A Building Block Favoring Reordering Method for Gene Positions in Genetic Algorithms

Onur Tolga Sehitoglu
Dept. of Computer Engineering
Middle East Technical University, Ankara
onur@ceng.metu.edu.tr

Göktürk Üçoluk
Dept. of Computer Engineering
Middle East Technical University, Ankara
ucoluk@ceng.metu.edu.tr

Abstract

This work proposes an algorithm to speed-up convergence of genetic algorithms that is based on the investigation of neighbouring gene values of the successful individuals of the chromosome pool. By performing some statistical inference on neighbour gene values, coherent behaving genes are detected. It is claimed that those coherent acting genes are belonging to the same building block that leads to the solution. It is desirable to keep genes of the same building block together to let them —probabilistically— live together in next generations. Therefore, a reordering of the gene positions is performed, periodically. The proposed algorithm is implemented for a minimum area non-overlapping rectangle placement problem and results are compared to a classical GA implementation.

1 Introduction

n-point crossover is known to favor the formation of the so called *building blocks*. Building blocks are those relatively short gene subsequences which, when combined, form better solutions. The mechanism of genetic algorithm (GA) implicitly assigns higher chances to building blocks to exist over generations. In other words, to survive.

The idea in this work is based on the fact that building blocks mainly consist of those group of genes which are closely located to each other. Now, consider two genes, which due to the nature of the problem would tend to form a building block, and are separated by some other genes in the chromosome in the default gene ordering of the chromosome representation. These two will not easily be able to form a building block, when subject to n-point crossover; because the probability of hitting some intermediate point as the cut-point of the crossover is high. Therefore the order of genes in the chromosome encoding plays an important role in the convergence of the GA.

We propose a method in which permutations of the gene ordering are considered dynamically. In Section 2 the proposed method is described. The method is implemented for a minimum non-overlapping rectangle placement problem and the implementation is described in Section 3. In Section 4 results are compared with the classical GA's results.

2 Proposed Method

In each generation a single global permutation is considered. This permutation maps a gene number to a position value in the chromosome encoding. The crossover operation is based on this mapping instead of the original gene order in the representation. During the evolution of the genetic algorithm, this global permutation is readjusted by means of statistical analysis on neighbouring genes, calculated for the bests of the pool.

- Consider that the solution to a subject problem requires the determination of a set of parameters $\mathcal{X}$. As the first step GA requires the determination of a one-to-one mapping from the set of parameters to the set of binary strings $\mathbf{\Gamma}$, this is called the *encoding* of the parameters.

  $$\mathcal{E}_i : \mathcal{X}_i \mapsto \Gamma_i \quad \text{where} \quad \mathcal{X}_i \in \mathcal{X}, \;\; \Gamma_i \in \mathbf{\Gamma}$$

  We name Γ_i as *genes*. Γ_i's, each of which are binary strings, are concatenated into a one dimensional binary array $\mathbf{\Gamma}$ so that Γ_i is followed by Γ_{i+1}. An instance of $\mathbf{\Gamma}$ is called a *chromosome*.

- We define a *permutation of genes* as an ordering relation of the genes in a chromosome. If $\mathcal{P}_p$ represents such a permutation operator on a chromosome, it is defined by means of its gene elements as:

$$[\mathcal{P}_p \Gamma]_i \triangleq \Gamma_{p(i)}$$

Here p is a permutation function:

$$p : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, n\} \quad \text{where}$$

$$n = |\mathbf{\Gamma}| \quad \text{and} \quad p^{-1} \quad \text{exists.}$$

- As a part of the proposed method we define for each gene position i of the chromosome a function f_i that admits a gene value as arguments and is defined as:

  $$F_i : \Gamma_i \mapsto \mathcal{R} \quad \text{where} \quad \Gamma_i \in \mathbf{\Gamma}$$

  Please also note that, in the most general case, Γ_i can be a tuple representation of various features. Therefore F_i may be defined over the corresponding allele tuple domain.

  For denotational simplicity, we will define

  $$f_i \triangleq F_i(\Gamma_i)$$

  so f_i is a real value which is calculated from a Γ_i by means of F_i.

  When the *building blocks* of a GA is reverse mapped to the actual problem domain, it is usually observed that the formation of blocks corresponds to some patternal, structural, mathematical invariance or covariances.

  The functions F_i will serve to express features as real values which will be used to discover some invariance or covariances. So, in a way, we are defining a handle, where the GA user has a chance to hook-in his hint for defining the features which may lead to building blocks.

- At each generation we calculate a *neighbour-affinity-function* $\mathcal{A}_i$ that is defined for each neighbour gene pair position in the current permutation over the whole pool. We propose it to be of the most general form:

  $$\mathcal{A}_i^p : \{\langle f_{p^{-1}(i)}, f_{p^{-1}(i+1)} \rangle\}_{pool} \mapsto [0, 1] \quad \text{where}$$

  $$i = 1, 2, \ldots, n-1$$

  Of course if the chromosome composition is a vector of genes (due to the nature of the problem) then all f_i's will reduce to a single f and hence $\mathcal{A}_i$ will reduce to a single function $\mathcal{A}$.

  $\mathcal{A}_i$ is a problem specific function and should be coded according to the characteristics of the problem encoding. Since the aim is to construct good building blocks, the result of $\mathcal{A}_i^p$ should be closer to 1 for gene values acting coherently and contributing for better solutions. Correlation and standard deviation analysis can be used as alternatives for $\mathcal{A}_i$.

- The next step is to modify the permutation mapping by looking at the results of neighbour-affinity value calculations which are calculated from the instances of the current generation of the pool. We will be calling these values A_i^p.

  To do this, every gene position in the permutation will be considered, and based on the right and left neighbour-affinity values of each gene a decision will be made for ots position. If this value is found to be less then a threshold value τ, these two genes will be considered as unrelated to each other and the permutation will be changed to separate them. Actually there exists 4 possible affinity cases for a gene:

  1. *Affinity value is greater then τ for both neighbours* so there is no problem with the current ordering. The gene position is kept in the permutation unaltered.
  2. *Affinity value with left gene is greater than τ but it is less than τ for the right gene.* This means left neighbour is okay but we should separate it from the right. So, gene exchanges position with the left neighbour. In this way the good affinity with the left neighbour is preserved.
  3. *Affinity value with left gene is less than τ but it is greater than τ for the right gene.* Similarly gene exchanges its position with the right neighbour.
  4. *Both affinity values are less than τ.* Gene is moved to a random position in the permutation.

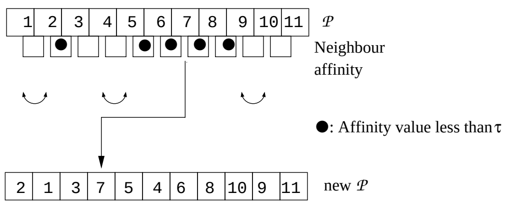

Figure 1: Modification of order according to affinity values

Modifications in the permutation mappings are kept to be as local as possible. The algorithm to do this is as follows:

```
for $i \leftarrow 2 \ldots n$ do
  {
    if $p[i-1]$ $is\ not\ modified\ in\ prev.\ iter.$
      if $A_{i-1} > \tau \wedge A_i < \tau$
        $p[i-1] \leftrightarrow p[i]$
      else if $A_{i-1} < \tau \wedge A_i > \tau$
        $p[i] \leftrightarrow p[i+1]$
      else if $A_{i-1} < \tau \wedge A_i < \tau$
        $Move\ p[i]\ to\ a\ random\ location$
  }
```

2.1 GA Engine

- Initialize the global permutation $\mathcal{P}$ to $[1, 2, \ldots, n]$
- Generate a random population, evaluate it and store it also as the former generation.
  Pool size = 100 chromosomes

Repeat :

- Mutate.
  Mutation Rate = Once each 10 generation one random chromosome
  Changes per Chromosome = Flip one randomly selected gene
- Mate all the pool by forming random pairs.
  Determine the crossover points.
  Perform crossovers among the chromosomes according to the permutation. Crossover point is taken in permutation mapping.
  Cross Over = At 10 random chosen random length gene intervals
- Evaluate the new generation.
  Keep Ratio = At most 10%
- If reorder period is reached:
  - Calculate A_i^p values for the selected-kept chromosomes according to current permutation mapping.
  - Modify the ordering of each gene position according to A_i values and find the new permutation.
- Display/Record performance result.
- If it was not the last generation the user demanded, **goto Repeat**.

3 Implementation

For testing and implementation of the proposed system, a minimum area rectangle placement problem is chosen. In this problem a set of rectangles is given as input. The aim is to find a placement of all rectangles such that all rectangles are in a bounding box which is minimized in the area and no two rectangles intersect. The input is the width and height information of n rectangles.

3.1 Problem encoding

- $I_i = \langle w_i, h_i \rangle$ is the width & height input of the i^{th} rectangle.
- $\Gamma_i = \langle x_i, y_i, o_i \rangle$ where x_i and y_i are offsets from the origin and $o_i \in \{0, 1\}$ is the orientation (not rotated, rotated 90°) in the placement. Each gene Γ_i defines a placement for I_i. Γ_i in combination with I_i describes a rectangle positioning with absolute coordinates.
- The fitness function for a chromosome is defined as:
  $V = cZ + dO, +eB \quad j = 1, \ldots, poolsize$
  where c, d, e are positive constant weights and Z is the total area of the rectangles placed out of the placement area, O is the total overlapping area among all rectangle pairs, B is the area of the minimum bounding box covering all rectangles in the placement. The aim of the genetic algorithm is to find a chromosome that *minimizes* V.
- One point crossover for crossover point k is defined by means of the permutation mapping $\mathcal{P}$ as:

$$\Gamma_i^{AB} = \begin{cases} \Gamma_i^A & \text{if } p(i) < k \\ \Gamma_i^B & \text{if } p(i) \geq k \end{cases}$$

$$\Gamma_i^{BA} = \begin{cases} \Gamma_i^B & \text{if } p(i) < k \\ \Gamma_i^A & \text{if } p(i) \geq k \end{cases}$$

  where Γ^A and Γ^B are crossedover to produce two offsprings: Γ^{AB} and Γ^{BA}.
- Since the gene values are three-tuples of numeric values, there is no need to define an auxilary function f_i which will map them to $[0, 1]$. Their numerical values are directly used in the formula.
- The neighbour affinity function is defined to be the total standart deviation of offset values describing the placement:
  $A_i^p = 1 - (t\sqrt{\sigma_x(i)} + u\sqrt{\sigma_y(i)} + v\sqrt{\sigma_o(i)})$
  where $\sigma_\alpha(i)$ is defined as the variance of the differences of the α features of the $\langle \Gamma_{p^{-1}(i)}, \Gamma_{p^{-1}(i+1)} \rangle$ tuples in the population. t, u, v are positive constant weights:

$$\sigma_\alpha(i) = \frac{\sum_{j=1}^{M} \delta_{i,j,\alpha}^2 - (\sum_{j=1}^{M} \delta_{i,j,\alpha})^2/M}{M}$$

$$\delta_{i,j,\alpha} = \Gamma_{p^{-1}(i),j,\alpha} - \Gamma_{p^{-1}(i+1),j,\alpha}$$

Due to the proposed algorithm, when A_i^p is small then two neighbour genes should be placed in relatively arbitrary positions in the population. This is so, a small A_i^p value means that they are statistically found not to cooperate well towards the solution. When A_i^p values become close to 1, the relative placement of neighbour genes is almost fixed in the population which means a building block is established by these neighbours.

- Reorder frequency is chosen as 5. That means, analysis of A_i values and permutation modification is done once in 5 generations.

3.2 Test results

The resulsts of the implementation of the proposed algorithm to the resulst of the classical GA implementation where all other paramaters like crossover operations, mutation frequency, keep ratio[1] are the same but no permutation shuffling is done. In the implementation of the proposed method, crossovers are carried out by using the permutation information, and permutation mapping is modified once in 5 generation. Tests are repeated 20 times with different random seeds. In Figure 2 the evolution of the best individual fitness value is indicated. The proposed gene reordering method converges significantly faster. It reaches the mimimum in 150 generations compared to 550 to 600 generations of the classical GA version.

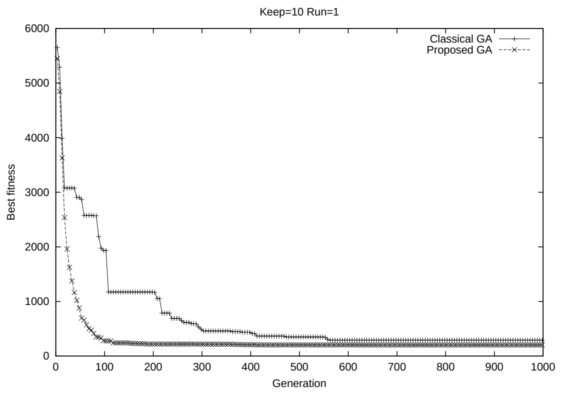

Figure 2: Evolution of the best individual for a sample execution

[1]Ratio of the best members of current generation to be kept in the next generation

Distributions of the best individuals for all 20 execution cases through the generations is given in Figure 3. The gene reordering version is consistently converging to a solution in less number of generations for all cases. furthermore, the classical version exhibits a slow convergence behaviour which is also likely to get trapped into a local minima more frequently.

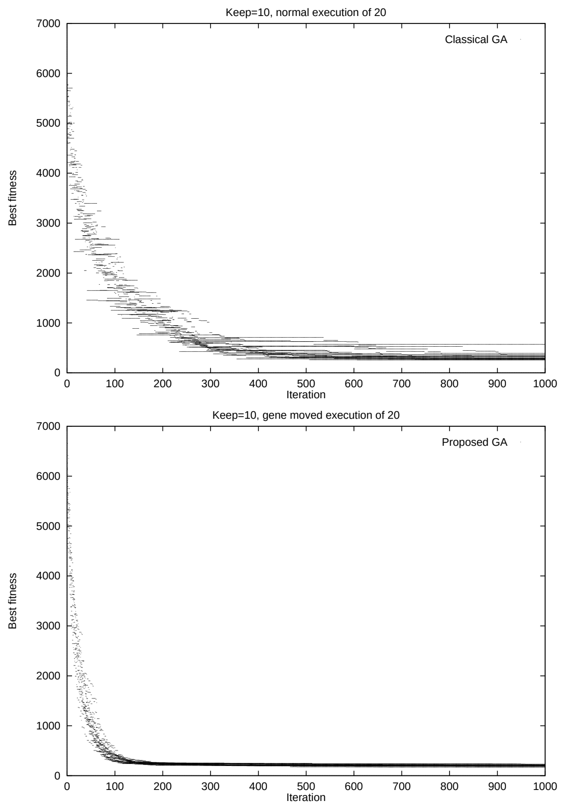

Figure 3: Distribution of the best fitness values for 20 executions where $Keep$ ratio is 10

When the fitness values of the best individuals after 1000 iterations are compared it is observed that though the proposed algorithm converges significantly faster it does not find a worse solution than the original algorithm (Figure 4). In contrary, since it gets caught in local minima, the classical algorithm requires more than 1000 generations to achive the quality of the solution which the proposed GA reaches in 150 generations.

Considering the reordering cost, it is also observed that after a small number of generations neighbour affinity values gets under the specified threshold and the system does not require a permutation change. In most of the cases a fixed permutation was converged to in 50 to 100 generations.

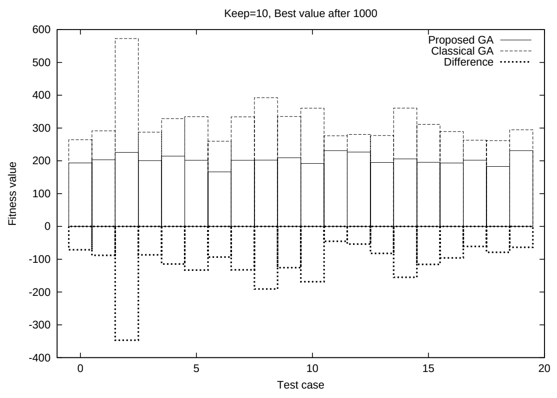

Figure 4: Best individuals fitness value after 1000 generation. (Small numerical value on fitness axis means a better solution.

4 Conclusion

The proposed method achieved a high improvement in the convergence speed without causing any genetic drift problem leading to a local minima. On the contrary the quality of the solution is usually better after a fixed number of iterations. Since the method converges faster, it is also much more suitable for time critical problems where a suboptimal solution is useful.

References

[1] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1996.

[2] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[3] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[4] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[5] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.

[6] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT press, 1996.