Adaptive mutation operator cycling

Aleksandar Prokopec, Marin Golub
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
{aleksandar.prokopec, marin.golub}@fer.hr

Abstract

Parameter tuning can be a lengthy and exhaustive process. Furthermore, optimal parameter sets are usually not only problem specific, but also problem instance specific. Adaptive genetic algorithms perform parameter control during the run, thus increasing algorithm performance. These mechanisms may also enable the algorithm to escape local optima more efficiently.

In this paper, we describe the fitness landscape for permutation based problems, and define local and global optima, as well as the notion of adjacency of solutions. Using these definitions we show why it makes sense to combine multiple genetic operators adaptively, give examples of this, and show that an algorithm combining multiple mutation operators has a greater chance of escaping local optima. We then describe the adaptive tournament genetic algorithm (ATGA) which uses multiple mutation operators, describing a variety of used adaptation mechanisms and conclude the paper by showing experimental results.

1. Introduction

Genetic algorithms are applicable to a wide range of problems. In designing a genetic algorithm, one must choose the representation for the solution and reproduction operators, but also tune the parameters to increase the performance of the algorithm. However, optimal parameter sets may take a significant time to be found, and they are also problem-instance specific. One parameter set may work very well for one problem instance, but it may prove inadequate for another. Finding optimal parameter sets for each problem-instance is hardly a task worth pursuing. A requirement is, thus, established that an algorithm should change its parameters based on the problem instance it solves and also, to increase performance further, on the current state of the search. To do this a feedback between the algorithm and the search must exist. Such algorithms are said to be adaptive.

The act of changing parameters during the run of the algorithm is called parameter control. Eiben, Hinterding and Michalewicz classify parameter control into three categories: deterministic, adaptive and self-adaptive [1]. In this paper, we restrict ourselves to adaptive parameter control. In adaptive parameter control, it has to be seen which parameters of the algorithm actually change and what is the evidence upon which the change is made. Our adaptive mechanisms will focus on changing mutation probability and mutation operators. The basis for the change will be described later for each adaptation mechanism.

We proposed mutation probability varying scheme, adaptive operator cycling, and mutation operator statistics mechanism, somewhat similar to [2], but working on the individual level instead of allele level.

Our adaptive mechanisms were applied to permutation problems, so we start by introducing certain terms related to permutations.

2. Fitness landscape for permutation-based problems

In problems in which representation of solutions is a real-valued (or an integer-valued) vector, the notion of fitness landscape is straightforward and obvious. Fitness landscape is a scalar field and may be envisioned easily. In case of the two-dimensional vectors, the fitness landscape is a surface with a global optimum and possibly local optima.

In the case of the permutation-based problems, the notion of a fitness landscape is no longer so intuitive. The solutions do not form a scalar field, as the elements of the permutation are integers instead of real numbers, and not all integer combinations are allowed (we exclude the random key encoding for permutations from [5], that encodes permutations with a real vector, from this analysis). We introduce the following definition of permutation adjacency.

Definition 1. *A permutation Q is adjacent to permutation P if the application of the mutation operator to the permutation P can generate permutation Q.*

Note that adjacency is conditioned by the mutation operator. Different mutation operators provide different adjacency relations. Note, also, that this definition of adjacency is not inherently symmetric – if the permutation Q is adjacent to permutation P, then permutation P may not be adjacent to permutation Q. However, for most mutation operators, adjacency is symmetric. For instance, inversion yields a symmetric adjacency relation between permutations – inverting a subsegment of a permutation may be discarded by repeating this operation on the same subsegment. However, a mutation operator that performs a local search in concordance with some fitness function will not yield a symmetric adjacency relation.

Definition 2. *The fitness landscape for permutation-based problems is a graph whose vertex set equals the set of all possible permutations, and edge set contains all vertex pairs whose respective permutations are adjacent. Each vertex is additionally assigned its fitness value.*

Possible permutations are those permutations that represent a solution in the search space – generally, not all permutations may be feasible. Further, it follows that the fitness landscape defined in this manner also depends on the mutation operator, and the fitness landscape may be a directed graph in some cases. Figure 1 shows this for permutations of the order 3, and for scramble and swap mutations [6] – it can be observed that even for the case of low order permutations, some of them are not necessarily adjacent, thus forming a different fitness landscape. Each permutation in the solution space has a certain fitness, so each vertex is assigned its fitness value. In visualization of the fitness landscape, each vertex may be assigned a height proportionate to its fitness value. This “landscape” may then resemble the one that follows from the real-valued representation. The graph, however, may not be and in most cases is not planar, thus a vizualisation may not be particularly helpful. However, it should be noted that visualization of higher dimension real vector functions is also vague. The notion of the fitness landscape shall become important after we introduce the following modified definition, borrowed from [5].

Definition 3. *A global optimum is a vertex in the fitness landscape whose fitness value exceeds the values of all other vertices (or is exceeded by all other values if we are minimizing). A local optimum is a vertex in fitness space that has the property that no chain of mutations starting at that vertex can lead to a vertex whose value is greater without first leading to a vertex whose value is lower than the starting vertex (or vice versa, if we are minimizing).*

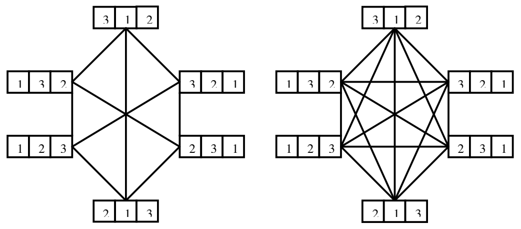

Figure 1. Adjacency of permutations of the order 3 for the swap and scramble operators

In other words, a local optimum is a vertex in the fitness landscape with the property that all of the adjacent vertices have a lower fitness value. As was deducted earlier, it follows that the distribution of local optima depends on the mutation operator. What is a local optimum in a specific problem instance for one mutation operator, may not be a local optimum for another. This may lead to the premature assumption that a mutation operator having the least number of local optima (or none) may be the best mutation operator, but this is not so. The scramble mutation is an example of an operator that yields a totally connected fitness landscape, thus having no local optima, because every solution is adjacent to the global solution. However, we’ve found that scramble mutation used for TSP is outperformed by other standard mutation operators such as swap, inversion and insert mutation, described in [6], and this is confirmed by Syswerda, as mentioned in [4]. It seems that another important trait of a mutation operator is that the changes it introduces in terms of fitness value are relatively small.

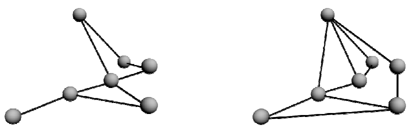

Figure 2. Illustration of two different fitness landscapes

An important conclusion is that a permutation that is a local optimum for one mutation operator, may not be a local optimum for another. Figure 2 illustrates this – it can be observed that the vertex in the upper-right part of the graph is a local optimum in the left graph, but isn’t in the right graph, as it is connected to a vertex with a greater height (fitness value).

Furthermore, we may introduce the notion of distance between solutions as the minimum number of vertices in the fitness landscape that must be traversed to reach a vertex Q starting from vertex P. Adjacent vertices then have the distance 1. In this context, escaping a local optimum may require traversing a greater distance with one mutation operator than it is the case with some other mutation operator. This is important, because the probability of performing the chain of mutations that leads to escaping a local optimum falls rapidly with the length of the mutation chain.

The simple conclusion is that it makes sense to exchange mutation operators during the search. Once an algorithm using some mutation operator gets stuck in a local optimum, changing the mutation operator may increase it's odds of escaping it.

3. Combining multiple mutation operators

During the search, a genetic algorithm may get stuck in a local optimum. The population converges, meaning that all of the individuals are close to the local optimum and the probability of performing the right chain of mutations to escape is low. As we have shown in section 2, once this happens, it makes sense to combine different mutation operators to increase chances of escaping local optima. It remains to be seen which mutation operators could this be – this is probably problem specific, and possibly also problem instance specific.

We will now describe the *shift mutation*. A very similar operator called *displacement mutation* was proposed by Michalewicz in [7]. One use of a similar operator is also mentioned in [3], in the context of closest substring problem.

The shift mutation starts by selecting a random subsegment of the permutation and a target position. It then shifts the elements of the selected subsegment to the target position. Figure 3 shows how this operator works on the permutation of order 8.

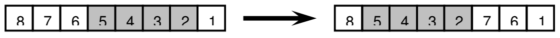

Figure 3. Example of the shift mutation

Testing the shift mutation within a tournament genetic algorithm applied to the traveling salesman problem showed that the performance of the algorithm using the shift mutation is comparable to that of one using insert mutation, better than the one using swap mutation, but worse than the one using inversion mutation.

While studying the behaviour of tournament genetic algorithm, we have noticed that it often gets stuck in local optima for greater number of cities. Figure 4 shows an instance of the traveling salesman problem which belongs to the class of problems we called Multicircular problems. The points represent the cities, and the left solution represents a global optimum – the shortest path needed to traverse all the cities. The right solution is a local optimum for the inversion mutation – no inversion will yield an individual with a higher fitness.

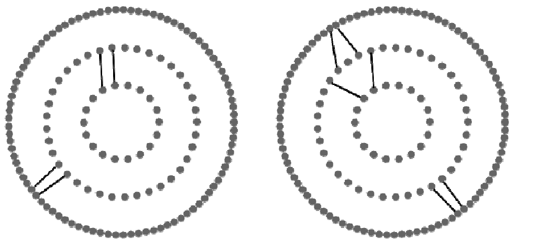

Figure 4. Global and local optimum for a Multicircular problem instance

By studying this solution and the belonging permutation, we have noticed that applying a shift mutation may actually make it possible for inversion mutation to generate an individual with a higher fitness. Figure 5 shows another example of this in more detail. Not all of the cities are shown – the dotted line represents omitted cities to make the figure more clear.

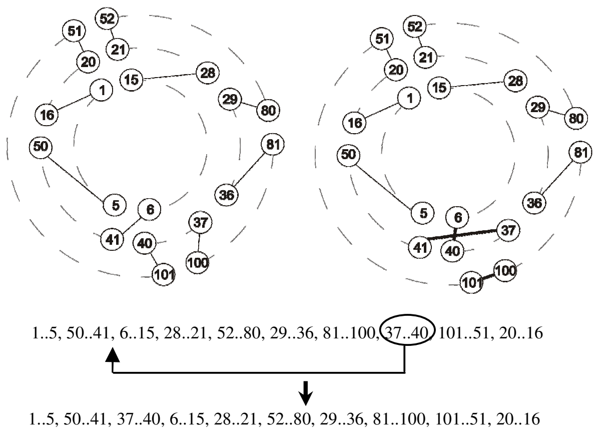

Figure 5. Applying a shift mutation

The left solution on Figure 5 represents a local optimum. It makes sense to try to make the cities belonging to the central circle adjacent, for instance, to make city 41 adjacent to cities 37 through 40. It can be observed from the belonging permutation that this could be achieved by shifting the subsegment 37 through 40 all the way to city 41. We, thus, obtain the right solution on the figure. It can be easily observed that inverting subsegment 37 through 40 yields a solution with a higher fitness.

Such examples formed a basis for the conclusion that inversion and shift mutation are a pair of mutation operators that work well in tandem. It remains to be seen whether or not this is the case for other problem instances.

4. Adaptive tournament genetic algorithm

We have so far decided what to change within our adaptive mechanisms – we will be changing the mutation operator and mutation probability. According to the questions listed in [1], we have yet to answer how the change is made, what is the scope of the change, and what is the evidence upon which the change is made. We will be making changes adaptively and the scope of the change will be the entire population. The evidence for the change will depend on the used adaptive mechanism. We'll now describe the adaptive tournament genetic algorithm.

ATGA is a tournament elimination genetic algorithm, using a classic integer list permutation encoding for solution representation. While solving a TSP, this results in certain permutations to represent the same solution. This is ignored, because wise choice of operators can eliminate this problem. The algorithm starts by randomly initializing the population and evaluating all individuals. The algorithm then repeats the following procedure until a prespecifed number of iterations has elapsed or a solution with a fitness value lower than the prespecified cost is found. It selects a predefined number of individuals (tournament size) and eliminates a number of individuals with the lowest fitness. It then randomly selects the remaining individuals in the tournament and performs crossover until it generates the number of individuals that were eliminated. For these experiments, we have used the order crossover, described in [6]. Each child is mutated with a predefined probability. After a prespecified number of iterations has elapsed (adaptation period), an adaptation mechanism is triggered.

4.1. Mutation probability varying

This method tracks the number of iterations that have passed since the last improvement of the best fitness value in the population. If this number is greater than the adaptation period, mutation probability is increased by prespecified value (mutation probability increase), and the counter value is set to zero. Once the best fitness value in the population improves, the mutation probability is decreased eight times the mutation probability increase. The idea is to quickly return to the initial mutation probability level. An additional rule is that mutation probability cannot get greater than 1 or smaller than the initial probability level. This way the mutation probability increases if the search gets stuck in a local optimum.

An excellent summary of work on dynamic mutation rates by Fogarty, Hesser and Männer, Bäck and Schütz, and others is given in [1].

4.2. Adaptive operator cycling

This method relies on a predefined list of mutation operators. During its work it exchanges the used mutation operators from that list stochastically.

Adaptive operator cycling (AOC) is triggered once the value of a special iteration counter becomes greater than the adaptation period, and at least one mutation has occurred since the last adaptation. A ratio of successful to all mutations r_s is then calculated, where a successful mutation occurs when the fitness value of the individual increases. The probability of operator exchange p_{oc} is then calculated according to the equation (1).

$$p_{oc} = 1 - r_s \qquad (1)$$

An analysis of this equation reveals that the probability of operator exchange is equal to 1 when no mutation is successful, and equal to 0 when all mutations are successful. This makes sense, because we want to exchange less successful operators with a greater probability. However, for increased sensitivity, we have used equation (2), which favours extremes (success rates close to 1 will cause operator change even less often, and vice versa).

$$p_{oc} = \frac{1}{2}\left(1 + \cos\left(r_s \cdot \pi\right)\right) \qquad (2)$$

There is an alternative to this method we called AOC2, which works in a similar manner as does AOC. The only difference is that the adaptation mechanism is triggered only after no improvement in the best fitness value occurred until the adaptation period has elapsed – this is interpreted as being stuck in a local optimum.

4.3. Mutation operator statistics

Mutation operator statistics is a method that tracks the success rate of each operator on the mutation operator list, deciding which operator to use before applying the mutation to the individual. This decision is biased by the success of the mutation operator relative to others.

Initially, all operators are given equal probability to be chosen. After the adaptation period has elapsed, this method calculates the ratio of successful mutations r_{si} and draws a random value b_i according to equation (3), where rand(0, 1) is a random value from the interval [0, 1] drawn from a uniform distribution. This random value b_i serves as a small positive or negative offset.

$$b_i = \left(rand(0,1) - 0.5\right) \cdot a \tag{3}$$

It then assigns the probability to be chosen p_{chi} to each operator which is defined by equation (4). The probability is proportionate to operator success rate.

$$p_{chi} = \frac{\max(0, r_{si} + b_i)}{\sum_j \max\left(0, r_{sj} + b_j\right)} \tag{4}$$

The reasoning behind the value b_i is to avoid that a certain operator gets assigned a zero probability to be chosen, thus being unable to be used later during the search, when it could prove useful. The value of the coefficient a is experimentally chosen to be 0.02. Lower values resulted in a slow change of p_{chi}, while higher values introduced an offset b_i which favoured wrong operators and thus interfered with this credit assignment mechanism.

5. Experimental results

We have compared the performance of ATGA with the version without adaptive mechanisms on a number of instances of TSP. We will show test results for five different problem instances. These are *kroA200*, *pr299*[1], *Spiral250*[2], *Multicircular250* and *Multicircular500*. We've set the tournament size to 3, and the number of individuals to eliminate to 1. Mutation probability was set to 0.5. Population size was set to 50. We have experimentally shown that the algorithm without adaptation mechanisms shows best performance for these values. The adaptation period was set to 25000 iterations. Mutation probability increase was set to 0.025. Lower adaptation period has been experimentally shown to exhibit unstable performance (information used for parameter adaptation became less exact), and higher has influenced parameters too slowly. Vice versa is true for mutation probability increase. The algorithm started using the inversion operator, and the second operator used was shift mutation.

Figure 6 shows the differences in performance between various adaptation methods. When using no adaptive mechanism, the search gets stuck in a local optimum after 10^6 iterations.

[1] TSP instances *kroA200* and *pr299* may be downloaded from TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/)

[2] A set of 250 cities arranged in a spiral

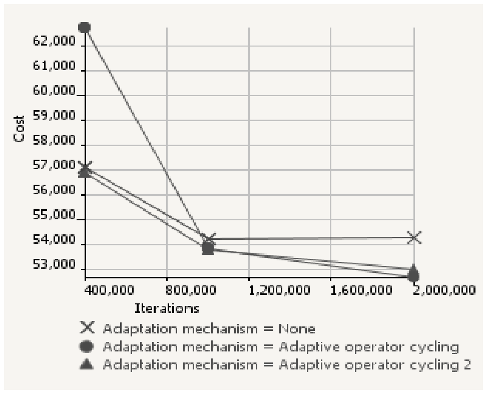

Figure 6. Test results for pr299

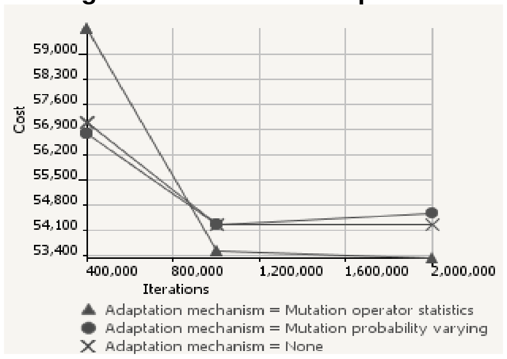

Figure 7. Test results for pr299

AOC exchanges shift and inversion operators before first local optimum is detected, thus performing worse in the initial period of the search, because shift mutation isn't the best operator for this type of problem. However, once a local optimum is found (usually after 10^6 iterations), the operator exchange mechanism yields better solutions. AOC2 doesn't exchange operators before reaching a local optimum, and is efficient even in the initial period. Other adaptive methods are only shown for *pr299*, but exhibit a similar behaviour as AOC.

Finally, for the *Multicircular500* problem better results were obtained after a greater number of iterations. Adaptive operator cycling found better solutions after approximately $4 \cdot 10^6$ iterations.

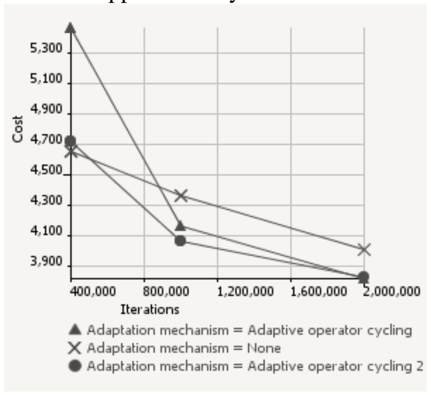

Figure 8. Test results for Multicircular250

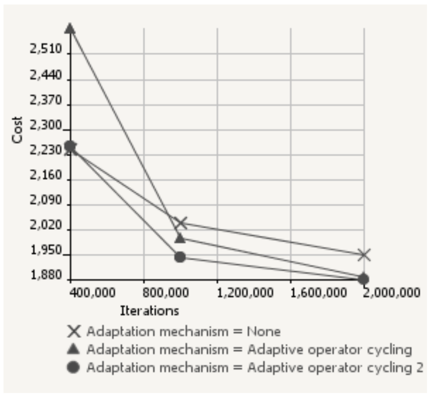

Figure 9. Test results for Spiral250

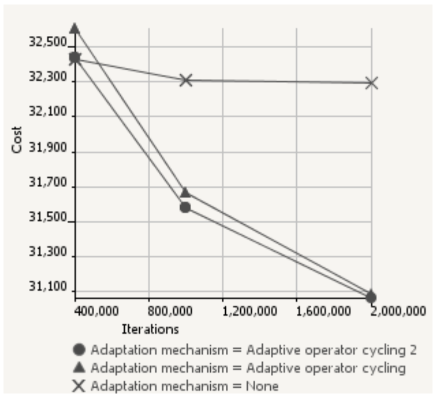

Figure 10. Test results for kroA200

Table 1 shows test results for 25 test runs for each problem instance and mechanism. Average best cost, minimum and maximum best costs, standard deviation and median are shown in the table. Each cell contains 3 values – top value obtained for using no adaptation, middle for AOC and the bottom value for AOC2. In most cases AOC2 shows best performance.

6. Conclusion

We have explained in section 2 that exchanging mutation operators transforms the fitness landscape in terms of distribution of local optima. Changing a mutation operator once reaching a local optimum can increase chances of escaping it. In section 3 we've hypothesized that combining inversion and shift mutation operators could increase performance of an algorithm solving a TSP.

It can be concluded from experiments that altering mutation operators during the search gives positive results in terms of escaping local optima. We have, thus, confirmed our hypothesis from section 2.

Table 1. Test results for no adaptation (top), AOC (middle) and AOC2 (bottom)

Problem instance	Avg	Min	Max	Standard deviation	Median
kroA200	32268 31244 31250	31149 30560 30286	33241 32217 32633	610 410 458	32285 31171 31202
Spiral250	1956 1864 1820	1739 1739 1739	2485 2053 1924	212 88 70	1904 1841 1821
M.c.250	4119 3930 3855	3724 3723 3723	4914 4595 4576	444 298 268	3882 3728 3726
pr299	54107 52865 53249	52069 51771 51335	56000 54632 55322	977 781 895	54218 52860 53187
M.c.500	4082 4348 4070	3685 3702 3685	4921 5030 5061	532 412 492	3689 4384 3716

However, these observations raise further questions. For instance, what are the good combinations of mutation operators? Can this be deducted from the problem instance? An interesting research would be modifying existing mutation operators to be adaptive in terms of length of the subsegment on which they operate. These are all topics that will be addressed in the future.

7. References

[1] A. E. Eiben, R. Hinterding, Z. Michalewicz, "Parameter Control in Evolutionary Algorithms", Parameter Setting in Evolutionary Algorithms, Studies in Computer Science, Springer, Berlin, 2007., pp. 19-46

[2] S. Yang, "Adaptive Crossover in Genetic Algorithms Using Statistics Mechanism", Proceedings of the 8. International Conference on Artificial Life, MIT Press, Cambridge, 2002., pp. 182-185

[3] H. Mauch, "Closest Substring Problem – Results from an Evolutionary Algorithm", Neural Information Processing, LNCS, Springer, Berlin, 2004., pp. 205-211

[4] K. De Jong, "The Handbook of Evolutionary Mutation", IOP Publishing Ltd and Oxford University Press, 1997., pp. C3.2:2

[5] D. Ashlock, "Evolutionary Computation for Modeling and Optimization", Springer, Canada, 2005.

[6] A.E. Eiben, J.E. Smith, "Introduction to Evolutionary Computing", Natural Computing Series, Springer, Berlin, 2003.

[7] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer, Berlin, 1992.

[8] M. Bhattacharyya, A. K. Bandyopadhyay, "Comparative Study of Some Solution Methods for Traveling Salesman Problem Using Genetic Algorithms", Cybernetics and Systems, Taylor&Francis, Inc., Bristol, 2009., pp. 1-24