LAYOUT PROBLEM OPTIMIZATION USING GENETIC ALGORITHMS

Jiří Kubalík, Jiří Lažanský, Petr Zikl

Department of Cybernetics, CTU Prague,
Technická 2, 166 27 Prague 6, Czech Republic
{kubalik, lazan, zikl}@labe.felk.cvut.cz

This paper presents an application of genetic algorithm to a problem of finding of an optimal layout of objects on a material strip with the aim to reduce the total length of material used. In order to reduce the total search space the problem was solved as an ordering problem. This paper focuses on the representation issues of the problem and on designing of such a crossover operator that would allow building blocks formation and facilitate their mixing. Different implementations of the genetic algorithm were experimentally evaluated on a number of test cases. Some interesting results which illustrate a performance of the genetic algorithm are presented here.

1. INTRODUCTION

This paper presents an application of a genetic algorithm (GA) to a hard optimization problem of layout planning. GAs are probabilistic search and optimisation techniques, which operate on a population of chromosomes, each representing a potential solution of the given problem, with aim to breed some high quality solution [1]. They are operationally simple and represent a good choice for solving problems with large search space, where only little is known about its characteristics and even for solving black-box problems. As such they have been applied to many problems from the field of parameter optimizations, planning and scheduling, design, etc. [2].

The problem addressed in this paper is defined so that we want to find an optimal layout of a set of two-dimensional objects such that (i) the objects cannot mutually overlap and (ii) the length of the used piece of the material is minimal. The material strip has defined its length L and width W . For the sake of simple representation and manipulation of the objects we consider only rectangular shapes, which use the same grid structure.

The first and the most important step of a design of the genetic algorithm is to choose the representation of the problem to be solved. This is a crucial point since on the representation other features of the algorithm depend. These are the size of the search space, efficiency of the genetic operators, etc.

The most natural way of representing solutions of this problem would be a sequence of genes, each coding the actual position of individual object. On the other

hand there is a considerable drawback of such an approach which is the size of the search space. Note that we are seeking the optimal solution among all possible combinations of N objects' positions. Since each object can happen to be anywhere on the stripe, defined by dimensions L and W , the total number of possible layouts configurations of objects would be $L^N \cdot W^N$. Thus, if we used for example $L=200$ and $W=50$ the total size of the search space would be $200^{20} \cdot 50^{20}$, which is too much for any technique not using explicit knowledge on the problem.

That is why we focused on solving the problem as an ordering one; the implementation is described in next section.

2. IMPLEMENTATION ISSUES

The original layout planning problem as defined above can be seen as an ordering problem so that the algorithm generates sequences of objects and uses some localization algorithm according to which the objects are properly¹ placed on the grid in the given order. So the goal is to find an optimal order in which the objects will be laid on the stripe making the most compact layout. The size of the search space is of order $N!$ now. The same approach has been already proposed in [3]. The core of that paper was just the layout determining algorithm used to find a position of individual objects.

In this paper we are focusing on analyzing of the problem from the point of view of the genetic algorithm as the main concern is to facilitate the building block formation, propagation and mixing.

2.1 Localization Algorithm

The localization algorithm used in this work searches the partially covered grid for a free space, where the next object can be placed. Algorithm starts at the top left corner of the grid and moves the object down till it either finds the available space or it reaches the bottom of the grid. If it finds the place the object will be put there.

Figure 1 – Localization Algorithm

Otherwise the algorithm returns the object back on the top edge of the grid, shifts to the right and tries to find the free space within that new column. The search goes

¹ Properly means that each shape is placed on the grid so that it does not overlap with any other already located shapes and the length of the used material is minimal.

on until any legal position for the objects is found. In the same way all objects are processed in the order as they appear in the chromosome.

The function of the algorithm is illustrated for three different objects in Figure 1. The left picture depicts the initial situation on the grid. The right picture shows simultaneously the final states after inserting object A, B or C. On the example of object A we can see the situation when it is possible to place the object at position which is surrounded by other objects. Object B did not fit into that place even that its absolute dimensions does not exceed the size of the place, which is 3×6 units, due to its actual rotation. Another place had to be found for it. The last object C was too large for any blank space in the current layout so it was placed on the first available position after all the objects.

In order to reduce the computational time spent by the localization algorithm the size of steps by which the search moves down and right is optimized. While the object is moving down across the grid the column of cells, wide as the width of the object, is checked and a distance from the left most free cell is recorded. This value is then used to set the size of the step to the right if needed. In the same way vertical *stepping stones* are determined.

2.2 Fitness

In this case the candidate layouts are assessed according to the following fitness measure, which incorporates the requirements on maximal compactness of layouts and non-overlapping of objects

$$f = (L_{\max} - L) + 1/(1+S),$$

where $L_{\max}$ stands for the maximal length of the material stripe that can be used, L is the actual length of the layout and S is the amount of the residual scrap particles that lie idle among the objects. The measure is a compound of two components. The first part expresses the length of the layout in terms of the unused part of the stripe. This is defined so in order to have bigger fitness for shorter layouts. The second part represents a compactness of the layout so that it is larger for more compact layouts with less scrap. Its value ranges between 0 and 1 so it can only distinguish between the layouts of the same length. Note that the primary goal is to find the shortest possible layout with the least inside scrap.

2.3 Genetic Operators

A very important role in genetic algorithm applications plays the recombination operator. This is the only means to generate new candidate solutions. The basic idea behind the crossover is that given two promising solutions² their offspring could hopefully inherit the best parts of each. Optimally the crossover should be designed in order to allow for any useful inheritance. On the other hand, this is very difficult to achieve in practice. The reason is that we do not know much about the decomposition of the solutions into parts that might represent important features of solutions or the information to be inherited is known but hard to capture within the used representation. For instance, when solving a classical ordering traveling

² Parents are selected on the basis the better solutions have the bigger chance to be selected.

salesman problem it turned out that the most important information that should be inherited from the parents to the offspring is the individual connections between cities. New tours are then generated by maximally utilizing the links that appeared in the original tours; the operator is known as the edge-recombination [4].

Unfortunately this is not the approach we could adopt here. The adjacency of two objects in the chromosome does not have to mean the actual spatial adjacency of the objects on the grid. For example see the links between objects in pairs J-B and P-N in the first parent in Figure 2. Objects J and B are neighbors in the chromosome while they are far away from each other on the grid. On the other hand P and N lie close together on the grid even though they are not directly linked in the chromosome. This happens because the spatial position of each object does not depend on the links between objects in the chromosome. Instead it is determined by the partial layout to which the object is added with use of the localization algorithm. Apparently, focusing on utilization of links from parents is not worth in this case.

Figure 2 – Simple 1-Point Crossover

1-point crossover. Due to the reasons described above we decided to use a type of a simple 1-point crossover and its modifications. Simple 1-point crossover works so that first a crossing point is chosen randomly. The new chromosomes get the head part from the parental chromosomes and then are completed with the remaining objects in the order as they appear in the other parent. Obviously any chromosome generated this way must represent a feasible solution since all the objects will be present just once in it.

Let us take a look at what happens with layouts when they are crossed over, especially what of the parents passes on to the offspring. An example of the crossover action is shown in Figure 2. The two chromosomes are crossed over after the sixth position. As can be seen the effect of the operation is rather destructive. The children do not have much in common with their parents. This is not surprising in the case of the tail parts taken from the parents. It is natural that the sequence of objects from the tail part of one chromosome will be represented by completely

different layout when appended to the head sequence taken from the other parent. This is because the position of each object strongly depends on the actual state of the layout into which it is added. In other words, as the context of objects, determined by the antecedent sequence of objects, changes the spatial position and links of the objects change as well.

However, even the head sequence of objects does not represent fully compact part of the whole layout. This is due to the fact that objects, which are neighbors in a chromosome, are not adjacent to each other on the grid and vice versa. There are many gaps that are filled by objects that appear later in a chromosome. Simply because they are placed on the grid on position found by the localization algorithm. That is why we can observe for instance object L surrounded by objects A, B and C in first parent although its position in the chromosome is quite far away from the triple A, B, C. When simply crossing the chromosomes the head parts does not map onto compact block of objects as the parental chromosomes did. Instead the head sequence generates just a partially compact skeleton structure with possibly many gaps. Since the remaining objects are then added in a different order, determined by the other parent, the gaps might not be filled as optimally as in the original layout.

Reordering algorithm. The situation described above can be improved to some extent when using an adjusted ordering of objects, which takes into account the right order of objects as they appear on the grid. The original chromosome is reordered according to the following algorithm:

1. create an ordered list L of objects, the objects are ordered according to their actual position on the grid – the more left and up the better - in a descending manner.
2. use an auxiliary empty grid, $j = 0$
3. $i = 1, j = j + 1$
4. take i -th object of list L
5. find a position for the given object
6. if position is different from its actual position in the original encoding then $i = i + 1$ and goto step 4
7. insert the object into the new chromosome at j -th position
8. if $j < \text{number_of_objects}$ then goto step 2 else stop

The performance of the algorithm is illustrated in Figure 3. The main goal is to order the objects in such a way that the compact parts of a layout will correspond to a compact group of objects within the chromosome. Note that the position of any object cannot be changed by just reordering the chromosome since it could cause changes in the compactness of the overall layout. This is guaranteed by checking each object's position made in step 6. If the positions were not checked the following situation might happen. Let us assume example from Figure 3 where we have already placed first eight objects into the new chromosome. The next candidate is the object I since it is the left-most unused object of the original layout. The localization algorithm will find the available position for it, which is by one row higher on the grid than it was in the original layout. This would cause that object D would be placed by three columns to the right of its original position and position of other objects will be influenced as well. When the step 6 is used, first object D and

then object I will be inserted into the new chromosome and the layout does not change.

Figure 3 – Original vs. Adjusted Ordering of Objects in Chromosome

Let us now explore the ability of the genetic algorithm with the simple 1-point crossover to form and propagate useful building blocks to subsequent populations. As described above the crossover operator uses the head part of one parental chromosome to begin a new layout, which is then completed with objects taken in order defined by the other parent. So the first part of the offspring represents something that is more or less³ inherited from one of the parents. The second part of offspring is created in rather random manner, where only little can be inherited from parents. Thus the most influencing part of chromosome is its very beginning. This is also the best candidate for possible forming of building blocks that can be preferred and propagated during the evolution. This means that once some good-looking beginning sequence shows up in the population it is hard to replace it with anything else. In some sense the process of evolution of the best solution can be seen as that it starts with competition among the candidate solutions to find the best beginning of the layout, and then the search gradually proceeds towards the end of the layout.

In order to further boost the exploration abilities of the algorithm following modification of the crossover operators was proposed. The operators are based on the idea of so called *hot-spots* as crossing points that are likely to generate some improvement. Here the hot-spots represent objects that are bound to the top edge of the material sheet; for instance objects L, C, D, G, K, and P in Figure 3. The basic idea behind the operators is that there might be some group of objects somewhere in the middle of chromosome that could better fit at the beginning of the layout.

1-point crossover with hot-spots (1-point HS). This operator does not strictly inherit the beginning of parental chromosome (layouts). It differs from the simple 1-point crossover in such a way that the randomly chosen crossing point (any top-most object of the layout) determines a starting object of the generated layout. From this object till the end of chromosome, i.e. the end of the parental layout, all objects are inserted to the offspring. Then the new chromosome is completed with remaining objects in the order as they appear in second parent.

2-point crossover with hot-spots (2-point HS). This operator is an extension of the previous one. It cuts out a “reasonable” part of one layout, uses it as a beginning of a new layout, and completes it with the objects from the other parent in a standard

³ Depending on whether the adjusted ordering of chromosomes is used or not.

way. Here the reasonable part means a group of objects, which are chained in a *column* from the top to the bottom edge of the area. The algorithm of cutting out the parts from inside of parental layout starts with an arbitrary object that subtends right below the upper edge of the area. Let us denote the object as O . Object O is inserted at the first position of the new chromosome and the algorithm proceeds as follows:

1. $i = 2$ // position of object in new chromosome
2. find unused object such that it lies under O on the grid, and is close to the bottom edge of O
3. if such an object exists
 - then insert the object into new chromosome at position i
denote the object as O
 $i = i + 1$
goto step 2
 - else choose an arbitrary end-object E out of the objects lying behind O on the grid and add them in that order into the new chromosome

The operator described above works so that from one parent it extracts group of objects that represents a compact block on the grid. Such a block should start with a complete column of objects if possible, see step 3 of the algorithm. For example in Figure 3b we can find the following columns {1, 2, 3, 4}, {6, 2, 5}, {7, 8, 5}, {9, 10}, {11, 12, 13, 14}, {15, 16, 17}, and {18, 19}. Thus any of the columns can become a beginning of new layout. This makes the algorithm more explorative and may increase the possibility of finding better solutions.

2.4 Configuration of Genetic Algorithm

The genetic algorithm used in our experiments belongs to the class of generational genetic algorithms with overlapping populations of size $PopSize=50$. This means that every generation $P_{cross} \times PopSize$ new individuals are created by crossover and the remaining individuals are added by replication operator. Probability of crossover was chosen $P_{cross} = 0.8$. A mutation changing position of randomly chosen object was used. Selection method was the simple roulette wheel with linear scaling.

3. EXPERIMENTS

This section presents experiments carried out (1) to evaluate capabilities of the genetic algorithm to solve the layout problem and (2) to investigate an effect of proposed reordering algorithm and hot-spots. All results are averaged over 20 runs.

Figure 4 – The best layout of the initial population and the final optimal layout

Two problems with 40 and 42 objects were used in the experiments. Figure 4 shows a successful resolving of the later problem that was achieved with 1-point HS crossover. The same good solution was found also with 2-point HS crossover.

Convergence curves are shown in Figure 5. Figure 5 a) compares simple 1-point crossover (dashed line) with 2-point HS crossover (solid line) on the first problem.

Figure 5 b) compares simple 1-point crossover (dashed line) with 1-point HS crossover (solid line) on the second problem. Both figures as well as the results in Table 1 show the improvement achieved when crossovers use the hot-spots. On the other hand the effect of the reordering algorithm was not as big as we had expected. A significant improvement was observed only when the first problem was solved using the 1-point HS crossover.

Figure 5 – Convergence Characteristics with the Three Crossover Operators

Table 1 – Results Obtained with the Three Tested Crossover Operators

Problem (objects)	simple 1-point		1-point hot-spots		2-point hot-spots	
	sort on	sort off	sort on	sort off	sort on	sort off
1 (40)	366.5	366.2	369.8	367.5	368.3	368.8
2 (42)	326.0	327.0	329.5	329.0	328.8	328.2

4. CONCLUSIONS

This paper presents an application of genetic algorithm to the problem of finding an optimal layout. The problem was solved as an ordering problem with the main focus on the representation issues and recombination operators. First experiments have shown the applicability of the presented ideas. However more comprehensive analysis should be carried out yet.

5. ACKNOWLEDGMENTS

The research was supported by the State Department of Education within the frame of the project MSM 212300014.

6. REFERENCES

1. Goldberg D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989
2. Dasgupta, D., Michalewicz, Z.: *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, Germany, ISBN 3-540-62021-4, 1997
3. Ono, T., Watanabe, G.: Genetic Algorithms for Optimal Cutting, in [2], pp. 515-530
4. Whitley D., Starkweather T., Fugay D.: Scheduling Problems and Traveling Salesman. The Genetic Edge Recombination Operator. In Proceedings of the Third ICGA, San Mateo, CA, Morgan Kaufmann Publishers, pp. 133-140, 1989