DETERMINING AN OPTIMAL SOLUTION TO A THREE DIMENSIONAL PACKING PROBLEM USING GENETIC ALGORITHMS

DONALD YING
STANFORD UNIVERSITY
dying@leland.stanford.edu

ABSTRACT

This paper determines the plausibility of using genetic algorithms as a method for finding optimal solutions to a three dimensional packing problem. Specifically, it will look at the problem of putting together a cube that has been partitioned into six pieces. Search spaces consisting of the locations and orientations of the set of six pieces will be represented by strings of bits. These search spaces will then be sifted through by genetic algorithms in order to attempt to put the cube back together from the six pieces. Various different representations and methods will be tried, with their outcomes discussed.

INTRODUCTION

During late February of 2002, I had partitioned a 3x3x3 cube into six pieces, consisting of a 3-polycube, a 4-polycube, and four 5-polycubes. After finding two solutions that were not trivially similar, I had passed this puzzle around, observing how people would approach the problem, and hoping to find a solution to the puzzle that I had yet to see before.

Eventually, many frustrated people had suggested using a computer to obtain other possible solutions to the puzzle. Seeing that writing a program that would exhaustively scan through all possible placements of the six pieces within a reasonable amount of time would have been a daunting task, I had turned to genetic algorithms to attempt to quickly search over the possible search space, in the hopes that an optimal solution will be found.

BACKGROUND

Genetic algorithms have been known to efficiently find good solutions in large search spaces. With their basic ideas lifted from biology, genetic algorithms use duplication, crossover, and mutation techniques to generate populations of strings representing points in a search space. Each string in a population is scored using a heuristic fitness function. The next population is then generated using ideas such as survival of the fittest to attempt to produce a population with a better average fitness score, while maintaining a good variation among the strings at the same time.

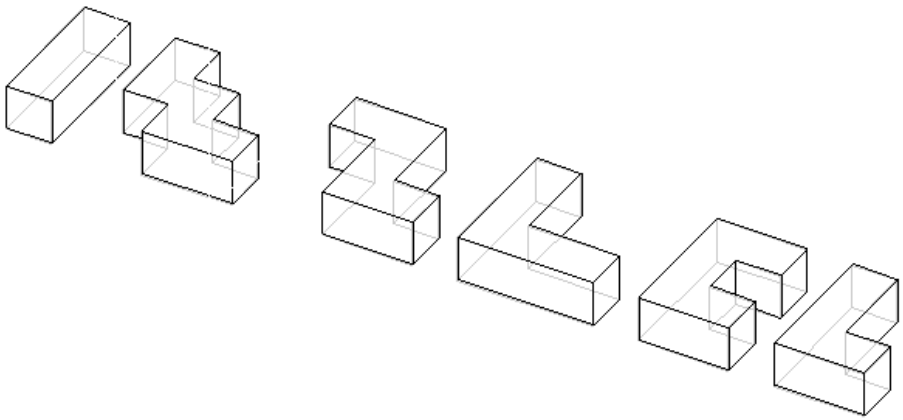

FIGURE 01: The Six Pieces Used to Construct a 3x3x3 Cube
From Left to Right: Piece I, Piece W, Piece S, Piece V, Piece U, and Piece L

STATEMENT OF PROBLEM

The specific three dimensional packing problem we will explore consists of the six polycube puzzle pieces shown in Figure 01. To refer to these pieces individually, each one has been given a name that refers to a Roman alphabet it closely resembles. The names are provided in the caption to the figure.

Restricting the orientation of the puzzle pieces to 90^0 rotations along the three primary axes and assigning one of the cubes in each of the polycube pieces to be the key cube in the piece, a total of 24 orientations with respect to the key cube can be obtained for each piece. A *packing* will then be defined as the assignment of an orientation for each piece, as well as a location for the key cube of each piece. The locations for the key cubes will be restricted to integer coordinates. The locations can either be defined as absolute coordinates, or as coordinates relative to a piece whose absolute coordinates have already been determined by the coordinates of itself and/or the previously placed pieces.

A *valid packing* is defined as a packing that can actually be modelled by the puzzle pieces, ie. a packing with no pieces overlapping each other. If the packing can be completely enclosed by a cube whose side length is less than or equal to the side length of the enclosing cube of any other packing, then the packing is called an *optimal packing*.

For example, since the six puzzle pieces in Figure 01 are a partition of the 3x3x3 cube, they can obviously be packed in such a way that the enclosing cube has side length 3. Obviously, since the total volume of the six pieces is 27, there can be no other packing with an enclosing cube of shorter length. Therefore, the packing of the pieces into a 3x3x3 cube is an optimal packing.

So far, I have discovered two optimal packings that are not trivially similar. They are illustrated in Figure 02. Due to chirality and rotational symmetry considerations, each of the two solutions actually represents 48 different but trivially similar solutions. Since the solution on the left utilizes Piece I in the center of the cube, while the solution on the right has Piece I on the corner of the cube, the solution to the left will be referred to as the Piece I in Center solution, and the other as the Piece I in Corner solution.

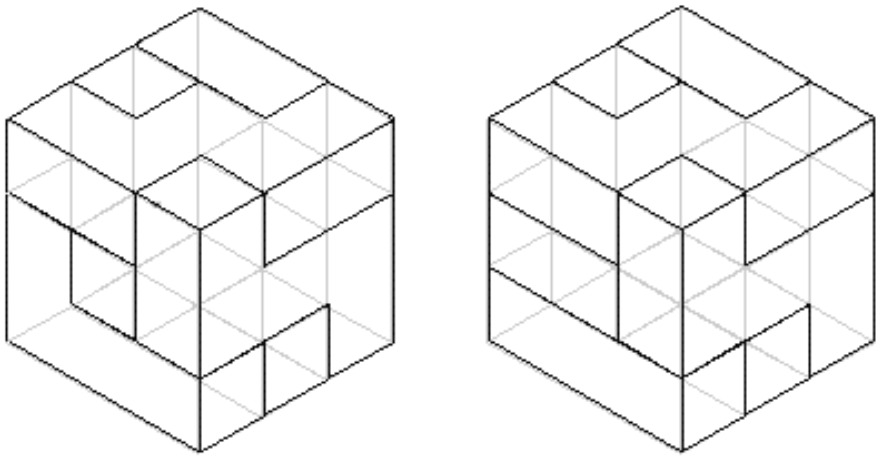

FIGURE 02: Two Distinct Optimal Solutions of the Three Dimensional Packing Problem
Piece I in Center solution on left, Piece I in Corner solution on right

It can be proven that the Piece I in Center solution, along with its 47 other related solutions, are the only possible solutions with Piece I in the center. Although the proof is not difficult, it is irrelevant to the main discussion of this paper, and is therefore left to the reader as an interesting exercise. However, it has not been proven whether or not the Piece I in Corner solution, along with its 47 trivially similar solutions, are the only solutions with Piece I not in the center.

A REDUCED VERSION OF THE PROBLEM: THE TWO DIMENSIONAL CASE

In order to test the feasibility of using genetic algorithms to solve this three dimensional packing problem, a greatly simplified two dimensional version of the problem was first formed. In this case, a 3x3 square is divided into three identical rectangular strips, each of size 1x3. Restricting the definitions of packing, valid packing, and optimal packing to two dimensions, one can easily see that an optimal packing of these three strips would have an enclosing square of length 3.

REPRESENTATION IN THE TWO DIMENSIONAL CASE

Defining a key square in each of the triomino pieces, there are a total of four different orientations for the triomino to assume with respect to its key square. We can interpret a number in the set {0, 1, 2, 3} as a specific orientation by assigning each number in the set to one of the four orientations.

Using a square with coordinates in {0, 1, 2} for both dimensions, the possible values of absolute coordinates of the first piece to be placed will be from {0, 1, 2} for both coordinates. If each subsequent piece is then placed with coordinates relative to the previous piece's key square, then a coordinate for any subsequent piece will be taken from the set {-2, -1, 0, 1, 2}. The following table summarizes the bits used in the representation string.

	First Piece		
	Absolute X-Coord	Absolute Y-Coord	Orientation
Range	0 to 2	0 to 2	0 to 3
Number of Bits	2	2	2

	Second Piece			Third Piece		
	Relative X-Coord	Relative Y-Coord	Orientation	Relative X-Coord	Relative Y-Coord	Orientation
Range	-2 to 2	-2 to 2	0 to 3	-2 to 2	-2 to 2	0 to 3
Number of Bits	3	3	2	3	3	2

FIGURE 03: Representation as Binary String in Two Dimensional Case

Obviously, the search space will be of size 2^{22}, or 4096, since the representation string is 22 bits long.

FITNESS IN THE TWO DIMENSIONAL CASE

The fitness function used for this algorithm was relatively simple. Since there are bit strings which represent values that are out of range (such as the bit string "110" for the relative x-coordinate of the second piece), these strings are docked with a huge penalty, chosen as 10,000,000 per invalid value in this case. Any string that represents a packing is then penalized for any part of a piece that stretched beyond the boundaries of the enclosing square of the optimal packing, with penalty:

$$p = m * ((\Delta x)^2 + (\Delta y)^2)$$

where m is the number of piece overlaps at a particular location outside the optimal enclosing square, Δx is the different in x-coordinates of that particular location from the closest edge of the optimal enclosing square, and Δy is defined similarly.

Note that the fitness function will not lie strictly in the interval [0, 1], as is sometimes the case for genetic algorithms. However, this restriction is

unnecessary since the software used does not rely on the fitness function lying in the unit interval.

PROGRAM OVERVIEW AND OUTCOME IN THE TWO DIMENSIONAL CASE

The genetic algorithm tableau for this reduced problem is shown in Figure 04.

Objective:	To find an optimal packing for the reduced two dimensional puzzle case.
Representation Scheme:	Structure: 9 variables K = {0, 1} (binary) L = 22
Fitness Cases:	The packing of the three triomino pieces with respect to the optimal enclosing square.
Fitness:	Function as described previously.
Parameters:	M = 500 G = 5000 $p_c = .6$ $p_m = .001$
Termination Criteria:	When fitness equals 0 or when all generations have been calculated.
Result Designation:	Structure with fitness equal to 0.

FIGURE 04: Tableau for the Two Dimensional Packing Problem

There was no specific reason to choose these values for parameters M, G, p_c, and p_m, other than the fact that they were used as default values before.

With this setup, the algorithm was able to produce an optimal packing by Generation 48. The success of this program leads us to believe that applying genetic algorithms to the three dimensional case seems plausible.

REPRESENTATION IN THE THREE DIMENSIONAL CASE

As mentioned before, a key cube in each of the polycube pieces will be defined, resulting in a total of 24 different orientations for the puzzle piece with respect to its key cube. We map these orientations to an integer from 0 to 23 inclusive.

Using a cube with coordinates in {0, 1, 2} in all three dimensions, the first puzzle piece can now be placed with absolute coordinates obtained from the set {0, 1, 2}. Subsequent pieces are placed relative to the previous piece's key cube, resulting in every subsequent piece's coordinate ranging from {-2, -1, 0, 1, 2}, as before. The table in Figure 05 summarizes the binary string used in the three dimensional case.

	First Piece				Second Piece			
	Abs X-Coord	Abs Y-Coord	Abs Z-Coord	Orientation	Rel X-Coord	Rel Y-Coord	Rel Z-Coord	Orientation
Range	0 to 2	0 to 2	0 to 2	0 to 23	-2 to 2	-2 to 2	-2 to 2	0 to 23
Number of Bits	2	2	2	5	3	3	3	5

	Third Piece				Fourth Piece			
	Rel X-Coord	Rel Y-Coord	Rel Z-Coord	Orientation	Rel X-Coord	Rel Y-Coord	Rel Z-Coord	Orientation
Range	-2 to 2	-2 to 2	-2 to 2	0 to 23	-2 to 2	-2 to 2	-2 to 2	0 to 23
Number of Bits	3	3	3	5	3	3	3	5

	Fifth Piece				Sixth Piece			
	Rel X-Coord	Rel Y-Coord	Rel Z-Coord	Orientation	Rel X-Coord	Rel Y-Coord	Rel Z-Coord	Orientation
Range	-2 to 2	-2 to 2	-2 to 2	0 to 23	-2 to 2	-2 to 2	-2 to 2	0 to 23
Number of Bits	3	3	3	5	3	3	3	5

FIGURE 05: Representation as Binary String in Three Dimensional Case

This time, the binary representation string is 81 bits long, giving a huge search space of 2^{81}. Obviously, an exhaustive search of this space using current computing facilities would not be able to complete such a search in a reasonable amount of time. This is why genetic algorithms will be used to attempt to solve this problem.

FITNESS IN THE THREE DIMENSIONAL CASE

The fitness function used for this algorithm parallels that of the two dimensional case. Again, bit strings that do not represent a packing of any kind due to range discrepancies will be docked with a huge penalty. Strings that represent packings are then penalized for pieces that protrude outside the boundaries of the enclosing cube of the optimal packing, with penalty:

$$p = m * ((\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2)$$

where m is the number of piece overlaps at a particular location outside the optimal enclosing cube, Δx is the different in x-coordinates of that particular location from the closest edge of the optimal enclosing square, and Δy and Δz are defined similarly.

Again, the fitness function will not lie strictly on the interval [0, 1], but as mentioned before, this is not a necessary restriction.

PROGRAM OVERVIEW AND OUTCOME IN THE THREE DIMENSIONAL CASE

The genetic algorithm tableau for the three dimensional packing problem is shown in Figure 06.

Objective:	To find an optimal packing for the three dimensional packing problem.
Representation Scheme:	Structure: 24 variables K = {0, 1} (binary) L = 81
Fitness Cases:	The packing of the six polycube pieces with respect to the optimal enclosing cube.
Fitness:	Function as described previously.
Parameters:	M = 500 G = 25000 p_c = .6 p_m = .001
Termination Criteria:	When fitness equals 0 or when all generations have been calculated.
Result Designation:	Structure with fitness equal to 0.

FIGURE 06: Tableau for the Two Dimensional Packing Problem

Although the parameter values for M, p_c, and p_m, shown in the tableau above, were chosen simply because they worked in the two dimensional case, other parameter values were also tried, with p_c ranging from .6 to .95, p_m from .0001 to .1, and M ranging up to 5000. G was increased to 25000 since no optimal packing solution presented itself within the first 5000 generations, as before.

However, using this structure and binary representation, I was unable to obtain an optimal solution, ie. a string with fitness value 0. The output of one program was able to produce a packing with fitness value 4, in which Piece I and Piece W both had one cube projecting out of the enclosing cube of the optimal packing.

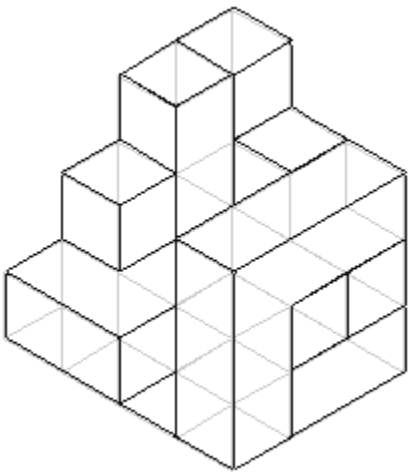

FIGURE 07: A Non-Optimal Valid Packing with Fitness 4

SPECULATIONS FOR REASONS OF FAILURE

It seems as if the main problem with solving this three dimensional packing problem with genetic algorithms is that there are too many local minima in the fitness function, while the global minimum may be nowhere near a local minimum. For example, in the optimal packings shown in the solutions of Figure 02, the relationship between Piece W and Piece U are the same. This relationship is shown in Figure 08. Although this relationship may not be necessary, since it has not been proven that no other optimal packing exist aside from those of Figure 02 and trivially similar ones, we may conjecture that this relationship is necessary due to the lack of evidence otherwise.

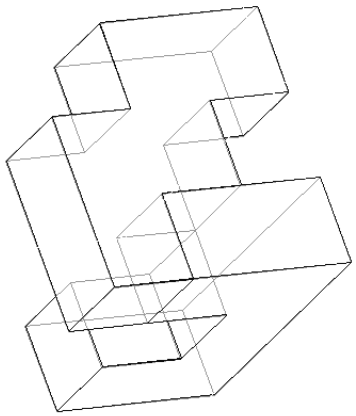

FIGURE 08: The Relationship Between Piece W and Piece U

When observing the fitness values of the best solution per generation obtained by the algorithm, one notices that during the first few generations, the best fitness value jumps down drastically. This shows how the invalid representations are first weeded out. Eventually, the valid representations get stuck in local minima, such as the one shown in Figure 07, and small values of p_m do not give the population enough variability to jump out of the rut. Of course, with large values of p_m, the population never hits a satisfactorily low fitness value in the first place, so there is a tradeoff in benefits and deficits when choosing a value of p_m.

However, once the population gets stuck at a local minimum, most of the population starts adopting the traits of the string with the best fitness value, due to crossover. Eventually, the relationships between pieces determined by the string with the best fitness value dominate the population, and if that local minimum did not have a relationship between Pieces W and U as seen in Figure 08, the algorithm may never terminate with a fitness value of 0.

REVISIONS TO THE ALGORITHM

Numerous other revisions to the genetic algorithm were eventually tried. For instance, a different representation that used absolute coordinates for the key cubes of each of the pieces was implemented, which resulted in a reduced search space of 2^{66}. The relationships between pieces were also not as rigid as before, since the piece locations were not defined relatively anymore.

Another revision, in which a large penalty was given to large pieces that protruded from the optimal enclosing cube, was also implemented. The idea behind this revision was that if a large piece would not fit in the optimum enclosing cube at first, it was highly unlikely that it would ever fit into the optimal enclosing cube.

A revision in which Piece I was fixed in the center of the cube was also tried, in order to attempt to reduce the size of the search space. Optimal packings obtained with this revision would, of course, be trivially similar to the Piece I in Center solution as shown in Figure 02.

Finally, a revision that simply fixed Pieces I, W, and S into part of an optimal packing was tried, and, when run with all the other revisions, the genetic algorithm was able to find an optimal packing in this reduced search space.

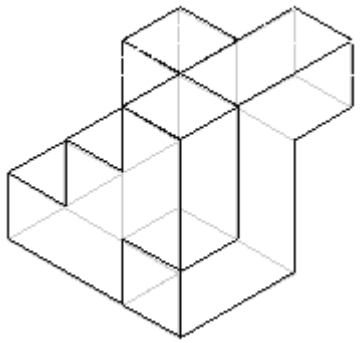

FIGURE 09: Fixing Pieces I, S, and W Into Position

FUTURE WORK

Although this paper specifically focused on the six polycube puzzle piece example and its optimal packings into a 3x3x3 cube, there are many other different examples of three dimensional packing problems in which similar genetic algorithms may be applied. For example, the luggage packing problem, in which small rectangular parallelepipeds, representing travel items, are packed into a larger rectangular parallelepiped, representing the luggage itself. Specific versions of the luggage packing problem are well-known in the literature, such as the Slothouber-Graatsma Puzzle, a partition of the 3x3x3 cube into nine

rectangular parallelepipeds, and the Conway Puzzle, which partitions the 5x5x5 cube into eighteen rectangular parallelepipeds.

It is also noted that although genetic algorithms may not be able to find the optimal solution to a hard three dimensional packing problem, such a program may be able to find packings that are near optimal. It may be interesting to see how such algorithms can be used efficiently in real-life packing problems in which an optimal solution is not required.

REFERENCES

Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, Massachusetts: Addison Wesley Longman, Inc. 1989.

Koza, John R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, Massachusetts: The MIT Press. 1992.