Non-revisiting Genetic Algorithm with Constant Memory

Yang Lou
Department of Electronic Engineering
City University of Hong Kong
Hong Kong SAR, China
felix.lou@my.cityu.edu.hk

Shiu Yin Yuen
Department of Electronic Engineering
City University of Hong Kong
Hong Kong SAR, China
kelviny.ee@cityu.edu.hk

Abstract—The continuous Non-revisiting Genetic Algorithm (cNrGA) uses the entire search history and parameter-less adaptive mutation to significantly enhance search performance. Experimental results show that it has better performance than Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a state of the art evolutionary algorithm. Storing the search history is natural and costs little when fitness evaluations are expensive. However, if the number of evaluations required is substantial, some memory management is desirable. In this paper, we propose two pruning mechanisms to keep the memory used constant. They are Random pruning and Least Recently Used pruning. The idea is to prune a node when a memory threshold is reached and a new node is required to be inserted, thus keeping the overall memory used constant. Experimental results show that both strategies can maintain the performance of cNrGA, up to the limit when 90% of the nodes are not recorded. This suggests that cNrGA can be extended to use in situations when the number of fitness evaluations are much larger than before with no significant effect on statistical performance, which widens the applicability of cNrGA to include more practical problems that require larger number of fitness evaluations before converging to the global optimum.

Keywords—Non-revisiting genetic algorithms; least recently used pruning; random pruning; binary space partition tree

I. INTRODUCTION

Revisiting evaluated solutions in Evolutionary Algorithms (EAs) is wasteful and distorts the true performance of algorithms [1]. Yet only a few existing algorithms have employed mechanisms to avoid re-evaluations. One such example is the Tabu search [2], which uses a set of rules, namely the Tabu list, recording a portion of the search history, to partially avoid re-evaluations. Another example is the continuous Non-revisiting Genetic Algorithm (cNrGA) [3,4], which records the entire search history. (The subscript c means that the version of NrGA [5] is designed for continuous variables.) Thus cNrGA completely avoids re-evaluation of any evaluated solutions. In the field of evolutionary computation, it is often assumed that the computational cost of function evaluations is the most expensive one, much greater than the solution generation cost of the evolutionary algorithms per se. In such a situation, the usage of memory to store the search history in memory is practically affordable and acceptable. Meanwhile, the price of memory is cheaper than ever as new hardware technologies are developing, and will be cheaper in the future. Storing the search history is natural and costs little when fitness evaluations are expensive. However, cNrGA could not deal with the problems well when the number of function evaluations is substantial, because 1) the usage of memory to store the evaluated solutions would become uncontrollably large; 2) the sub-region for adaptive mutation may become too small, thus the effectiveness of mutation would be significantly weakened; and 3) as the accumulated amount of search history becomes larger, it needs longer time to check for revisits and perform other operations to the memory archive. Thus memory management for cNrGA is desirable. Practically, cNrGA has experimented with using 40,000 as the maximum number of function evaluations [3-5], which makes a compromise between the performance of the algorithm and the archive size. Experimental results in [4] shows that cNrGA outperforms the state of the art Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [6,7], which is one of the best performing evolutionary algorithms to date. It shows strong optimizing abilities in some recent competitions. Variants of CMA-ES [8,9] did very well in the CEC 2013 competition [10], BBOB 2012 and 2013 competition [11].

To widen the applicability of cNrGA, especially when it requires substantial number of evaluations, we propose a novel pruning mechanism to maintain the memory used constant. We set a memory threshold as the maximum storage we want to invest, and when the threshold is reached, a node will be deleted so that the newly generated solutions can be stored, keeping the memory constant. In this paper, we propose two pruning mechanisms to keep the memory used constant. They are 1) pruning the Least Recently Used (LRU) searching information, and 2) pruning the Randomly (R) selected information. In acronym, the cNrGA/CM/LRU and cNrGA/CM/R are proposed, where CM is short for constant memory. Both variants of cNrGA keep constant the overall memory to store the evaluated solutions, so that both are able to deal with situations when the number of function evaluations is very large. Using this useful extension, our method removes the limit to the maximum number of evaluations in cNrGA.

In contrary to the Tabu list, which stores a portion of the search history, cNrGA/CM/LRU and cNrGA/CM/R delete a portion from the entire history. On the one hand, using constant memory leads to a loss of historical information, and makes
cNrGA not completely non-revisiting; yet on the other hand, using constant memory does not impose restrictions on the maximum number of function evaluations, thus, our method widens the applicability of cNrGA to include more practical problems that require larger number of fitness evaluations before converging to the global optimum. Two other advantages are 1) due to the limit of partitioning the search space, it offers relatively larger sub-region for adaptive mutation; and 2) visiting an archive with constant size is faster and is of constant speed.

In the extreme case, the cNrGA/CM/LRU and cNrGA/CM/R may maintain the same amount of history information as the Tabu search does. However, cNrGA uses the binary space partitioning (BSP) tree, a more sophisticated data structure than the Tabu list employed by Tabu search, in organizing the search archive. The operations of cNrGA/CM/LRU and cNrGA/CM/R are also different from the Tabu list, namely, the Tabu search does not support the parameter-less adaptive mutation, a key operator in cNrGA.

The rest of the paper is organized as follows. Section 2 introduces the archive organization structure of cNrGA, the two pruning strategies, and the interpretation of the pruning strategies as novel parameter-less adaptive mutation operator for cNrGA. Section 3 reports the experimental results. Section 4 draws the conclusions.

II. LEAF NODE PRUNING

A. Archive Organization in cNrGA

In cNrGA, all the evaluated solutions are stored in the BSP tree. Thus each leaf node of the BSP tree contains an evaluated solution, as well as its allocated sub-region. The entire BSP tree T corresponds to the whole search space S. A leaf node li in T includes an evaluated solution xi (including the solution position in the search space, the fitness value, etc.), and its allocated sub-region hi in S. Figure 1 gives an example in 2-dimensional search space. In Figure 1(a), the search space (right) is partitioned into two sub-regions by the generation of two solutions a and b. Note that in the BSP tree (left), the left child node a stores the same solution as the root node a, while the child b has a smaller sub-region than its parent. Actually, the root node a is a virtual node of the left child node a, and in this case, the archive consists of the leaf nodes when virtual nodes are not counted. The dot-dash line represents the partition of the search space. Then, when the next solution c is generated, it is inserted into the BSP tree and allocated with a sub-region, as in Figure 1(b). During the running of cNrGA, the size of the BSP tree grows as function evaluations increase. Assume the current number of function evaluations is n, then the number of leaf nodes in the BSP tree is also n. Because of the presence of virtual nodes, though the entire BSP tree consists of 2n – 1 nodes, the usage of memory to store the search archive by far is n units (nodes). Full details on how the BSP tree is constructed, as well as illustrative examples, can be found in [3-5]. Source code of cNrGA is also available in [12].

The more solutions it has generated, the larger is the BSP tree, and the smaller the sub-region allocated for each solution on average. In cNrGA, the offspring is reproduced by crossover, and if a revisit occurs after checking with the BSP tree archive, then a parameter-less adaptive mutation is performed within the corresponding sub-region and its neighborhoods [3-5].

B. Two Pruning Mechanisms

For discrete (combinatorial) optimization problems, NrGA [5] prunes a sub-tree when all the possible solutions within its sub-region have been evaluated, and no more searching will be performed within it. However, in the continuous space, it is impossible to traverse all the possible solutions within an arbitrarily small sub-region, and therefore impossible to prune the sub-tree which contains more than one solution. Thus, our strategy to keep the usage of memory constant is to prune leaf nodes one at a time, rather than a sub-tree containing more than one nodes. As all the search information is stored only in the leaf nodes, and the non-leaf nodes are virtual nodes, pruning a leaf node removes search information.

The basic idea is that when the algorithm reaches the threshold of memory defined by the user, it would online prune one old leaf node and then add back one new leaf node, in general, in a different location in the search tree as directed by the cNrGA search mechanism [3,4]. So the memory will be constant.

The new leaf node refers to the most recently generated solution and its auxiliary information, while the old one to be pruned can be chosen in several ways. In this paper, we propose two mechanisms to choose the leaf node to prune. They are the Least Recently Used (LRU) pruning and the Random (R) pruning:

1) The Least Recently Used (LRU) Pruning

To apply the LRU pruning mechanism, a time stamp ts, recording the time when the tree node is constructed, is attached together with the solution and stored in the archive. Thus a leaf node li is an attribute set {ts, x, h}, representing its time stamp, solution and its allocated sub-region respectively. Then it is straightforward to identify the LRU leaf node. An
LRU leaf node means that its time stamp value is the smallest \((i.e., \text{oldest})\) amongst all the current leaf nodes. In effect, its sub-region is the least recently exploited, or in other words, the algorithm has not suggested to search in the sub-region for a long time.

Technically, not all leaf nodes can be pruned because of the structure of the BSP tree. If the identified LRU leaf node stores the same solution as its parent, \(i.e.,\) the parent is a virtual node of the LRU leaf node, then it is inappropriate to prune it, because the parent virtual node becomes meaningless if the real child leaf node corresponding to it is pruned. This may affect virtual nodes further up the tree also. For example, the grandparent of a (real) leaf node may also be a virtual node, and in the worst case, the ancestor virtual node may be traced all the way back to the root node. If so, the entire BSP tree has to be drastically reorganized As a result, only the LRU leaf node storing a different solution from its parent can be pruned, \(i.e.,\) those leaf nodes without virtual nodes as their parents. For example, in Figure 2(a), it is inappropriate to prune the leaf node \(a\) due to its virtual node; while in Figure 2 (b), the leaf node \(b\) may be pruned.

![LRU pruning example](image)

Figure 2 (b) to (d) gives an example to illustrate LRU pruning in detail. In Figure 2, the “…” symbol above the BSP tree mean the top node may not be the root of the whole tree, and the “…” symbols at the bottom of the tree means those nodes may not be leaf nodes. The leaf node \(b\) in Figure 2(b) is
to be pruned. Figure 2(c) shows the tree after \(b\) is removed. The shaded part is the sub-region allocated for \(b\). In Figure 2(d), it is partitioned into two parts, which are merged into the sub-regions of \(a\) and \(c\), respectively. If \(a\) and/or \(c\) are non-leaf nodes, the sub-region will be further re-partitioned. In this way, the LRU leaf node \(b\) is pruned successfully.

Note that pruning node \(b\) enlarges the sub-regions of both \(a\) and \(c\), and if \(a\) and/or \(c\) are not leaf nodes, it enlarges the sub-regions of all the leaf nodes under the sub-tree of \(a\) on average. As a result, the pruning mechanism enlarges the sub-regions for adaptive mutation.

2) The Random (R) pruning

The second pruning mechanism is to prune a randomly selected leaf node. The idea is to randomly go down the BSP tree, starting from the root node of the BSP tree, and randomly choose the left or right child to go through, with equal probability, until it reaches a leaf node. If the leaf node has no virtual node, then prune it. Otherwise, repeat the random traversal process starting from its sibling until it finds a leaf node that has no virtual node. For example, if leaf node \(b\) is reached in Figure 2(b), then this node can be pruned right away. However, if leaf node \(a\) is reached in Figure 2(a), then the random search goes down the sub-tree with node \(b\) at its head. This process will always terminate because the BSP tree is constructed such that a virtual parent node has at least one real child node.

The random pruning has less computational burden than the LRU pruning, as it only randomly finds a usable leaf node.

The LRU leaf node can be found by traversing the entire tree once. In implementation, we keep a list of leaf nodes and sort it according to its time stamp. When pruning is required, we prune the first node in the list. This continues until the whole list is exhausted. Then the entire tree is traversed again and a new list of leaf nodes is constructed.

C. Novel Parameter-less Adaptive Mutation Operators

Interestingly, the LRU and R pruning strategies may be interpreted as novel parameter-less adaptive mutation operators:

Both pruning strategies can be regarded as enlarging the mutation range \((i.e., \text{the sub-region size})\) for the adaptive mutation operator, which is a key operator in cNrGA. As both pruning strategies do not introduce any additional parameter, the operator is parameter-less.

LRU pruning operator: As LRU pruning deletes the least recently used leaf nodes, the average sub-region size of a leaf node increases, but non-uniformly. Older nodes are increased more than recent nodes. As the mutation range is the sub-region size, the strategy is to allow a larger exploration if a long ago unvisited sub-region is visited again.

R pruning operator: As R pruning deletes leaf nodes randomly, the average sub-region size of a leaf node increases uniformly. The strategy is thus to increase the mutation range, while keeping the structure of the mutation strategies. That is, leaf nodes with deeper depths are mutated less, meaning more exploitation, and vice versa.

1716
III. EXPERIMENTAL STUDY

In this section, we examine the performance of cNrGA/CM/LRU and cNrGA/CM/R by comparing it to the original version of cNrGA [4]. The key issue is to find out if the pruning mechanisms affect the performance of cNrGA significantly, and investigate whether the pruning brings better mutation conditions for the optimization.

A. Test Functions

The 28 benchmark functions in CEC 2013 test suite [9] are employed. The dimension of functions is set to be $D = 30$. The Maximum number of Function Evaluations (MaxFEs) is chosen to be 40,000, 50,000, 60,000, when the Memory Threshold (MT) is set to be 30,000 for cNrGA/CM/LRU and cNrGA/CM/R, and MaxFEs = 100,000 when the MT is 10,000. Since the MT is smaller than the MaxFEs, some portion of history information is lost. Table I shows the loss of search information with each setting. For cNrGA, the setting is MaxFEs = MT.

<table>
<thead>
<tr>
<th>MaxFEs</th>
<th>40,000</th>
<th>50,000</th>
<th>60,000</th>
<th>100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT</td>
<td>30,000</td>
<td>30,000</td>
<td>30,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Information Loss</td>
<td>25%</td>
<td>40%</td>
<td>50%</td>
<td>90%</td>
</tr>
</tbody>
</table>

To evaluate the effect of the pruning fairly, for cNrGA, cNrGA/CM/LRU, and cNrGA/CM/R, we use the same parameters suggested in [4]. Thus the population sizes are set to 100 and the crossover rate is set 0.5. Also, in each run, the three algorithms are identical before pruning occurs. However, as soon as MT is reached, cNrGA/CM/LRU and cNrGA/CM/R each prunes one node from their respective BSP tree, then inserts the new node representing the new solution into the tree. This is repeated for each new solution generated. Hence the memory used is kept constant. For cNrGA, no pruning is done and the new solutions are simply inserted into the BSP tree, making the tree larger and larger. As the composition and structures of the trees become different as MT is reached, the three algorithms have different search strategies after MT is reached.

B. Results

The performance of an algorithm is measured by statistics from 30 independent runs.

Table II in the Appendix show the mean results, the standard deviations, and the p-values of Mann-Whitney U test (with significance level $\alpha = 0.05$) obtained by the three algorithms when 25%, 40%, 50% and 90% search information is lost (refer to Table I). The best mean results amongst the three algorithms for each problem are shaded in gray.

For the 25%, 40%, 50% cases, no algorithm is found to be significantly different from the other two. For the 90% case, no algorithm is significantly different from the other two in 25 functions. cNrGA outperforms cNrGA/CM/LRU significantly in solving F1 and F5, and outperforms cNrGA/CM/R in F4 and F5.

The algorithms are ranked using their mean fitness values. Table III shows the average ranks. Note that cNrGA/CM/LRU has the smallest average rank in the 25% case, cNrGA/CM/R in the 40% case, cNrGA in the 50% case, but interestingly, in the 90% case, it is cNrGA/CM/R that has the smallest average rank.

IV. CONCLUSIONS

The Continuous Non-revisiting Genetic Algorithm (cNrGA) employs the binary space partitioning (BSP) tree to store the entire search history, so that the parameter-less adaptive mutation can be performed via the partitioned search space. By keeping the entire search history, significant search performance gain has been observed by using the search history to advise the search. cNrGA has been shown to outperform state of the art methods such as the covariance matrix adaptation evolution strategy (CMA-ES) in continuous optimization problems.

Though it is reasonable and indeed natural to store the entire search history when the application involves expensive fitness function evaluations, it is interesting if one can extend the applicability of cNrGA when the number of evaluations is larger.

In this paper, we propose two pruning strategies which keep the memory usage in cNrGA constant throughout the search process. Thus in principle, cNrGA can be used for all kinds of search problems, including those involving large number of function evaluations. The pruning strategies are least recently used pruning and random pruning.

Interestingly, the two pruning strategies can be regarded as novel parameter-less adaptive mutation operators, which modify the search strategies of cNrGA fundamentally.

The experimental results reveal that these pruning strategies may not degrade the performance of cNrGA. With up to 90% pruning, the performance of the two cNrGA variants and the original cNrGA does not show any significant difference.

As a result, preliminarily, one may conclude that the proposed pruning mechanisms for memory management widens the applicability of cNrGA to include more practical problems that require larger number of fitness evaluations before converging to the global optimum, and the pruning strategies furnish novel, effective parameter-less adaptive mutation strategies.

ACKNOWLEDGMENT

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 125313]. We thank Dr. Chi Kin Chow for his initial work on discrete version of NrGA.


http://coco.gforge.inria.fr

TABLE III. AVERAGE RANKS OF cNrGA, cNrGA/CM/LRU, AND cNrGA/CM/R

<table>
<thead>
<tr>
<th></th>
<th>MaxFEs = 40,000</th>
<th>MaxFEs = 50,000</th>
<th>MaxFEs = 60,000</th>
<th>MaxFEs = 100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1 = 30,000</td>
<td>M1 = 30,000</td>
<td>M1 = 30,000</td>
<td>M1 = 10,000</td>
</tr>
<tr>
<td>cNrGA CM/LRU</td>
<td>1.96</td>
<td>1.86</td>
<td>2.11</td>
<td>1.89</td>
</tr>
<tr>
<td>cNrGA CM/R</td>
<td>1.93</td>
<td>2.07</td>
<td>2.04</td>
<td>2.19</td>
</tr>
<tr>
<td>Average Rank</td>
<td>1.96</td>
<td>1.86</td>
<td>2.11</td>
<td>1.89</td>
</tr>
</tbody>
</table>

1719