A Genetic Algorithm for Packing Three-Dimensional Non-Convex Objects Having Cavities and Holes

Ilkka Ikonen*
William E. Biles
Department of Industrial Engineering
University of Louisville
Louisville, KY 40292

Anup Kumar
Rammohan K. Ragade
John C. Wissel
Engineering Math and Computer Science Dept
University of Louisville

Abstract

In this paper we describe a unique three-dimensional bin-packing problem with non-convex parts having holes and cavities. Parts can be arranged in any orientation and location in the packing cylinder, where parts float like in a weightless environment. The solution approach utilizes a genetic algorithm (GA) which represents a solution for the problem as a three-dimensional chromosome. Each dimension in the chromosome is an ordered list of integers. As a function evaluator we have developed a packing simulator, which utilizes actual CAD file of parts, when packing parts. Part intersection calculations are based on methods common in computational geometry.

Keywords: genetic algorithms, 3D bin packing, non-convex objects, selective laser sintering, rapid prototyping

1 INTRODUCTION

1.1 SELECTIVE LASER SINTERING RAPID PROTOTYPING TECHNOLOGY

Like all current rapid prototyping (RP) techniques, the selective laser sintering (SLS) machine produces parts directly from CAD files through an additive manufacturing process. The part is modeled as a solid three-dimensional (3D) CAD model, which is "sliced" into thin (0.003" - 0.02") horizontal slices which are manufactured one at the time. Before slicing, the solid model is converted to an STL file format which is an industry-standard interface between a 3D model and a rapid prototyping machine. The STL file is a tessalated approximation of the part being prototyped, where each surface of a solid model is replaced by a set of triangles. The size and number of triangles depend on the desired accuracy and the file size. When viewed as a text file, an STL file consists of x,y,z-coordinates indicating vertexes of triangles. When a triangle is cut into horizontal slices, points on a horizontal plane are achieved. These points are used to toggle the laser beam on and off, as required to create the cross sections of parts at each layer. Selective laser sintering uses fine, heat-fusible powder to build a part. The powder is melted by a CO_2 laser beam which follows the shape of the cross section of the part being built. The intensity of the laser beam is controlled to melt the powder only in areas defined in the cross section. Unmelted powder outside part boundaries remains in the build cylinder (15 inches high, 12 inches diameter), providing support for the layers above. Due to this supporting powder, parts can be arranged in any orientation and location inside the cylinder. No support structures are needed to hold them in place. Parts "float" in the powder as they would in a weightless environment.

After the laser has sintered the current layer, the

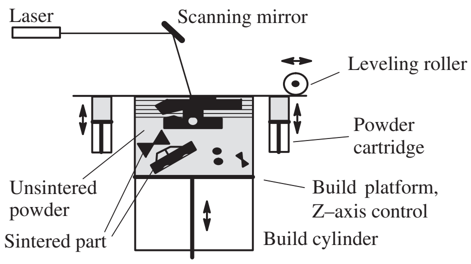

Figure 1: Selective Laser Sintering Process

*Corresponding author, email to: itikon01@starbase.spd.louisville.edu

building platform is lowered one layer thickness inside the cylinder and a new layer of powder is spread on the top of the previous one by a leveling roller. The fresh powder is in turn sintered as a new layer is added to the part. Layer-by-layer sintering continues until all parts in the batch are finished. The building platform is elevated from the cylinder, causing loose powder to fall away and reveal the finished prototypes. The selective laser sintering machine has very few restrictions in the part shapes which it can produce. Most prototyped parts are used in industry and are therefore complex in shape, having irregular surfaces, cavities and holes.

1.2 RELATED EARLIER RESEARCH

The part placement problem in an SLS machine is a form of the bin packing problem, which is a classical operations research problem. Bin packing problems have been defined in literature in several different forms depending on the specific application [Dyckhoff, 90]. Some of the various names this problem is given are: stock cutting, vehicle loading, pallet loading, nesting, knapsack problem and scheduling. Most of these problems, and their respective solution approaches, are restricted to one or two dimensions. When solving three-dimensional problems, researchers many times simplify their approach and handle the problem as a two-dimensional case. Several palletizing algorithms are a good example for this simplification.

Researchers have used several different exact and heuristic methods to solve these problems, one of them being genetic algorithms [Sweeney, Paternoster, 92]. In [Jakobs, 96], the author uses GA to pack rectangular pieces in two dimensions as the same problem arises in steel industry. The same problem is defined in [Kroger, 95], in [Ismail, Hon, 95] and in [Hwang et al., 94]. A similar problem for a two-dimensional placement application is solved in [Williamson, Watson, 94], in [Shahookar, Mazumder, 90] and in several other papers in the field of integrated circuit layout design. For the three-dimensional packing problem, there have been several GA applications. In [Corcoran, Wainwright, 92] and in [House, Dagli, 93] authors describe a GA approach to solve a rectangular 3D packing problem. In [Juliff, 93], the author uses a multi-chromosome approach to solve a truck-loading problem. A GA application for packing part bounding boxes in a stereolithography (SLA) rapid prototyping machine is described in [Wodziak, Fadel, Kirshman, 94].

All the previously published approaches make some assumptions about the shape of parts to be packed. They also have several constraints (bottom-left etc.) which do not apply to the SLS packing problem. Our GA approach does not have these restrictions. It is able to represent the actual part geometry and places no unnecessary restrictions to solution.

2 CHROMOSOMAL REPRESENTATION

The chromosomal representation chosen for a genetic algorithm should translate the under lying real world problem as accurately as possible into a number code which can be manipulated by genetic operators. All the needed information for the problem should be included and represented in the selected chromosomal representation. For a three-dimensional packing problem with non-convex parts and no gravity in the build cylinder, there are several pieces of information that have to be included in the chromosome to make meaningful representation possible. Also, the selected representation should not restrict solutions in any artificial way by presenting constrains that do not exist in the actual packing problem. One of these is the 'bottom-left' condition, which is required in most traditional packing problems. Another restriction is how parts relate to each other. In most traditional packing problems, parts are located next to each other in some systematic way, such as "part A is left from part B", "A is on top and right from B" or "B is behind, right and below of A". In the SLS packing problem due to support powder there is no need to have a 'bottom left' requirement. Also, parts are non-convex and many of them have holes and cavities where other, smaller parts can be packed. This makes description of relations between parts very complicated using above described traditional relations. The selected chromosomal representation does not include these constraints and allows total freedom of parts and search.

Three pieces of information are required to create a good packing from given parts. The first one is the order in which parts are placed inside the cylinder. Intuitively one would place large parts first and fill the remaining space with smaller ones which might fit into cavities of larger parts. We are not using part size information directly and we do not sort parts in any way before packing. This is, however, an important aspect of a packing problem and it is left to the GA to find a good permutation of parts.

A second important piece of information is the orientation of each part. Each part has a local coordinate system associated with them. Rotation of a part occurs around one of these local coordinate axes. For a human operator, there are no restrictions in the number of orientations in which parts are placed inside the cylinder. Moreover, the SLS RP machine places no restrictions on orientation it is able to manufacture parts. However, to reduce the search space, number of allowed orientations is reduced to 24. Each part is free to rotate in 45 degree increments around each coordinate axis (3x360/45 = 24).

The third data requirement is how parts relate to each other. In a good packing parts are most likely next to each other and smaller parts are inside holes of larger parts. When two parts are well packed, they have some features that are close to each other, i.e. outer corners are near each other or outer corner of one part is next to a corner of a hole in other part. To represent this, so called 'attachment points' are defined for each part. Use of 3D points to indicate where a part can be located is used also in [Autere, 92]. These points are defined by the human operator using a CAD software which can display three-dimensional coordinates of important features. These features are, for example: outer upper right front corner, bottom left front corner of a cavity or a corner of a hole. For each part, five attachment points are specified. Five different possibilities for part attachment is sufficient to give enough freedom for part relations while keeping the search space smaller. Attachment points of the previously packed part define points in three dimensional space where the next part can be located, or 'attached'. Points are selected so that cavities and holes can be utilized. To reduce part intersection and represent actual packing more realistically, each attachment point is offset from the part surface by some small value (0.1"). Attachment points are stored in a separate attachment point file where each attachment point is represented by three coordinate values similar to the STL file.

Each chromosome is a list of three sublists of integers. The first sublist is a permutation of part numbers being packed, gene values being between 0 ... n-1, where n is the number of parts to be packed. Only one of each part numbers can exist in the list. The second sublist is the orientation list, which indicates the orientation of each part. Gene values are between 0 ... 23 and any value can be present more than once. The third sublist is the list of attachment points for each part. Values are between 0 ... 4 and a value can be used more than once. For example, if we are packing five parts, part numbers run from 0 to 4 and one chromosome could be:

List1: 4 3 0 2 1
List2: 6 9 22 1 17
List3: 2 3 0 4 1

This represents packing where part number 4 is packed first in orientation 6 (rotated 270 degrees around x-axis). The first part is attached to the global origin. Then part number 3 is packed in orientation 9 (rotated 45 degrees around y-axis) and is attached to attachment point number 2 on previously packed part. Part number 0 is in orientation number 22 (rotated 270 degrees around z-axis) and attached to attachment point number 3 in previous part. Part number 2 is in orientation 1 (rotated 45 degrees around x-axis) and attached to attachment point number 0 in previous part. The last part, number 1, is packed in orientation 17 (rotated 45 degrees around z-axis) and is attached to attachment point number 4 in the previous part.

3 CROSSOVER AND MUTATION

Genetic algorithm uses two basic operators, crossover and mutation, to create new individuals from old ones. After two individuals are selected for crossover, each sublist in them is handled separately by these operators. Standard ordered list crossover operators can be used for the first sublist, but for two other lists operators have to be modified to allow same allele values in offsprings (i.e. two parts can have the same orientation or attachment point numbers). Changes to the operators are kept in minimum to allow otherwise normal crossover. Traditional swap mutator performs well for part number list, but does not suffice for two other sublists. For the orientation list several allele values are not present in a short list like the one above. In a relatively small population (50 individuals) there might be only few instances for some orientation values and if these happen to be in an unpromising packing, these values are removed from gene pool too fast. To introduce orientation gene values back to the population, a higher mutation rate is required for orientation list. After some initial testing a mutation rate at 0.2 was found to perform well. Even though this is very high when compared to previously found good mutation rates (0.001-0.05) [Goldberg, 89], it does not distract the search and is able to introduce enough new orientation values to the gene pool. The mutator is a destructive one; it destroys an allele value and creates a new one randomly from a given range (0 ... 23). A similar operator is used for attachment point list. The mutation rate for this sublist is kept low (0.01).

4 FITNESS FUNCTION

A genetic algorithm requires a goodness value for each candidate packing to guide the search into a promising direction. The approach used in this paper is so called penalty-function method where unfeasible solutions are given a penalty to indicate how much defined constraints are violated. The use of penalty functions in genetic algorithms were studied in detail in [Smith, Tate, 93], in [Richardson et al., 89] and in [Michalewicz, Janikow, 91]. The packing simulator is given one chromosome (packing) at the time to get evaluated. It packs parts in order, orientation and in location as specified by chromosome. After all parts are packed, the simulator calls an evaluator function which calculates three different values to indicate how good the packing is. Calculated values include (1) the sum of distances of parts from the global origin, (2) the amount of intersection between parts, and (3) the amount of intersection between parts and the build cylinder. The calculation of intersection between parts is based on intersection of triangles on parts. Two or more parts intersect only if included triangles intersect. To calculate the extent to which triangles in three-dimensional space intersect, several steps and levels of calculations are required. Calculations follow methods used in computational geometry [Laszlo, 96]. To reduce the number of triangles for triangle intersection calculations, the bounding boxes of parts and triangles are first checked for intersection. The bounding box of a geometric object is the smallest box that contains the object, where the edges of the box are parallel to the coordinate axis. Objects intersect only if their bounding boxes intersect.

First the bounding boxes of parts are checked; if they are found to intersect, the triangles of parts are then checked for intersection. This requires pair-wise comparison of the bounding boxes of triangles. Triangles whose bounding boxes are found to intersect are checked for intersection. Calculation of intersection for two triangles in three-dimensional space requires checking of line-triangle intersection where the line is one edge at the time of the other triangle. Method *TriInt* below calculates how much of each edge of one triangle is through the other [Ikonen, Biles, 97]. This is done for each pair of triangles and parts whose bounding boxes intersect. Returned real values are added as an indication of total part intersection for packing. Part intersection routine can be written as a following pseudo code:

s = number of parts to be packed
t_i = number of triangles in part i
BB_i = bounding box of part i
bb_i = bounding box of triangle i
$BB_i \cap BB_j$ = intersection between bounding boxes i and j
$TriInt_{mk}$ = method to calculate triangle intersection between triangles m and k

procedure **PartIntersection**
FOR i:= 0 TO s-2 **DO** {
  FOR j:=i+1 TO s-1 **DO** {
    IF $BB_i \cap BB_j \neq \emptyset$ **THEN** {
      FOR k:=0 TO $t_i - 1$ **DO** {
        FOR m:=0 TO $t_j - 1$ **DO** {
          IF $(bb_k \cap bb_m \neq \emptyset)$ **THEN** {
            calculate $TriInt_{mk}$
            }
          ELSE
        }
      }
    }
  }
}

RETURN

$$\sum_{i=0}^{s-2} \sum_{j=i+1}^{s-1} \sum_{k=0}^{t_i-1} \sum_{m=0}^{t_j-1} TriInt_{mk}$$

The second value calculated by the packing evaluation function is the overall distance of parts from the global origin, which is located at the bottom and center of the build cylinder. Distance from the origin of a part is calculated using coordinates of the center of its bounding box. A sum over all the distances represents how tightly parts are packed. Containment inside the cylinder is checked for the bounding box of each triangle. If the distance on horizontal plane from the center of the cylinder to the maximum dimension of bounding box is greater than the radius of the cylinder, the triangle is not inside the cylinder and the intersection counter is incremented by one. Minimum and maximum Z-dimensions are checked at the same time.

These three indicators are used to calculate the overall evaluation of packing. These values are multiplied by factors and added to calculate the total evaluation of packing. The values for factors depend which one of the three values is the most significant for each packing problem. The objective function and the optimization problem can be formulated as follows:

α, β, γ = weight factors

$\sum_{i=0}^{s-1} D_{i0}$ = sum of distances of centers of part bounding boxes to the global origin

$\sum_{i=0}^{s-2} \sum_{j=i+1}^{s-1} \sum_{k=0}^{t_i-1} \sum_{m=0}^{t_j-1} TriInt_{mk}$ = sum of triangle intersections between parts

$\sum_{i=0}^{s-1} ContInt_i$ = number of triangle bounding boxes that are not totally contained in the container.

MINIMIZE

$$\mathbf{F}(ContInt_i, TriInt_{mk}, D_{i0}) =$$

$$\alpha * \sum_{i=0}^{s-1} ContInt_i \ + \ \gamma * \sum_{i=0}^{s-1} D_{i0} \ +$$

$$\beta * \sum_{i=0}^{s-2} \sum_{j=i+1}^{s-1} \sum_{k=0}^{t_i-1} \sum_{m=0}^{t_j-1} TriInt_{mk}$$

5 THE SEARCH SPACE

The search space within which the genetic algorithm explores is highly discontinuos and multi-modal; therefore, the search can easily be trapped in a local optimum and has no method to check if the global optimum is found. Small changes in part orientation can move the search deep into the unfeasible region. Similarly, an unfeasible solution can be changed to a good, if not optimum, packing by changing only one or two orientations or relative locations of parts. In geographic terms, hills and valleys in the search space have steep slopes and the highest peak can be located next to the lowest valley. It can be assumed that the search space looks like the one for test function F5 which De Jong used in his famous research [Goldberg, 89]. The shape of the search space depends, however, on the shape, size and similarity of the parts to be packed. If the parts to be packed are simple and similar in shape, the slopes in the search space should become less steep. This is obvious if one thinks about packing of solid blocks where a good packing can be easily found. Good packing pattern will be orthogonal, and any other part orientations will evaluate lower score depending on how much and how many parts are not oriented according to pattern. When more complex parts are packed and they are all different in shape, the search space becomes more unfriendly. For example, if parts have holes and smaller parts can fit inside these holes, a small change in a chromosome can make a great difference in the packing evaluation. Parts which are slightly disoriented from a good packing plan can produce a very low score. How well a genetic algorithm can search such a space depends on the evaluation function used and how well it is able to differentiate between promising and poor intermediate solutions.

6 INITIAL TESTING

The first tests of the proposed GA were performed to find a good generation replacement scheme and parameter values. Traditional simple GA, steady-state GA and incremental GA approaches were tested to find out which of them would perform the best. As a fast test problem, a simpler evaluation function was used. This was done to speed up initial testing. The test problem tries to sort each chromosomal sublist depending on the other two lists. This mimics the actual packing problem relatively well, where good gene values in each sublist have to be found simultaneously. The evaluation function used here can be described as a following pseudo code:

Given a chromosome:
List 1: $\{A_i\} : \{A_0, A_1, A_2, A_3, A_4\}$
List 2: $\{B_i\} : \{B_0, B_1, B_2, B_3, B_4\}$
List 3: $\{C_i\} : \{C_0, C_1, C_2, C_3, C_4\}$

```
FOR i := 0 TO 4 DO{
    if (A_i = i) score1 += 3
    if (A_i + B_i = 17) score2 +=3.1
    if (A_i + C_i = 4) score3 +=3
    }

RETURN α * score1 + β * score2 + γ * score3
```

Unlike the actual packing problem this is a maximization problem. Factors α, β *and* γ in the evaluation function were found through trial and error. Weight factors were introduced to balance differences in the hardness of scoring functions. If the factors were set to 1 the first scoring problem (to sort List1) was the easiest and was the only one satisfied. Good values for the initial test problem were found to be: $\alpha = 1.3, \beta = 2$ *and* $\gamma = 1$. The steady-state GA with generational replasement = 0.1 performed the best and was able to find the optimum in most cases in 300 generations when the population size was 50 individuals. The known optimum is

List1: {0, 1, 2, 3, 4}
List2: {17, 16, 15, 14, 13}
List3: {4, 3, 2, 1, 0}

It was observed that even small changes in the weight factors of the evaluation function (α, β, γ) make a big difference to the results an algorithm is able to obtain. Especially, the second scoring function was harder to satisfy than two others. This is due to larger search space and the fact that in a short list there are only few of the possible gene values present. This was corrected by higher mutation rate. Other parameters for the GA were kept the same during this research. These were: crossover probability = 0.9, roulette wheel selection, linear fitness scaling with selection pressure set to 1.1. Several different ordered list crossover operators were tested (order crossover, PMX and cycle crossover). This seemed to have little, if any, influence on the search results. PMX crossover was used. Figure 2 shows generation best, average and worst over 500 generations for the test sorting problem. The search has converged to the optimum after 300 generations. Population size was 50.

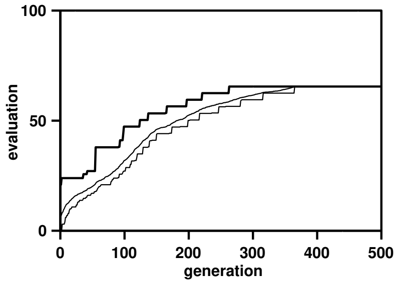

Figure 2: List Sorting Problem

7 PACKING OF SOLID BLOCKS

The first tests with actual parts that have an STL file associated with them were with solid rectangular blocks. Solid blocks are the simplest part type that can be packed in the RP machine, having 12 triangles in STL file. They can also be seen as bounding boxes for more complex part shapes. Five solid blocks of size 2" x 2" x 1" were packed inside a cylinder of size 4 " (height) and 3" (radius). Only a few parts and a small cylinder were used to make testing faster. The steady-state GA approach was able to find the best solutions. The quality of solutions found depended on the weight factors in the evaluation function. This is due to different value ranges for three different goodness indicators. The sum of distances has typically a value ranging from 10 to 30. Intersection has a value range of 0 to about 250. Containment evaluation has value range from 0 to 60. Weight factors are used to balance these differences and to give more weight to some aspect of packing. In most cases part intersection has the highest priority and weight value. After some testing it was noticed that the distance evaluation distracts the search. Better results were obtained if the weight factor γ was set = 0, i.e. distance has no importance in this problem. If a solution with evaluation = 0 is found, it is guaranteed to be a good packing. With correct selection of factors ($\alpha = 1.5, \beta = 4$ and $\gamma = 0$) GA is able to find a non-overlapping part packing where all parts are located inside the cylinder and are close to the global origin. The small cylinder size was used to create enough large differences between individuals and to apply enough selection pressure towards better individuals. If the packing problem is too easy, i.e. the build cylinder is too large for few small parts, there is no significant difference between packings and the GA is unable to differentiate between promising and unpromising regions. Figure 3 represents a graph of the

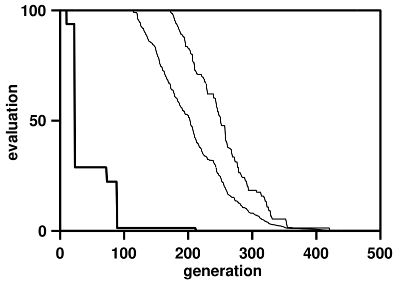

Figure 3: Packing of Five Solid Blocks

best, average and worst in a population for a typical search over 500 generations with a population size of 100. In most runs a good, non-overlapping packing was found. A high mutation rate (0.2) for the orientation gene does not disrupt the search. The genetic algorithm is able to steadily improve population average during the search. This can be seen as an indication that the search space is relatively easy for a GA.

8 PARTS WITH HOLES AND CAVITIES

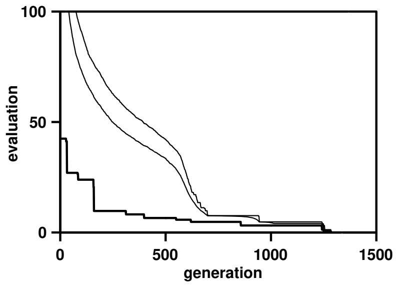

Figure 4: Packing of Parts with Holes

Figure 4 plots the generation best, average and worst evaluation for a typical run for more complicated part geometries. Seven parts are packed, two of which have holes and other parts fit inside these holes (and should be packed inside them in a good packing plan). Bin size is kept small to guide the search. If a non-intersecting and totally contained packing is found, it has to be very tight. Again, the steady-state GA performs the best and the search converges relatively rapidly (within 1500 generations) to one packing. However, population size has to be increased. Population size = 300 was required to find a non-intersecting and totally contained packing. Good weight factors were found to be $\alpha = 0.5, \beta = 1.3$ and $\gamma = 0$. As the

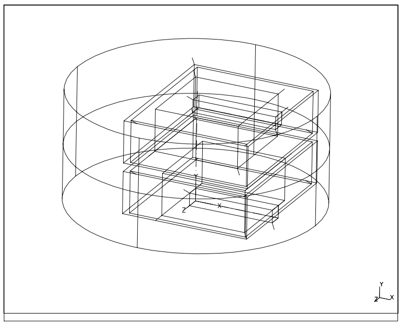

Figure 5: Packing Solution for Parts with Holes

number of triangles changes with parts, weight factors have to be adjusted accordingly. Previously described parameters for the GA worked well except for the linear scaling selection pressure. Better performance was obtained if this was set to lower value (1.01) reducing the convergence pressure. With higher value only a local optimum was found. Figure 5 shows a typical packing solution which is found by the GA.

9 CONCLUSIONS AND FUTURE WORK

We have described a unique three-dimensional packing problem which has several special features that have not been described in the previous literature. The proposed solution approach uses genetic algorithms and actual three-dimensional part geometry to find a good, if not optimal, solution. A simple-list sorting problem was used to find initial parameter values for the genetic algorithm. Initial experiments with simple 3D parts (solid blocks or bounding boxes for more complex parts) are promising. Starting with a randomly generated initial population search convergence to a solution within 500 generations with a small population size (100 individuals). Quality of the packing found depends heavily on factors employed in the evaluation function. Correct weight factor values depend on number of parts, part geometry and the number of triangles in parts. Packing with more complex parts (holes and cavities) is a much more difficult problem than packing of solid blocks.

Future work will consist of experiments with more difficult part geometries to see how well a GA is able to pack parts that are typically manufactured in RP industry. Based on these results more detailed analysis on the search space in hand can be obtained. Setting values for weight factors can be automated based on the geometry of parts. We are also working on improvements in the speed of the search to make routine use more feasible. Figure 4 shows a typical GA search behavior: the best in the population improves fast at the beginning of the search but settles after a while and is unable to improve untill a good mutation occurs. There are two ways to correct this: include a hill climbing method with the GA which will be applied after the GA is unable to improve fast enough, or increase the mutation rate towards the end of the search to increase number of new gene values in the population. We are also working on use of distributed GA in a parallel processor computer.

References

[Autere, 92] Autere, Antti, GA and 3D Bin Packing, Proceedings of the First Finnish Workshop on Genetic Algorithms and their Applications, Nov 4-5, 1992, Helsinki University of Technology, Finland

[Corcoran, Wainwright, 92] Corcoran, Arthur L. III, Wainwright, Roger L., A Genetic Algorithm for Packing in Three Dimensions, Symposium on Applied Computing (SAC 92), Kansas City March 1-3, 1992, pp. 1021-1030.

[Dyckhoff, 90] Dyckhoff, Harald, A typology of cutting and packing problems, European Journal of Operations Research 44, 1990, pp.145-159.

[Goldberg, 89] Goldberg, David, Genetic algorithms in search, optimization and machine learning, Addison-Wesley, 1989.

[House, Dagli, 93] House, L. Randy, Dagli, H. Cihan, Genetic Three Dimensional Packer, in Intelligent Engineering Through Artificial Neural Networks, vol 3, pp. 837-842, ASME Press, 1993.

[Hwang et al., 94] Hwang, Shian-Miin; Kao, Cheng-Yan; Horng, Jorng-Tzong, On Solving Rectangle Bin Packing Problems Using Genetic Algorithms, in Proceedings of IEEE International Conference on systems, man and cybernetics, vol.2, Oct 1994, San Antonio, Texas, pp. 1583-1590.

[Ikonen, Biles, 97] Ikonen, Ilkka, Biles, William E., A Genetic Algorithm for Optimal Object Packing in a Selective Laser Sintering Machine, in Proceedings of the 7th Int. Conf. on Flexible Automation and Intelligent Manufacturing (FAIM), June 25-27, 1997, Middlesbrough, UK

[Ismail, Hon, 95] Ismail, H.S., Hon, K.K.B, The nesting of two-dimensional shapes using genetic algorithms, Proc Instn Mech Engrs, vol 209,1995, pp. 115-124, Inst of Mech Eng, 1995.

[Jakobs, 96] Jakobs, Stefan, On Genetic Algorithms for the packing of polygons, European Journal of Operation Research 88, 1996, pp.165-181.

[Juliff, 93] Juliff, Kate, A Multi-Chromosome Genetic

Algorithm for Pallet Loading, in Proc. of the 5th International Conference on Genetic Algorithms, Urbana, Ill.,1993, Kaufman Publishers, pp. 467-473.

[Kroger, 95] Kroger, Berthold, Guillotinable bin packing: A genetic approach, European Journal of Operation Research 84, 1995, pp.645-661.

[Laszlo, 96] Laszlo, Michael J., Computational Geometry and Computer Graphics in C++, Prentice-Hall, 1996

[Michalewicz, Janikow, 91] Michalewicz, Zbigniew; Janikow, Cezary, Handling Constraints in Genetic Algorithms, in Belew, Richard; Booker, Lashon (Ed.), Proc. of the 4th International Conference on Genetic Algorithms, 1991, Kaufman Publishers.

[Richardson et al., 89] Richardson, Jon; Palmer, Mark, Liepinds, Gunar; Hilliard, Mike, Some Guidelines for Genetic Algorithms with Penalty Functions, in David Schaffer (Ed.) Proc. of the 3rd Int. Conference on Genetic Algorithms and Their Application, 1989, Kaufman Publishers.

[Shahookar, Mazumder, 90] Shahookar, K, Mazumder, P., Standard Cell Placement and the Genetic Algorithm, Advances in Computer-Aided Engineering Design, vol 2, pages 159-233, JAI Press Ltd, 1990.

[Smith, Tate, 93] Smith, Alice; Tate, David, Genetic Optimization Using a Penalty Function, in Eshelman, Larry (Ed.), Proc. of the 6th Int. Conference on Genetic Algorithms, 1993, Kaufman Publishers.

[Sweeney, Paternoster, 92] Sweeney, Paul; Ridenour Paternoster, Elizabeth, Cutting and Packing Problems: A Categorized, Application-Oriented Research Bibliography, Journal of Operations Research, vol. 43, no. 7, 1992, pp.691- 706.

[Williamson, Watson, 94] Williamson, A.G., Watson, K., Optimizing Flexible Manufacturing System Layout with Genetic Algorithms, Factory 2000-Advanced Factory Automation Conference, 3-5 October 1994, Conference Publication No 398, IEE 1994, pp. 12-18.

[Wodziak et al., 94] Wodziak, John R., Fadel, George M., Kirschman, Chuck, A Genetic Algorithm for Optimizing Multiple Part Placement to Reduce Build Time, proceedings of the 5th International Conference on Rapid Prototyping, pp. 201-210, June 12-15, 1994, Dayton, Ohio.