USING EVOLUTIONARY COMPUTATION FOR FINDING COMPACT SHAPE ARRANGEMENTS

G.R. Raidl*

*Institute of Computer Graphics, Algorithm and Programming Methodology Group,
Vienna University of Technology, Karlsplatz 13/1861, 1040 Vienna, Austria

ABSTRACT

This paper deals with the problem of finding very compact arrangements of a given set of arbitrarily shaped two-dimensional objects. Existing techniques utilizing *genetic algorithms* (GAs) are discussed. A new approach based on *evolution strategies* (ESs) and improved by a local optimization technique called *skillful genotype decoding* (SGD), which avoids infeasible and meaningless solutions, is introduced. In general, the evolution strategy converges to better solutions than comparable GA based techniques known to the author.

INTRODUCTION

In material processing, it is a very hard optimization task to find the most compact arrangements for arbitrarily shaped two-dimensional objects. One application is to cut various parts from a given raw material. The arrangement searched for should be as tight as possible so that material waste is minimized. Such problems occur for example in steel or tailor mills. Depending on a concrete application, objects may be placed and rotated arbitrarily or various restrictions may be given (e.g. rotation only in 90 degree steps, only for some objects, or generally not allowed). Another area where this class of optimization problems occurs is the design of electronic circuits: Differently shaped electronic parts need to be placed on an integrated circuit or printed board with the goal to get the chip or board area as small as possible. Besides minimizing the chip or board size, a very important goal in this application is to make connections between components as short as possible.

In literature, this class of optimization problems is often referred to as *cutting, packing, facility layout* or *facility arrangement problem.* Many different approaches to get good solutions can

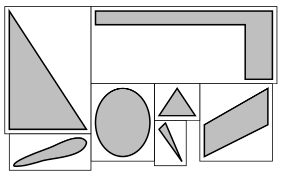

Figure 1: *A solution to the cutting problem created by a traditional approach which arranges rectangular bounding boxes of arbitrarily shaped objects*

```
Evolutionary Algorithm:
$P \leftarrow$ set of random solutions;
determine fitness values for all solutions ($P$);
while not(termination condition) do
   $Q_S \leftarrow$ select parent solutions ($P$);
   $Q_R \leftarrow$ recombine parents ($Q_S$);
   $P \leftarrow$ mutate offsprings ($Q_R$);
   determine fitness values for all solutions ($P$);
end
```

Figure 2: *Basic evolutionary algorithm*

be found in literature, but unfortunately most of them only deal with rectangular objects, see e.g. [3, 16], *floorplanning*: [5, 10, 11, 17]. The traditional way to apply these algorithms to arbitrarily shaped objects is to arrange bounding boxes of all objects. But depending on the objects' shapes, the use of bounding boxes may introduce significant material waste, and a pretty good solution for the bounding boxes is often far from optimal for the actual shapes, see Fig. 1.

EVOLUTIONARY ALGORITHMS

Nearly three decades of research and applications have clearly demonstrated that the mimicked search process of natural evolution can yield very robust, direct computer algorithms for difficult optimization tasks, although these imitations are crude simplifications of biological reality. These *evolutionary algorithms* (EAs) are based on the collective learning process within a population of individuals, each of which represents a search point in the space of potential solutions to a given problem, see Fig. 2. The population is initialized arbitrarily, and it evolves towards better and better regions of the search space by means of randomized processes of *selection*, *mutation*, and *recombination*. A *fitness function* evaluates search points, and the selection process favours those individuals of higher fitness to reproduce more often than worse individuals. The recombination mechanism allows the mixing of parental information while passing it to their descendants, and mutation introduces innovation into the population. For a general introduction into the practice and theory of EAs see [2, 7, 12].

There are some variants of EAs which mainly differ in the preferred methods of encoding solutions and doing selection, recombination and mutation. One class of EAs is called *genetic algorithms*, see [6, 8, 9, 12]. According to [8], solutions are typically encoded as binary strings, called *gene strings* (or *chromosomes*). The selection is usually a random process where a solution is selected as parent with a probability proportional to its fitness value. The recombination exchanges randomly selected parts of two parents' gene strings, and mutation flips a random element of this string.

GAs were already successfully applied to cutting problems in [4, 5, 10, 13, 14, 15, 16, 17]. While [5, 10, 16, 17] concentrate only on rectangular objects introducing the problems discussed above when applied to bounding boxes of arbitrary shapes, [4, 13, 14, 15] actually deal with arbitrarily shaped objects. [4] uses a two-dimensional binary chromosome in which the positions and possible orientation values are encoded. Special recombination and mutation operators are defined to improve the performance. [13] introduces another GA which uses binary encoding of object positions (no rotations are possible) and an improved varying fitness function. Unfortunately, both methods have problems with the large portion of infeasible and meaningless solutions they generate during the search: Many solutions contain overlapping objects or objects that are not located side by side but are dispersed over the whole area with only few connections. To guide the search to feasible solutions, the two approaches mentioned add penalties according to overlapping areas to fitness values of infeasible solutions. But note that this technique does not guarantee feasibility of a final solution.

In most cases it is definitely more efficient to look for an encoding for the problem's solutions and/or appropriate operators for the GA which only produce feasible and meaningful solutions. In our previous work [14, 15], a GA similar to [13] was enhanced by a mechanism called *skillful genotype decoding* (SGD). This new mechanism improves the performance of the GA essentially by avoiding overlapping objects and therefore infeasible solutions. Furthermore, each arrangement is locally optimized during SGD by moving shapes together. SGD is performed once for every new solution before fitness calculation.

Evolution strategies (ESs) are another class of EAs, see [1, 2, 9, 7]. This kind of optimization method is especially useful for the optimization of difficult multimodal numerical functions. The main difference to GAs is the fact that numerical parameters of a solution are not encoded in a binary gene string but in a vector of floating point or integer values. Furthermore, the selection is usually defined as simply choosing the best offsprings of a population and is therefore deterministic. While recombination plays an essential role in GAs, it is not always needed and useful in ESs. Mutation is the primary operator of ESs for driving the search.

A new approach to solving cutting problems using a specialized evolution strategy which is again supported by SGD will be outlined in the following section.

AN EVOLUTION STRATEGY FOR THE CUTTING PROBLEM

To apply an ES to the cutting problem, solutions must be encoded by vectors of floating point or integer values. For each object i ($i = 1 \ldots n$) two coordinate values x_i and y_i denote the so-called *original position*. An additional orientation transformation value o_i is used for each object which may be rotated or mirrored. In our experiments, rotation in 90 degree steps and

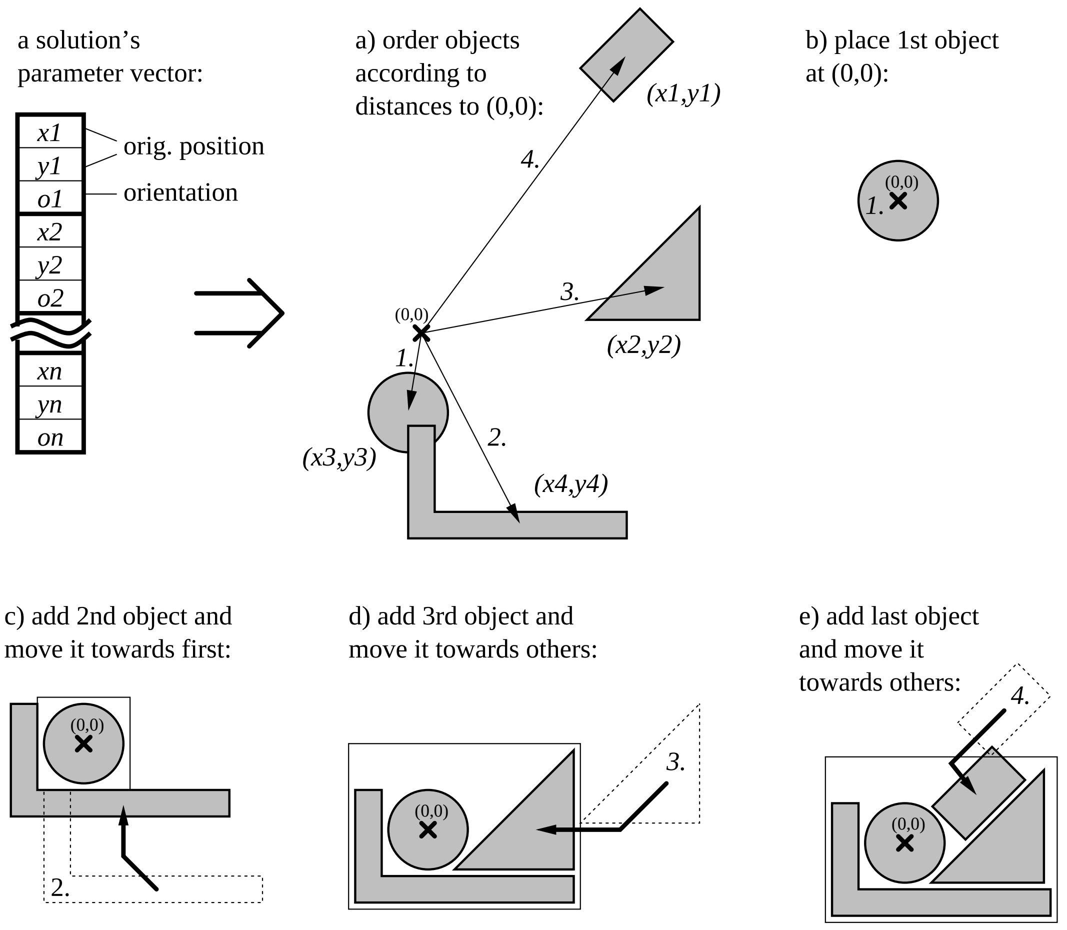
a solution's parameter vector:
x1
y1
o1
x2
y2
o2
xn
yn
on
orig. position
orientation
a) order objects according to distances to (0,0):
(x1,y1)
(x2,y2)
(x3,y3)
(x4,y4)
(0,0)
1.
2.
3.
4.
b) place 1st object at (0,0):
c) add 2nd object and move it towards first:
d) add 3rd object and move it towards others:
e) add last object and move it towards others:

Figure 3: *Encoding solutions and skillful genotype decoding (SGD)*

mirroring were allowed individually for each object. o_i therefore denotes one of eight possible Manhattan orientations.

Placing all objects at their original positions will in general result in infeasible solutions due to overlapping objects. In [14, 15], skillful genotype decoding has been proposed to modify object positions in such a way that solutions are locally optimized and overlap-freeness and therefore feasibility can be guaranteed. SGD can also be applied to the ES, see Fig. 3: All objects are first ordered according to their distances to the origin (0,0). The first object is then simply placed at the origin. Each other object is placed as follows: First it is added outside of the bounding box of all previously placed objects maintaining the original direction towards (0,0). Then the object starts moving ("*floating*") towards other objects in a very specific way, always avoiding collisions. Therefore, floating shifts objects close together leading to more meaningful solutions. See [14, 15] for a detailed description of the floating mechanism.

Fig. 4 shows the pseudo code of the ES adapted to the cutting problem. As in [14, 15], the

```
$(\mu, \lambda)$–Evolution Strategy:
$P(0) \leftarrow$ set of $\mu$ random solutions;
apply SGD and determine fitness values $(P(0))$;
$t \leftarrow 0$;
while not(termination condition) do
  $t \leftarrow t + 1$;
  $Q(t) \leftarrow \lambda$ copies of randomly chosen solutions from $P_{t-1}$;
  $S(t) \leftarrow$ mutate $(Q(t))$;
  apply SGD and determine fitness values $(S(t))$;
  $P(t) \leftarrow$ select $\mu$ best offsprings from $S(t)$;
end
```

Figure 4: *Evolution strategy for the cutting problem*

following fitness function is used to evaluate a solution s after SGD:

$$F(s) \leftarrow MW \cdot \sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2} \cdot A_i \qquad \text{(to be minimized)} \tag{1}$$

The objects' final distances to (0,0) are weighted with the object areas A_i, added up, and multiplied by the material waste MW (which is the area of the bounding box of all objects minus the sum of all object areas). This fitness measure has the advantage of being dependent on all objects and not just the outermost, as it would be the case if only MW were chosen. For more details, see [14, 15].

Note that no recombination operator is used in the proposed ES. Our tests have shown that the ES converges faster by only regarding mutation. The mutation operator, which modifies an existing solution randomly, is implemented as usual in ESs (see [1, 2, 9]): A normally distributed random offset is added to each component x_i and y_i of a solution's parameter vector. Smaller changes are therefore more likely than larger ones. Standard deviations of normal distributions are selected using an extended version of Rechenberg's 1/5–rule, see [2]. Orientation values o_i are set to new random values with a probability which also adapts according to an algorithm similar to Rechenberg's 1/5–rule.

RESULTS

Fig. 5 shows examples for final shape arrangements created by a (15,100)-ES with and without SGD. The CPU times of both runs were limited to 180min. All simulations were run on a 133MHz Pentium PC using Linux and GNU-C++. Note that infeasible solutions created by the ES without SGD were penalized by adding a term to the fitness function that depends on the total overlapping area.

Various test cases were used to examine the efficiency of the ES-approach compared to the GA in [14, 15]. Results show that in general the proposed ES converges slightly faster towards good solutions than the GA. Final solutions of the ES are usually more compact and therefore

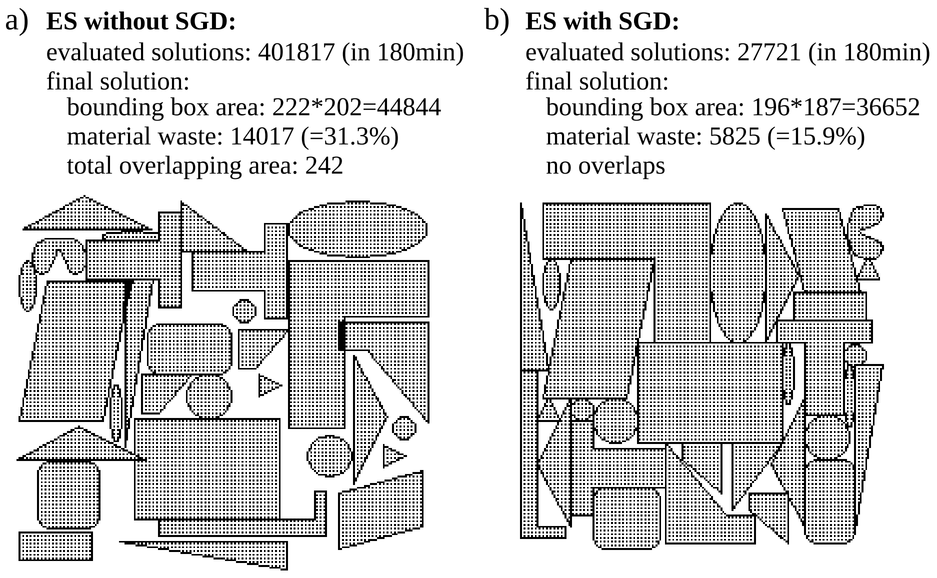
a) ES without SGD:
evaluated solutions: 401817 (in 180min)
final solution:
bounding box area: 222*202=44844
material waste: 14017 (=31.3%)
total overlapping area: 242
b) ES with SGD:
evaluated solutions: 27721 (in 180min)
final solution:
bounding box area: 196*187=36652
material waste: 5825 (=15.9%)
no overlaps

Figure 5: *Typical final solutions to a cutting problem with 30 objects determined by a (15,100)-ES without SGD (a) and with SGD (b) after 180min CPU time*

Used EA	Bounding box area	Material waste	Total overlapping area
GA without SGD	47982	17155 (= 35.75%)	270
ES without SGD	45032	14205 (= 31.54%)	234
GA with SGD	37217	6390 (= 17.17%)	–
ES with SGD	36799	5927 (= 16.22%)	–

Table 1: *Average values for final solutions calculated out of 15 runs per EA, 180min CPU time per run*

better than comparable GA-solutions. One reason for the observed better performance of the ES seems to be the direct use of floating point/integer values. Furthermore, it seems that the mutation operator of the ES is not as disruptive and therefore better suited than the GA's combination of crossover and mutation. See Table 1 for an average performance comparison between a (15,100)-ES and the GA with and without SGD. In these experiments, the same set of 30 shapes as in Fig. 5 was used.

CONCLUSIONS

Evolutionary algorithms have shown their suitability to find compact arrangements for sets of arbitrarily shaped objects. A GA has been significantly improved by introducing SGD as a local optimization technique to avoid infeasible and meaningless solutions. Latest experiments have shown that even better results are possible using an ES supported by SGD and only normally distributed random mutation instead of a GA with its binary parameter encoding and recombination as its primary operator. Again, SGD proved to be very substantial to always getting feasible final solutions of high quality.

REFERENCES

1. T. Bäck, F. Hoffmeister, H. P. Schwefel: *A Survey of Evolution Strategies*, in Proc. of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1991.
2. T. Bäck: *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.
3. A. R. Brown: *Optimum Packing and Depletion*, American Elsevier Inc., New York, 1971.
4. H. Chan, P. Mazumder, K. Shahookar: *Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome*, Integration: the VLSI journal 12, 1991, p.49.
5. J. P. Cohoon, S. U. Hegde, W. N. Martin, D. S. Richards: *Distributed Genetic Algorithms for the Floorplan Design Problem*, IEEE Transactions on Computer-Aided Design, 10(4), 1991, p.483.
6. L. Davis: *Handbook of Genetic Algorithms*, Int. Thomson Publishing Inc., 1991.
7. D. B. Fogel: *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ, 1995.
8. D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, 1989.
9. F. Hoffmeister, T. Bäck: *Genetic Algorithms and Evolution Strategies: Similarities and Differences*, Systems Analysis Research Group, University of Dortmund, Department of Computer Science, Technical Report No. SYS-1/92, 1992.
10. K. Kado, P. Ross, D. Corne: *A Study of Genetic Algorithm Hybrids for Facility Layout Problems*, in Proc. of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Pittsburgh, 1995, p.498.
11. T. Lengauer: *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley – Teubner, 1990.
12. Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 1992.
13. V. Petridis, S. Kazarlis: *Varying Quality Function in Genetic Algorithms and the Cutting Problem*, in Proc. of the 1st IEEE Conference on Evolutionary Computation, Orlando, FL, 1994, p.166.
14. G. R. Raidl: *Skillful Genotype Decoding in EAs for Solving the Cutting Problem*, in Proc. of the 5th International Conference on Evolutionary Programming, edited by D. Fogel, MIT Press, San Diego, CA, 1996, in print.
15. G. R. Raidl: *Solving the General Cutting Problem with an Improved Genetic Algorithm*, in Proc. of the 11th International Conference on Systems Engineering, edited by A. Iyer, Las Vegas, NV, 1996, p.680.
16. P. Y. Wang, *Two Algorithms for Constrained Two-Dimensional Cutting Stock Problems*, Operations research 31(3), 1983, p.573.
17. D. F. Wong, C. L. Liu: *A new algorithm for floorplan design*, in Proc. of the ACM-IEEE Design Automation Conference, Las Vegas, NV, 1986, p.101.