GRASP Approach for the Glass Cutting Problem

Abhradeep Guha Thakurta[1], Ashwath Kumar.K[2], and Vijayan Immanuel[3]

[1] National Institute of Technology Karnataka, Surathkal, Srinivasnagar-575025, India
abhra80@rediffmail.com
[2] National Institute of Technology Karnataka, Surathkal, Srinivasnagar-575025, India
kashwathkumar@gmail.com
[3] National Institute of Technology Karnataka, Surathkal, Srinivasnagar-575025, India
vijayan@nitk.ac.in

Abstract. This paper presents a greedy randomized adaptive search procedure(GRASP) for the Glass Cutting Problem. The Glass Cutting Problem deals with the placing of rectangular shapes of different sizes into a rectangular piece having a given width and given length. The length has to be minimized, thereby minimizing the wastage of glass which in turn maximizes the profit. This is an NP-hard problem.
Index Terms:GRASP;Glass Cutting Problem;Heuristics ...

1 Introduction

The industry of glass production [1] is composed of two mainly independent branches: one producing hollow glass (bottles, glasses, and other differently shaped products) and dealing with flat glass (for windows, mirrors, and so on). This paper deals with the flat glass production, where a large glass sheet is first produced, with an area of some tens of square meters, and individual glass pieces of various dimensions are then cut from this sheet.
A glass maker usually starts from a set of desired pieces that have been commissioned to him, and a set of sheets to cut from. Loss is created when a sheet can not be exactly covered by the desired pieces. Loss is constituted by small glass pieces, called scraps, that cant be utilized in any way.
Contemporary algorithms that deal with this problem mainly base on Heuristic approach or Adaptive approach. In this paper we present a GRASP (Greedy Adaptive Reactive Search Procedure) approach to the solution of the problem. We also claim that the settling time of this algorithm to optimal solution is lesser compared to the contemporary approaches.

Received 21/06/06
Accepted 03/10/06
Final version 11/10/06

2 Problem Description

2.1 Geometrical Description

The Glass Cutting Problem consists of finding the best way of placing a given set of n rectangular pieces i = 1, .., n of given heights and widths (hi,wi), without overlapping into a strip of width W and finite height H. We assume that the pieces have fixed orientation. An example is shown in Figure 1, in which several pieces have to be packed into a strip of width W = 40. The problem is NP-hard in the strong sense because the strongly NP-hard one-dimensional bin-packing problem can easily be transformed into the Two Dimensional Glass Cutting problem. Therefore most research effort has been focused on developing heuristic algorithms for this problem.

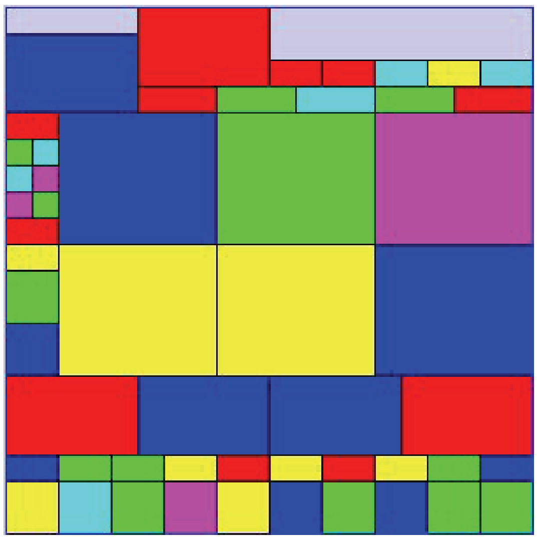


Fig. 1. A typical example of a solution to NP-Glass Cutting Problem

2.2 Glass Cutting Technology

In flat glass cutting [1] two separate steps are needed to extract a set of smaller pieces from a large sheet:

- First, a cutting pattern is scored on the glass sheet using a cutting wheel.
- Second, pieces are separated by using breakout bars, that hit the glass and create a mechanical stress in the neighborhood of one of the lines that were scored in the previous step. Internal glass tension breaks the glass in correspondence with the scored lines. Due to the critical breakout mechanism,

several constraints must be observed when scoring the cutting pattern, the most notable of which is that cuts must always be drawn from end to end of a glass piece.In practice, a cutting pattern is composed of a series of end-to-end vertical cuts (called X-cuts), breaking the sheet in a series of large stripes. Each stripe is then cut end-to-end in the horizontal direction (Y-cuts). This process may be repeated (W- and Z- cuts) in order to obtain the desired piece sizes.

– Third, a minimum distance between adjacent parallel cuts must be observed, in order for breakout bars to operate correctly.
– Fourth, the extreme borders of the sheet cannot be used due to irregularities: pieces must observe a minimum distance from the sheet boundary.
– Fifth, some sheets might be damaged in some internal point: no piece should be placed over these points.
– Sixth, a maximum distance between adjacent vertical parallel cuts must be observed, since the cutting machines have a vertical size smaller than the horizontal one and the pieces need to be rotated during breakout.

Because of all these constraints it is very difficult to make a Non-Adaptive Heuristic algorithm for the problem.

3 Constructive Algorithm

The algorithm adopted to solve the real life glass cutting problem is an iterative process which is called the constructive algorithm [2]. We follow an iterative process in which we combine two elements: a list P of pieces still to be packed, initially the complete list of pieces, and a list L of empty rectangles of infinite height in which a piece can be packed, initially containing only the strip S of width W. The rectangles will be denoted by the pair wi, li, where wi is the width and li is the level of its bottom side. At each step, a rectangle is chosen from L and, from the pieces in P fitting into it, a piece is chosen to be packed. That usually produces new rectangles going into L and the process goes on until all the pieces have been packed. The steps to be followed are as follows-

– Step 0: Initialization
  L = {S}, the set of empty rectangles.
  P = {p1, p2. . . pm}, the set of pieces still to be packed, ordered by non-increasing wi. Ties are broken by non-increasing hi.
  Qi is the number of pieces of type i to be packed.
  C =$\oslash$; the set of pieces already packed.

– Step 1: Choosing the rectangle in L
  Take R* = {w*, l*}, where l* = min {li | {wi, li} ε L}.
  Ties are broken by minimum wi. If none of the remaining pieces can fit into R*:
  - L is updated by lifting the bottom side of R* to the minimal level li of its adjacent rectangles and merging R* with the rectangle(s) of minimum level

li.
- That leaves a closed rectangle below which is considered waste.
- Go back to select a new R*.
Otherwise, go to Step 2.

– Step 2: Choosing the piece to pack
Once a rectangle R* has been chosen, consider the pieces i of P fitting into R* in order to choose which one to pack. For each of these pieces i we compute mi= max{m ε Z+ | m*wi $\leq$ w*m $\leq$ Qi}, the number of copies of piece i fitting into the width of R*, and consider all possible blocks of 1, 2, ..,mi adjacent copies as alternatives to fill R*. The height of a block composed of k copies of piece i is hi and its width k * wi. All these alternatives form the set P*. For the sake of simplicity, all the elements of P* will be called pieces from this point on.
Several criteria have been considered to select the piece to pack:
1. Piece j with maximum width wj , breaking ties by non-increasing hj .
2. Max{wj + 0.1 * hj}
3. Max{wj + 0.5 * hj}
4. The piece j whose height hj is more similar to the difference between the bottom side of R* and the bottom sides of one of the adjacent rectangles.
The first three criteria are based on the width, trying to fill the bottom of rectangle R* as much as possible. Each one of them gives a different importance to the height of the pieces. The fourth criterion tries to maintain a profile of the current solution which is as smooth as possible, avoiding peaks and troughs. However, all these criteria may delay the packing of tall pieces which will cause large increases in the required height H at the end of the process. In order to avoid this situation, we complement these criteria by computing a double estimation of the effect of not packing the tallest remaining piece. When we select a piece, according to the chosen criterion, before packing it we check if it is the highest remaining piece fitting in R*. If that is the case, we place the piece into the strip. Otherwise, we do a double computation:

We put the tallest piece j into the strip and see if that piece increases the current required height H. If it does, we determine the empty area, E, defined by the new height and compare it with the area of the pieces still to be packed, M, plus an estimation of the unavoidable waste involved in the process: $U = (W * LB - A)/4$, where LB is a lower bound on the required length and A is the total area of the pieces. If E > M + U, the tallest piece j is selected for packing and mj copies of it are packed into R*. Otherwise, we compute the second estimation.
We put the selected piece into the strip and then we put the tallest piece j in one of the other rectangles of L or on top of the selected piece, wherever it produces the minimum required length. We repeat the argument of the first estimation, decide if the tallest piece j is preferred for packing and then

pack mj copies of it.

- Step 3: Rotation and optimizing
  Once the step 2 is completed, the rectangle can be positioned in two ways, one with its longer side as the base and other with its smaller side as the base. We have to compute the efficiency in both the cases, and adopt the posture giving more efficiency.
  The Heuristics used can be represented as follows
  1. After finding a rectangle from the strip database, the algorithm checks if the rectangle fits in the upright manner. If it does not fit in, the rectangle has to be flipped and then the optimal placement has to be carried out.
  2. The end result expected after this computation is the height must be minimized.

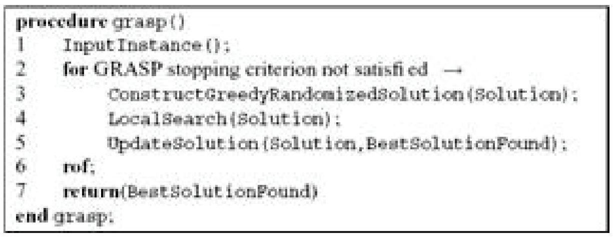
- Step 4: Choosing a position in R* to pack the piece
  Usually the piece to be packed does not completely fill rectangle R*. Therefore we have to decide its position inside R*. Obviously, the piece will be at the bottom of R*, but its position on the left or on the right hand side of R* has to e determined. Let us denote by Rl and Rr the rectangles of L adjacent to R* on its left and on its right.
  1. If Rl = ⊘ (Rr = ⊘), the piece goes on the left (right) hand side of R *. If Rl = Rr = ⊘, the piece is placed on the right hand side.
  2. Otherwise, we take into account the levels of Rl and Rr, ll and lr:

  If l* + hi = ll (l* + hi = lr), the piece is placed on the left (right) hand side.
  If ll = lr, the piece is placed as near as possible to one strip side.
  Otherwise, the piece is put adjacent to the rectangle with maximum level.

- Step 5: Updating the lists C = C ∪ {pi}
  Make Qi = Qi k, where k is the number of copies of the piece i forming the block chosen to be packed. If Qi = 0, remove piece i from the list P Add the new rectangles to L. Merge two rectangles if they are adjacent and have the same level.

4 GRASP Algorithm

GRASP has a strong intuitive appeal, a prominent empirical track record, and is trivial to efficiently implement on parallel processors [3]. GRASP is an iterative randomized sampling technique in which each iteration provides a solution to the problem at hand. The incumbent solution over all GRASP iterations is kept as the final result. There are two phases within each GRASP iteration: the first intelligently constructs an initial solution via an adaptive randomized greedy function; the the second applies a local search procedure to the constructed solution in hope of finding an improvement. In this case GRASP is an iterative



```
procedure grasp()
1    InputInstance();
2    for GRASP stopping criterion not satisfied →
3        ConstructGreedyRandomizedSolution(Solution);
4        LocalSearch(Solution);
5        UpdateSolution(Solution,BestSolutionFound);
6    rof;
7    return(BestSolutionFound)
end grasp;
```

Fig. 2. Generic GRASP Pseudocode

procedure combining a constructive phase and an improvement phase. In the constructive phase a solution is built step by step, adding elements to a partial solution. In order to choose the element to be added, a greedy function is computed, which is dynamically adapted as the partial solution is built. However, the selection of the element is not deterministic but subjected to a randomization process. In that way, when we repeat the process, we can obtain different solutions. After each constructive phase, the improvement phase, usually consisting of a simple local search, tries to substitute some elements of the solution which are there as the result of the randomization, by some others, producing an overall better solution.

4.1 Constructive Phase

In our algorithm the constructive phase corresponds to the constructive algorithm described in Section 3, introducing randomization procedures when selecting the piece to pack. Let si be the score of piece i ε P* on the selection criterion we are using and smax=max{ si| i ε p* },and let δ be a parameter to be determined (δ lies between 0 and 1). We have considered four alternatives:

- 1. Select piece i at random in set
  C = { j | sj = (smin + δ (smax - smin)) }
  (C is commonly called a Restricted Set of Candidates).

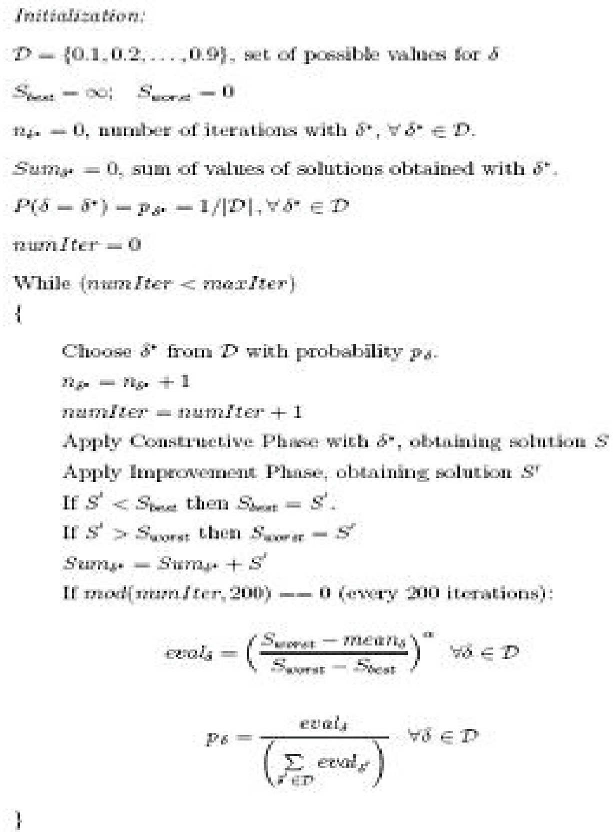
- 2. Select piece i at random in set
  C = {j | sj = (smax) }

- 3. Select piece i at random from among the best
  100 (1 - δ)% of the pieces, irrespective of their score.

- 4. Select piece i from among the whole set P* but
  with probability proportional to its score si (pi = si/Σsj).

Using one of these randomization procedures on one of the selection criteria described above, a piece is chosen to be packed at each step of the constructive procedure. Nevertheless, the estimations of the effect of the tallest piece are also taken into account. These estimations are also randomized by using a parameter

γ. If $E > \gamma * (M + U)$, he tallest piece j is selected and a number of copies randomly chosen between 1 and mj is packed. At each iteration the parameter δ is randomly chosen in the interval (0.9, 1.6). A preliminary computational study showed that the term M + U tends to underestimate the total area which will be required by the remaining pieces and therefore the value of γ should oscillate above the value 1, though we also allow it to be slightly lower than 1.



```
*Initialization:*
$\mathcal{D} = \{0.1, 0.2, \ldots, 0.9\}$, set of possible values for $\delta$
$S_{best} = \infty$;   $S_{worst} = 0$
$n_{\delta^*} = 0$, number of iterations with $\delta^*$, $\forall \delta^* \in \mathcal{D}$.
$Sum_{\delta^*} = 0$, sum of values of solutions obtained with $\delta^*$.
$P(\delta = \delta^*) = p_{\delta^*} = 1/|\mathcal{D}|, \forall \delta^* \in \mathcal{D}$
$numIter = 0$
While $(numIter < maxIter)$
{
    Choose $\delta^*$ from $\mathcal{D}$ with probability $p_{\delta}$.
    $n_{\delta^*} = n_{\delta^*} + 1$
    $numIter = numIter + 1$
    Apply Constructive Phase with $\delta^*$, obtaining solution $S$
    Apply Improvement Phase, obtaining solution $S'$
    If $S' < S_{best}$ then $S_{best} = S'$.
    If $S' > S_{worst}$ then $S_{worst} = S'$
    $Sum_{\delta^*} = Sum_{\delta^*} + S'$
    If $mod(numIter, 200) == 0$ (every 200 iterations):

        $eval_{\delta} = \left(\frac{S_{worst} - mean_{\delta}}{S_{worst} - S_{best}}\right)^{\alpha} \quad \forall \delta \in \mathcal{D}$

        $p_{\delta} = \frac{eval_{\delta}}{\left(\sum_{\delta' \in \mathcal{D}} eval_{\delta'}\right)} \quad \forall \delta \in \mathcal{D}$
}
```

Fig. 3. The GRASP Procedure for Glass Cutting Problem

4.2 Determining the parameter δ

A preliminary computational experience showed that no value of always produced the best results. Therefore, we considered several strategies basically consisting of changing the value of δ randomly or systematically along the iterations. These strategies were:

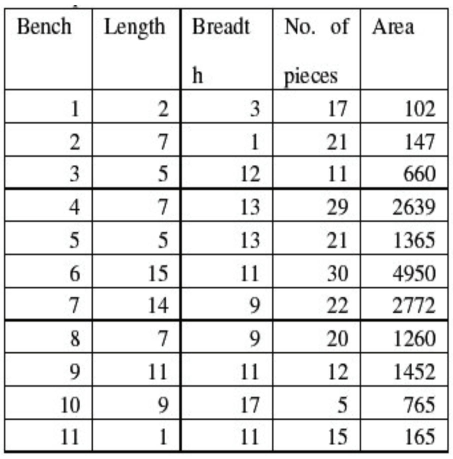
- 1. At each iteration, choose δ at random from the interval [0.4, 0.9]
- 2. At each iteration, choose δ at random from the interval [0.25, 0.75]
- 3. At each iteration δ takes one of these 5 values in turn: 0.5,0.6,0.7,0.8,0.9.
- 4. $\delta = 0.75$
- 5. Reactive GRASP
  In Reactive GRASP δ is initially taken at random from a set of discrete values, but after a certain number of iterations, the relative quality of the solutions obtained with each value of δ is taken into account and the probability of values consistently producing better solutions is increased.

5 Experimentation and Comparison with other Contemporary Algorithms

To experimentally evaluate the algorithms performance, in terms of result quality we selected some real benchmarks [5] and a comparison of the results with the set of commercial optimizers and our algorithm is tabulated.

5.1 Benchmarks

The characteristics of benchmarks used are reported in Figure 4. For each benchmark, the number of different piece sized and the total number of pieces are reported. The reported area is the sum of the areas of all the pieces. The bench-



Bench	Length	Breadt h	No. of pieces	Area
1	2	3	17	102
2	7	1	21	147
3	5	12	11	660
4	7	13	29	2639
5	5	13	21	1365
6	15	11	30	4950
7	14	9	22	2772
8	7	9	20	1260
9	11	11	12	1452
10	9	17	5	765
11	1	11	15	165

Fig. 4. Results of the Experimentation

marks selected are regarded as real cases since they represent actual data used by some glass makers. Further, they represent critical data, since they are some cases for which glass makers complained aboutthe optimization efficiency. The total glass area . The glass piece taken given initially is 135*135, the total area of all the pieces is found to be 16277 m2. The area wastage is found to be (3+12+15+14) m^2 , i.e 44 m^2 . The distribution given by the software given is as shown in figure 6. The study of the parameter δ in comparison with other contemporary algorithms give the following results.

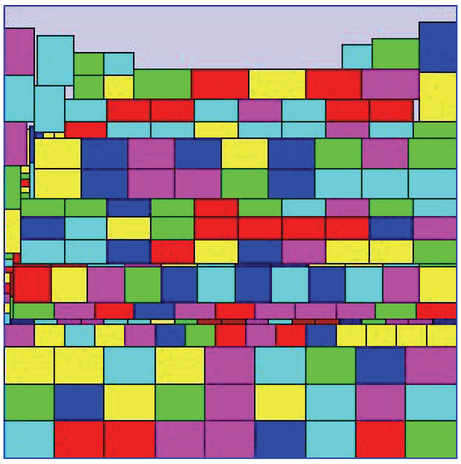


Fig. 5. Distribution Result

	Average deviation from bound		
	Literature	*Martello et al. and Berkey et al.*	*Hopper*
Random in [0.4, 0.9]	2.01%	2.15%	2.76%
Random in [0.25, 0.75]	2.03%	2.67%	3.22%
Deterministic from 0.5 to 0.9	1.95%	2.06%	2.83%
Fixed to 0.9	2.95%	2.01%	3.18%
Reactive Grasp	1.93%	2.03%	2.84%

Fig. 6. Delta Comparison

6 Conclusion

We have modified the conventional GRASP and used a Heuristic approach for the Glass Cutting Problem. The complete algorithm obtains good results on large sets of fairly randomized test cases. The algorithm is quite flexible and can be easily adapted to accommodate other conditions or constraints. Future work will involve the design of more efficient procedure to check the existence of a feasible packing layout, in order to take more advantages on the less number of feasibility tests performed.

References

1. The handbook of glass manufacturing, Haslee Publishing, NY USA.
2. BrazilJo.o P. Marques Silva and Karem A. Sakallah, GRASP A New Search Algorithm for Satisfiability.
3. Jakob Puchinger, Gunther R. Raidl,and Gabriele Koller, Solving a Real-World "Glass Cutting Problem
4. R. Alvarez-Valde , F. Parreno and J.M. Tamarit, Reactive GRASP for the Strip Packing Problem.
5. Zhang, D., Kang, Y., Deng, A. (2005) A New heuristic recursive algorithm for the Strip rectangular packing problem, Computers and Operations Research,in press.
6. Youzou Fukagaway, Yuji Shinanoz, Tomoyuki Wadaz, Mario Nakamori, Optimum Placement of a Wafer on a Rectangular Chip Grid. Proceedings of 6th World Congresses of Structural and Multidisciplinary Optimization Rio de Janeiro, 30 May - 03 June 2005.