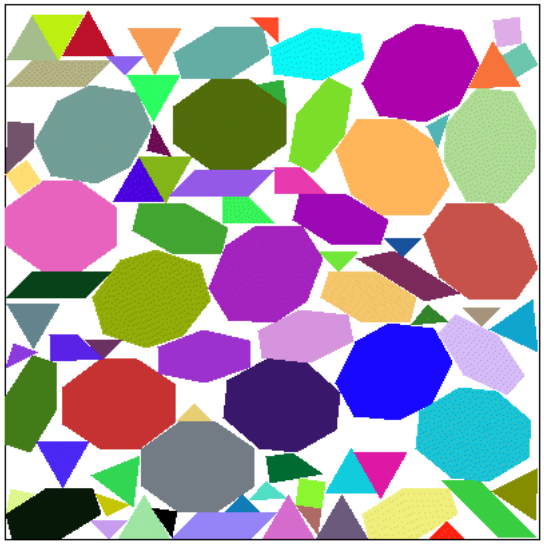


**Figure 15. Packing in random order. Utilization: 74%**

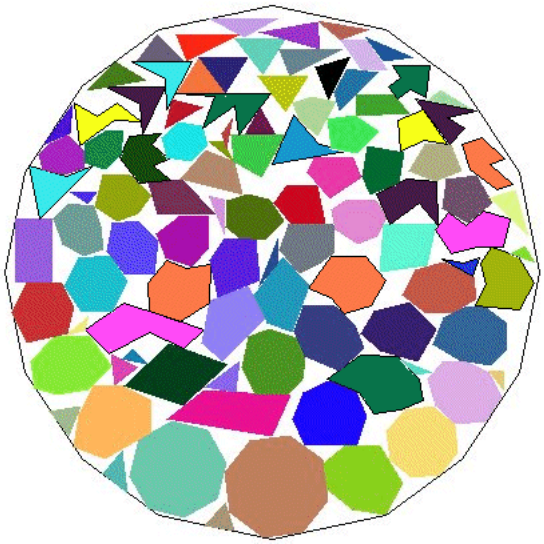


**Figure 16. Packing with convex container**

### 3.4.2 Experimental Result and Analysis

We performed extensive experiments over synthetic test cases to test our greedy packing approach with boundary representation. Figure 16 and 17 are the packing configurations for the mixture of convex and concave objects. It is true that for some portions of the configuration, human manipulation could improve the packing density. Note that such operation as a program involves a huge amount of artificial intelligence.

## 4 Conclusions

We proposed several methods for the 2D packing of general irregular shaped objects. This has not been well studied in the past because of its complexity. We started our research by adopting the rectilinear representation. Representing objects by a matrix (2D array) enabled us to perform advanced heuristic methods. We first came up with a Greedy algorithm by combining the objects. A Genetic Algorithm with modified Mutation and Crossover operators and Tabu Search algorithms were proposed and then tested for synthetic test cases with general shaped objects. It turned out that such representation had two limitations. For small objects, the discrete approximation error was large and for big objects, it was computationally inefficient. Polygon boundary representation was our second approach for representing general irregular shaped objects. This is more accurate, but is also more complicated. We designed the move, rotation, area and overlapping check operators as ba-

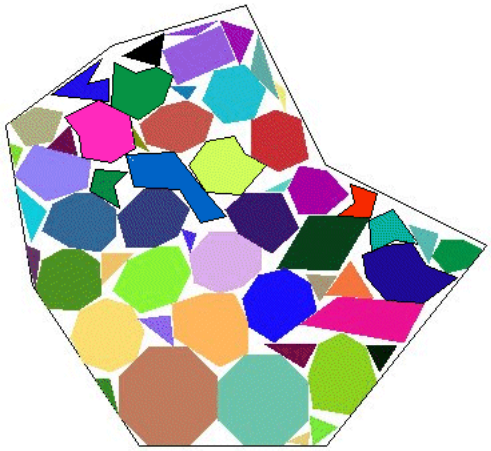


**Figure 17. Packing with concave container**

sic elements for manipulating objects. It seemed very difficult to apply advanced heuristic methods. As a first attempt on this problem, we designed a greedy method to solve the problem. From the experimental results, when the objects to be packed are more rectilinear-like, rectilinear representation is good. On the other hand, boundary representation is more powerful when the objects are highly diverse in shape.

## References


[1] M. Adamowica and A. Albano. Nesting two dimensional shapes in rectangular modules. *Computer Aided Design*, pages 27–33, 1976.

[2] A. Albano and G. Sapuppo. Optimal allocation of two-dimensional irregular shapes using heuristic search method. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 242–248, 1980.

[3] T. Back. Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms. 1996.

[4] H. P. Blazewicz, J. and R. Walkowiak. Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Annals of Operations Research*, pages 313–327, 1993.

[5] D. Dori and M. Ben-Bassat. Efficient nesting of congruent convex figures. *Communications of the ACM*, pages 228–235, 1984.

[6] K. Dowsland and W. Dowsland. Heuristic approaches to irregular cutting probelms. *Working Paper EBMS*, 1993.

[7] H. Freeman and R. Shapira. Determining the minimum area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, pages 409–413, 1975.

[8] D. Ginat. *Genetic algorithm : a function optimizer*. Imprint College Park, MD : University of Maryland, 1988.

[9] F. Glover. Tabu search. *Technical report, University of Colorado*, 1988.

[10] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, pages 190–206, 1989.

[11] F. Glover. Tabu search : a tutorial. *Interface*, pages 74–94, 1992.

[12] A. Hertz and D. D. Werra. The tabu search metaheuristics: How we used it. *Annals of Mathematics and Artificial Intelligence*, 1991.

[13] S. N. Keishi, S. and K. Yoji. The multi-bsg: stochastic approach to an optimal packing of convex-rectilinear blocks. *ICCAD98*, pages 267–274, January 1998.

[14] Z. Maggie and W. Wayne. Arbitrary rectilinear packing based on sequence pair. *ICCAD98*, pages 257–266, 1998.

[15] W. Qu and J. Sanders. A nesting algorithm for irregular parts and factors affecting trim losses. *International Journal of Production Research*, pages 381–397, 1987.

[16] J. Sakait and C. Hae. Two-dimensional packing problems using genetic algorithms. *Engineering with Computers*, pages 206–213, 1998.