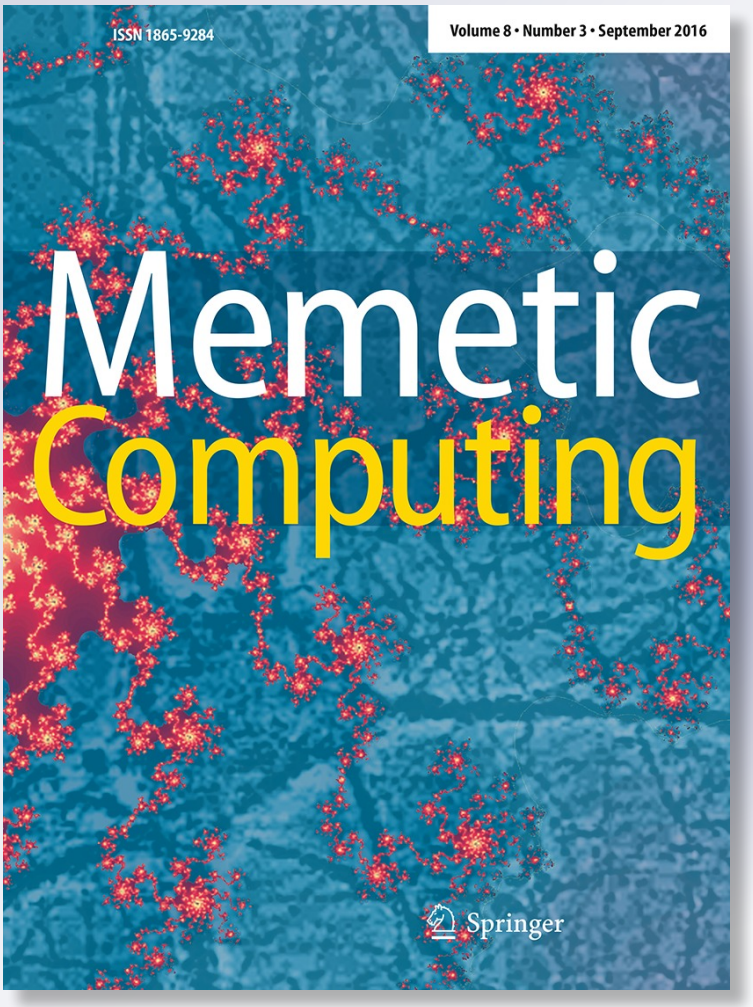
Non-revisiting genetic algorithm with adaptive mutation using constant memory

Yang Lou & Shiu Yin Yuen

Memetic Computing

ISSN 1865-9284
Volume 8
Number 3

Memetic Comp. (2016) 8:189-210
DOI 10.1007/s12293-015-0178-6



Springer

Memetic Comp. (2016) 8:189–210
DOI 10.1007/s12293-015-0178-6

REGULAR RESEARCH PAPER

Non-revisiting genetic algorithm with adaptive mutation using constant memory

Yang Lou[1] · **Shiu Yin Yuen**[1]

Received: 13 July 2015 / Accepted: 17 December 2015 / Published online: 5 January 2016

Abstract The continuous non-revisiting genetic algorithm (cNrGA) uses the entire search history and parameter-less adaptive mutation to significantly enhance search performance. Storing the entire search history is natural and costs little when the number of fitness evaluations is small or moderate. 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 least recently used pruning and random pruning. The basic idea is to prune a unit of memory when the memory threshold is reached and some new search information is required to be stored, thus keeping the overall memory used constant. Meanwhile, both pruning strategies naturally form parameter-less adaptive mutation operators. A study is carried out to evaluate the impact on performance caused by loss of search history information. Experimental results show that (1) both strategies can maintain the performance of cNrGA, up to the empirical limit when 90 % of the search history is not recorded, (2) cNrGA and its variants with constant memory outperform the real-coded genetic algorithm and the standard particle swarm optimization. By pre-extracting all the current prune-able history information and storing them into a list, namely, to-prune-list, the overhead of both pruning strategies becomes small. This suggests that cNrGA can be extended to use in situations when the number of fitness evaluations is much larger than before with no significant effect on statistical performance. This widens the applicability of cNrGA to include more practical problems that require larger number of fitness evaluations before converging to the global optima.

Keywords Non-revisiting genetic algorithms · Least recently used pruning · Random pruning · Binary space partition tree

✉ Shiu Yin Yuen
kelviny.ee@cityu.edu.hk

Yang Lou
felix.lou@my.cityu.edu.hk

[1] Department of Electronic Engineering, City University of Hong Kong, Hong Kong, China

1 Introduction

Revisiting evaluated solutions in evolutionary algorithms (EAs) is wasteful and distorts the true performance of algorithms [1]. Not only the no free lunch (NFL) theorems declare that all algorithms without revisits have on average the same performance assuming that the distribution of all the problems is uniform [1], but also many empirical studies reveal that duplicate removal improves the performance of genetic algorithm (GA) significantly [2–7]. Yet only a few existing algorithms have employed mechanisms to avoid re-evaluations. One such example is the Tabu search [8], which uses a set of rules and banned solutions, 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) [9,10], which records the entire search history. (The subscript c means that the version of NrGA [7] is designed for continuous variables.) Thus cNrGA completely avoids re-evaluation of any evaluated solutions. The non-revisiting mechanism has also been applied to algorithms other than genetic algorithm, such as non-revisiting simulated annealing (NrSA) [11], and non-revisiting particle swarm optimization (NrPSO) [12]. Let call them by a common name non-revisiting stochastic search (NrSS). The NrSS framework is shown in Fig. 1. It can be considered as a communication process between the stochastic search method and the stored historical information

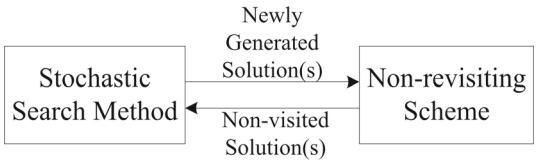


Fig. 1 Framework of non-revisiting stochastic search

[11]. Before the newly generated solutions are evaluated, they should first pass the revisit-check. More than simple checking and warning of revisit, the non-revisiting scheme offers search suggestions using history information, e.g. for NrPSO, the non-revisiting scheme redirects the revisited particle as if it has collided with the obstacle, and the reactive force leads the particle to a random yet non-visited position. In NrSA, a revisit always leads the search to its nearest neighbor in the discrete domain. For any stochastic search, an archive can be added for revisit checking. Moreover, the information in the archive may be more efficiently used to provide a parameter-less adaptive local search operator [7]. In NrSS, the synergy of global search, local search and learning historical information forms a memetic paradigm [13].

In NrSS, a revisit means it generates a solution which is exactly the same as one that has been generated and evaluated in the search history. Recently in the enhanced vine creeping optimization (EVCO), the authors redefined a revisit as generating a solution which is bounded by the convex set of a convergence region, not necessarily exactly the same as any solution in the history. In EVCO, the non-revisiting scheme is applied for global exploration [14].

In continuous domain or numerical optimization, it is much less often than in the discrete domain or combinatorial optimization to generate a solution which is identical to another solution that has been evaluated previously in the search history. Different strategies incur different probabilities (risk) of revisits. For example, we observe in our experiments that if the boundary handling method is the so-called *absorbing* scheme, i.e., if the values of the parameters exceeding the boundaries are set to the values at the boundaries, the EA suffers from a non-negligible number of revisits on boundaries. The *absorbing* scheme is typically employed by artificial bee colony (ABC) [15,16], and widely employed by differential evolution (DE) [17–19] and particle swarm optimization (PSO) [20,21]. In contrast, covariance matrix adaptation evolution strategy (CMA-ES) [22,23] does not employ such *absorbing* scheme and even does not require a set of known boundaries; as a result, negligible or no revisits occur in the search process of CMA-ES by nature. In addition, premature convergence of any stochastic search algorithm will also generate revisits.

The price of memory is cheaper than ever as new hardware technologies are developing, and will be still cheaper in the future, as long as the development of hardware technology continues to follow Moore's Law [24]. Storing the search history is natural and costs less and less. Thus even if an algorithm has no revisits in the continuous domain, storing the entire search history enables better decisions to be made [7]. On the other hand, cNrGA records the entire search history and makes sure that revisits occur due to crossover. This suggests that a design philosophy such as that in cNrGA will give better search performance. Rather than only to prevent revisits, more importantly, cNrGA employs a framework that derives benefit from revisits. Note that cNrGA produces more revisits than conventional stochastic search methods. Because cNrGA basically employs only selection and crossover, the probability of revisits is high. Revisits are expected and when it happens, cNrGA will perform parameter-less adaptive mutation to create new gene in the current population.

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 may become exceeding, (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 a longer time to check for revisits and perform other operations to the memory archive. Thus memory management for cNrGA may be desirable in some practical situations. Practically, cNrGA has experimented with using 40,000 as the maximum number of function evaluations [7,9,10], which makes a compromise between the performance of the algorithm and the archive size. When the number of evaluations is 40,000, experimental results in [10] show that cNrGA outperforms the state of the art CMA-ES [22,23], which is one of the best performing evolutionary algorithms to date.

To widen the applicability of cNrGA, especially when it requires substantial number of evaluations, we propose two novel pruning mechanisms 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 unit is deleted so that the newly generated solutions can be stored, keeping the memory constant. The two pruning mechanisms 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. Employing this useful extension, our method removes the limit to the maximum number of evaluations in cNrGA. Referring to Fig. 1, we can limit the memory usage of the non-revisiting scheme by pruning strategies, such that all the non-revisiting stochastic search methods could employ a variant using constant memory.

In contrary to the Tabu list, which stores a portion of the search history, cNrGA/CM/LRU and cNrGA/CM/R delete a portion of 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, it widens the applicability of cNrGA to include more practical problems that require larger number of fitness evaluations before converging to the global optimum.

If cNrGA concentrates on a particular region, the density of visiting that region may become large. Thus the sub-region size of a leaf node at that region may become small. If it concentrates to search on a few sub-regions, then cNrGA may miss the exact location of the global optimum. Thus two other advantages brought by pruning are (1) due to the limit of partitioning the search space, it offers relatively larger sub-region size 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, specifically, the Tabu search does not support the parameter-less adaptive mutation, a key operator in cNrGA. Moreover, it is interesting to note that the pruning operators may be regarded as novel parameter-less adaptive mutation operators per se. A preliminary version of this paper has appeared in [25]. In this paper, (1) the properties of the pruning operators, as novel parameter-less adaptive mutation operators, are studied in depth. (2) A comprehensive study on different experimental settings is done. (3) The computational costs of the proposed pruning strategies are also examined. (4) Finally, all three versions of cNrGA are compared with real-coded genetic algorithm (RGA) [26–28] and the standard particle swarm optimization (SPSO) 2011 [21].

The rest of the paper is organized as follows. Section 2 introduces the archive organization structure of cNrGA. Section 3 introduces the two pruning strategies, and the interpretation of the pruning strategies as novel parameter-less adaptive mutation operator for cNrGA. Section 4 reports the detailed experimental results, including performance comparison and analysis, and Sect. 5 draws the conclusions.

2 Continuous non-revisiting genetic algorithm

cNrGA employs BSP tree to store the entire search history, and each leaf node stores the search information of one solution. As the structure of BSP tree is efficient and easy to implement, it is commonly used as the data structure of archive [7,9–12,14]. Based on BSP tree, any position in the entire archive can be easily reached. Moreover, the information can be employed for fitness landscape approximation [29], and parameter control [30]. In cNrGA, the memory usage of a BSP tree is equal to the number of function evaluations. The more history information it accumulates, the more precise it can approximate the fitness landscape or predict the performance of using different parameters.

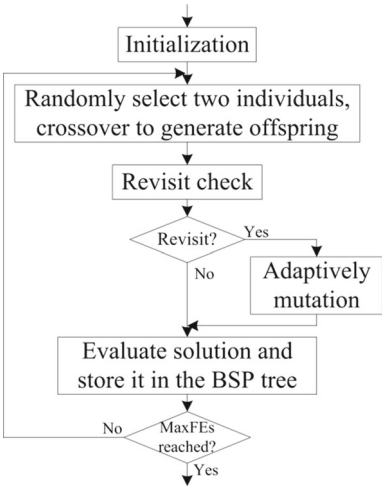


Fig. 2 Flowchart of the main loop in cNrGA

In cNrGA, in each generation a population of candidate solutions is generated by uniform crossover. Thus each component (dimension) of a new generated solution is from either of its two parents selected at random. The uniform crossover is prone to generate more and more solutions that have been evaluated previously as the number of evaluations increases, since the principle of crossover is to recombine the genes, rather than to generate new genes. These evaluated solutions are found by searching in the archive, where the BSP tree structure makes this process simple and efficient. After the check of revisit, all revisits will be identified so that they will not be evaluated. The mutation operator employed in cNrGA is named one-gene-flip (OGF) mutation [9], which is simple but effective. The OGF operator works as follows: if a solution is a revisit, then uniformly mutate on a randomly selected dimension. Figure 2 shows the flowchart of the main steps in cNrGA.

A revisited solution will perform adaptive mutation within its sub-region. A sub-region is a portion of the whole search space. Within an assigned sub-region, there is one and only evaluated solution. The sub-region is unique (one-to-one corresponding) to a stored solution. As the sub-region is a continuous region with infinite size, the mutant which is different from the existing solution would definitely be a non-visited one.

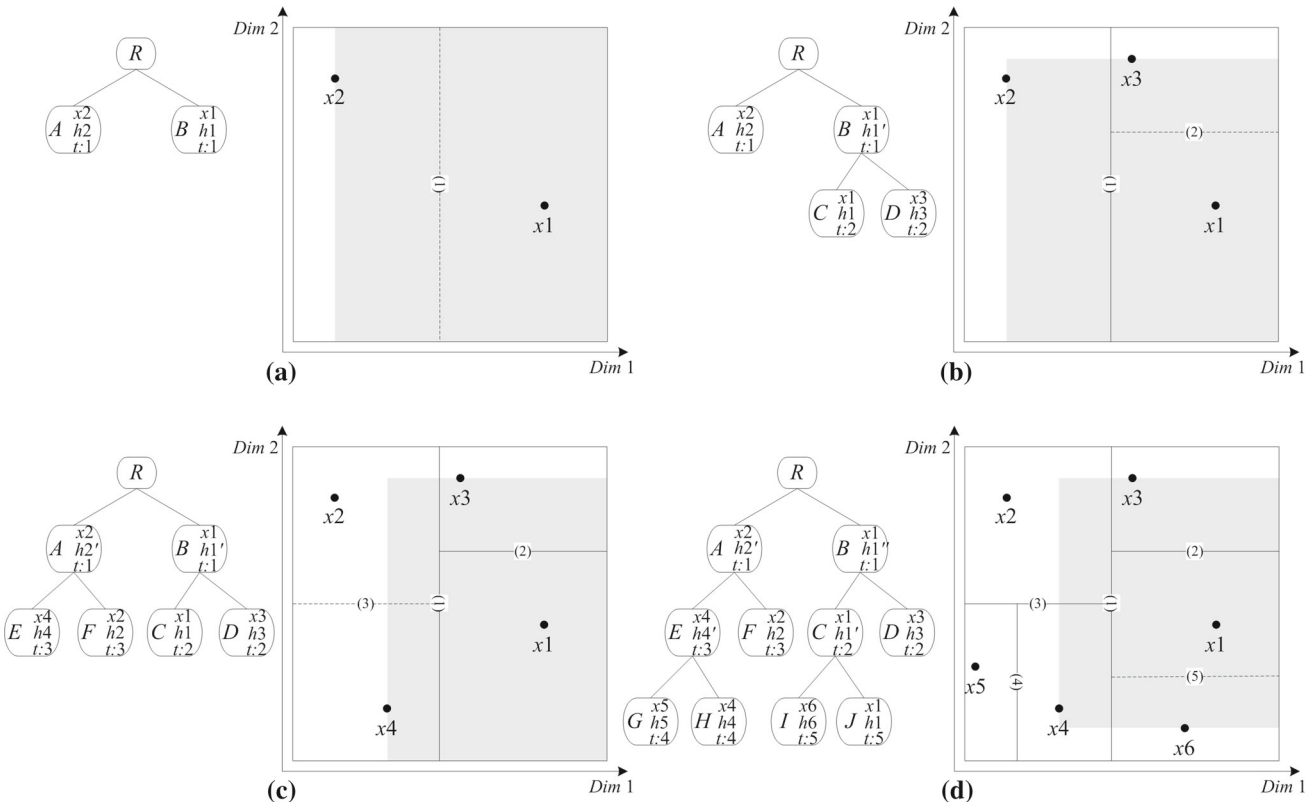


Fig. 3 The one-to-one correspondence relationship between solutions and their sub-regions

All the evaluated solutions are stored in the BSP tree. Each leaf node of the tree contains a unique evaluated solution, as well as its allocated sub-region, while all the non-leaf nodes are auxiliary for construction of the tree structure. These nodes are also known as the virtual nodes. Consider leaf node l_i in the BSP tree T. It stores in the node l_i an evaluated solution x_i (including the position in the search space, the fitness value, etc.), and its allocated sub-region h_i in S, where S represents the entire search space, and $\sum_i h_i = S$.

Figure 3 gives an illustration of generating the BSP tree archive and allocating the sub-regions in a two-dimensional search space. In Fig. 3a, the search space (in the right coordinate) is partitioned into two sub-regions h_1 and h_2, by the generation of the first two solutions x_1 and x_2. Accordingly, in the BSP tree on the left of Fig. 3a, the root (denoted by R) of the BSP tree generates two offspring, nodes A and B to store the information of solutions and sub-regions. Note that the dash line in Fig. 3 represents the latest partitioning in the search space. The latest partitioning cuts the dth dimension where $d = \arg\max |x_i(d) - x_j(d)|$ at the *decision threshold*, which in [7,9] is the mid-point of $x_i\ (d)$ and $x_j\ (d)$, while in [10], the decision threshold is $x_j\ (d)$ for h_i, and $x_i\ (d)$ for h_j, other than the mid-point, thus the allocated sub-regions overlap with each other. The overlapped sub-region for adaptive mutation is meaningful and significantly improves the performance of cNrGA. The sub-region of h_1 is shaded grey in Fig. 3a–d, and the size of h_1 decreases as the number of solutions increases. In Fig. 3b, a new solution x_3 is put in the sub-region of the original h_1. Let re-denote the original h_1 by h'_1, and it is partitioned to two parts: h_1 for x_1 and h_3 for x_3, h_1 is a subtraction of h'_1. Accordingly, two offspring nodes C and D are inserted under node B, though nodes B and C store the same solution x_1, the corresponding sub-regions are different. Note that node B actually becomes an auxiliary *virtual node* from now on. In the same way, nodes E and F are inserted under node A, partitioning the original sub-region h_2 into h_2 and h_4, as shown in Fig. 3c. Nodes G and H are inserted under node E, and I and J are inserted under C in Fig. 3d. Note that all six solutions evaluated so far could be found in the leaf nodes. Any revisit would be discarded and replaced by a mutant. For example, if before the generation of x_6, the actually 6th generated solution is a revisit of x_1, it would be discarded without evaluation. The adaptive mutation would be performed within the sub-region of x_1, i.e., the gray shaded area in Fig. 3c, and x_6 is eventually generated in the position shown in Fig. 3d. Note that the sub-region of x_1 is further subtracted after insertion of x_6. Because there is a unique solution within the sub-region of h_1, the operation of mutation ensures that the mutant would be a non-visited one.

During the running of cNrGA, the size of the BSP tree grows as the number of function evaluations increases. Assume the current number of function evaluations is n, the number of leaf nodes in the BSP tree is also n. Let denote the usage of memory to store the search archive so far as n units (counting only leaf nodes). In cNrGA the maximum usage of memory is equal to the pre-defined maximum number of evaluations $MaxFEs$. If $MaxFEs$ is not substantial, it is natural and affordable to store the entire search history using $MaxFEs$ unit of memory. Otherwise effective memory management is desirable. On the one hand, storing the entire search history prevents any re-evaluations and saves the computational resource for exploration or exploitation in unknown regions. While on the other hand, the more solutions generated and stored in the BSP tree, the smaller the sub-region allocated for each solution on average. When the $MaxFEs$ is large, the average size of sub-regions for adaptive mutation is quite small, which may lead to insufficient exploration ability for cNrGA. Therefore the motivation of proposing a cNrGA variant with constant memory is not only to save the memory used, but also to maintain a sufficient sub-region size for adaptive mutation. The size of sub-region is inversely proportional to the depth of the BSP tree, or the number of partitions in the search space. Thus if the memory usage is limited, it would effectively prevent the tree from becoming too deep. As a result, a sufficiently large mutation space can be maintained.

Full details on how the BSP tree is constructed and illustrative examples can be found in [7,9,10]. Source code of cNrGA is also available in [31].

3 Memory management strategies

For discrete (combinatorial) optimization problems, NrGA [7] prunes a sub-tree when all the possible solutions within its sub-space have been evaluated, and no more searching will be performed within it. The best solution found within this sub-space represents the information of the entire sub-space. Thus all other information could be pruned except the local best solution. However, in continuous space, it is impossible to evaluate all the possible solutions within any arbitrarily small sub-region, and therefore impossible to prune any sub-tree that contains more than one node. Therefore our strategy is to keep the usage of memory constant by trying to prune leaf node one at a time, rather than pruning a sub-tree containing more than one node. As the entire search history can be found in the leaf nodes, while the non-leaf nodes are virtual nodes for organizing the search archive, pruning leaf nodes removes search information.

The basic idea is that when the algorithm reaches the user defined memory threshold, 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 [9,10]. So the memory usage will be constant.

3.1 Two pruning mechanisms

If the memory threshold is smaller than the maximum number of function evaluations, when the memory threshold is reached, some pruning operation is necessary to maintain the memory usage constant. A new leaf node would be inserted in the BSP tree after pruning an old one, where new means that the solution is newly generated, and old means it already exists in the tree. Which old one is chosen to be pruned can be determined in different ways. In this paper, we propose two mechanisms to choose the leaf nodes to prune. They are the least recently used (LRU) pruning and the random (R) pruning:

3.1.1 The least recently used (LRU) pruning

In the previous discussion in Fig. 3, we deliberately ignore one attribute of the tree nodes, the *time stamp*, denoted by t, which is newly introduced together with the LRU pruning strategy introduced here. To record the time sequence of tree nodes, a time stamp t, recording the time when the node is inserted into the BSP tree, is attached together with the solution and stored in the archive. In Fig. 3, the time stamp for each node is attached. Note that the two siblings have the same time stamp for they are generated at the same time. Thus a leaf node l_i basically contains three attributes $\{x_i, h_i, t_i\}$, representing its solution, allocated sub-region and time stamp respectively. Then it is straightforward to identify the LRU leaf node. An LRU leaf node means that its time stamp value is the smallest (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 searching in the sub-region for the longest elapsed time.

Though the basic idea of pruning is clear and simple, there are two technical problems to be addressed. First, to identify an LRU node requires one traversal throughout the whole BSP tree, and the complexity of this process is $O(n)$, this is a high cost when the number of prune-and-insert operations is considerable, since for each LRU node, it has to traverse the whole tree to find it. Therefore a to-prune-list is proposed to address this problem. A to-prune-list is a list of leaf nodes, sorted by increasing order of time stamp values. For example, assuming the BSP tree in Fig. 3d has reached its maximum memory usage. Then before we prune a tree node and insert a new one, we need to identify which one is the LRU node to prune. In this case, a to-prune-list could be formed by traversal throughout the tree once and sort once, which is $\{D\ (t = 2)\ ,\ F\ (t = 3)\ ,\ G\ (t = 4)\ ,\ H\ (t = 4)\ ,\ I\ (t = 5)\ ,\ J\ (t = 5)\}$. The pruning operates as follows: the first element

would be pruned and deleted from the list. Note that every node in the to-prune-list must be a leaf node, thus after each pruning and insertion, it is necessary to check whether the insertion of the new generated node is operated under any node in the to-prune-list. If so, the insertion makes that node a non-leaf node and should be ruled out from the list.

If a to-prune-node has a sibling that is also a leaf node, then it is easy to perform a two-step pruning, i.e., first, deletes all the information about the to-prune-node, and second, adjusts the corresponding information as if the pruned node has never appeared. Both the to-prune-node and its leaf sibling are pruned directly. However, if the sibling is not a leaf node but a sub-tree, then the situation is complicated. As a result, we encounter the second technical problem, namely, *not* every leaf node can be pruned due to the structure of the BSP tree.

If the to-prune-node stores the same solution as its parent, i.e., the parent is a virtual node of the to-prune-node, then it is inappropriate to prune it. The virtual node may affect virtual nodes further up in the tree, and in the worst case, the ancestor virtual nodes may be traced all the way back to the root node. If so, the entire BSP tree has to be drastically reorganized, which is time consuming. As a result, only leaf nodes storing different solutions from their parents can be pruned, i.e., those leaf nodes without virtual nodes as their parents. For example, let review Fig. 3d and put it in Fig. 4. It is inappropriate to prune the leaf node F due to its virtual node A; while D can be pruned. Leaf nodes $\{G\ (t=4)\ ,\ H\ (t=4)\ ,\ I\ (t=5)\ ,\ J\ (t=5)\}$ are easy to prune as illustrated. Note that pruning nodes G and H are equivalent, because they are siblings and have the same time stamps, and so are pruning I and J. Then we need only keep the one storing a different solution from their parent. The nodes feasible to prune are shaded in gray in Fig. 4. From the viewpoint of search space partitioning shown on the right figure, the sub-region of a feasible-to-prune node could be directly merged with another sub-region to form a smoothly combined sub-region, e.g., to merge the sub-regions of x_1 and x_3, the operation is to simply remove the partition cut marked (2). However, for the sub-region of solution x_2, the merging operation is not simple, no matter whether we remove the partition cut marked (3) or the one marked (1). To save the computational resource, we simply define this kind of leaf nodes as infeasible-to-prune. As a result, considering the two technical problems above, the to-prune-list is shortened to $\{D\ (t=2)\ ,\ G\ (t=4)\ ,\ I\ (t=5)\}$ from all able-to-prune leaf nodes $\{D, G, H, I, J\}$.

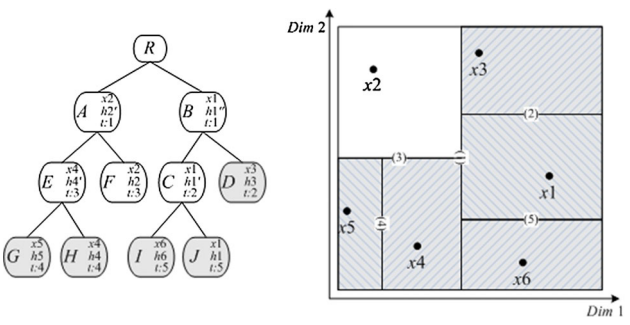


Fig. 4 Feasible or infeasible to prune

The to-prune-list is necessary to be re-formed only if whole list is exhausted. Then the entire tree is traversed again and a new list of leaf nodes is constructed.

For the sake of both efficiency and parameter-less-ness, we recommend unlimited length of the to-prune-list in the implementation. If the memory threshold for the archive is M units, the length of the to-prune-list is less than M. Note a unit of usage in an archive not only means a set of attributes including the solution, allocated sub-region and time stamp, etc., but also some virtual node(s) which enlarges the actual usage. However, for each element in the to-prune-list, there is an address pointing to the corresponding leaf node. The memory usage of the to-prune-list is negligible, since in case of a 30-dimension problem and we only store three attributes $\{x_i, h_i, t_i\}$ in a tree node, thus 31 real numbers and 1 integer is stored, including (on average) one virtual node, 62 real numbers and 2 integers are recorded as one unit of memory cost by a tree node. In contrast, an element in the to-prune-list is nothing but an integer.

Note that pruning a node in the BSP tree enlarges the sub-regions of all related tree nodes. For example in Fig. 4, if node D is pruned, then nodes I and J share the combined sub-regions that were shared by nodes D, I and J. As a result, the pruning mechanism enlarges the sub-regions for adaptive mutation.

3.1.2 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, and randomly choose the left or right child to go through, with equal probability, until it reaches a leaf node. The random pruning has less computational burden than the LRU pruning, as it only randomly finds a usable leaf node.

The second technical problem, i.e., not all nodes are able to prune as mentioned previously is also encountered for random pruning. The solution is as follows: if the leaf node reached has no virtual (parent) node, meaning that it is a feasible-to-prune node, then prune it. Otherwise, restart the random going down process starting from its sibling until it finds a feasible leaf node. For example, if any leaf node of $\{G, H, I, J, D\}$ is reached as shown in Fig. 4, then it can be pruned right away. However, if leaf node F is reached, then the random search goes down the sub-tree with its sibling node E as its head. This process will always terminate

Input: 1) BSP Tree T whose size reaches the memory threshold, 2) To-prune-list L, 3) New solution s, 4) Pruning Method P

```
If L is empty
    Traverse the entire T and get L      // L is unsorted at this point
    If P = 'LRU'
        Sort L by the time stamp attached in each node
    End
Else
    If P = 'LRU'
        k := 1                           // The first element in the sorted L is the LRU node
    Else                                 // The pruning method is P = 'R'
        k := rand(1, Length of L)        // Pick a random element from the unsorted L
    End
    node := L(k)                         // Get an element from L
    Delete L(k) from L
    Re-organize T by pruning node
    node := s
    Insert node in T                     // Some node in L may become non-leaf node after insertion
    Check and delete non-leaf nodes in L
End
```

Output: 1) The updated BSP Tree T, 2) The updated to-prune-list L

Fig. 5 Pseudo code of the two pruning mechanisms

because the BSP tree is constructed such that a virtual parent node has at least one real child node.

The to-prune-list for R pruning is not essential but is recommended. It is the same as that for LRU pruning but without sorting. In LRU to-prune-list, the first element is always fetched and used, while in R to-prune-list a random element is picked. Note that when the to-prune-list is applied to R pruning, then (1) it is not exactly the same as picking a random leaf node each time, because the recently inserted nodes are not included in the list; they only avail themselves for random pruning when the to-prune-list is exhausted and regenerated, and (2) the second technical problem is naturally avoided, because those unsuitable-to-prune nodes are skipped and will not be included in the list.

Figure 5 gives the pseudo code of two proposed pruning mechanisms, where we employ the to-prune-list for both LRU and R pruning.

3.2 Novel parameter-less adaptive mutation operators

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

Both pruning strategies can be regarded as enlarging the mutation range (i.e., the sub-region size) for the adaptive mutation operator, which is a key operator in cNrGA. As both pruning operators do not introduce any additional parameters, the operators are parameter-less. However, they have different properties:

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. The sizes of 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 uniformly on average the mutation range, while keeping the same mutation strategies as in the original cNrGA, which is: the leaf nodes with deeper depths are mutated less, meaning more exploitation, and vice versa.

4 Experimental studies

In this section, we examine the performance of cNrGA/CM/LRU and cNrGA/CM/R by comparing it to the original version of cNrGA [7], as well as RGA [26–28] and SPSO 2011 [21]. The key issue is to find out if the pruning mechanisms affect the performance of cNrGA significantly, and investigate whether pruning brings better mutation for the optimization. All algorithms are implemented in Matlab. The experiments are performed on a PC with a 3.40 GHz quad core CPU with 4.00 GB memory. The source code of cNrGA/CM/LRU and cNrGA/CM/R can be found in [31].

The rest of the section is organized as follows. The parameters settings and analysis of experimental results are presented in Sects. 4.1 and 4.2. Followed by Sect. 4.3, which

Table 1 Loss of history information in cNrGA/CM/LRU and cNrGA/CM/R

MaxFEs	1E+04	2E+04	3E+04	4E+04	5E+04	6E+04	7E+04	8E+04	9E+04	1E+05
MT	1E+04	1E+04	1E+04	1E+04	1E+04	1E+04	1E+04	1E+04	1E+04	1E+04
Information loss	0	0.50	0.67	0.75	0.80	0.83	0.86	0.88	0.89	0.90

shows both pruning strategies do not affect the effectiveness of mutation in cNrGA. Both theoretical and experimental running time are discussed in Sect. 4.4. At the end of this section, the utility of the to-prune-list is analyzed, empirically and theoretically.

4.1 Test settings

The full set of 28 benchmark functions in CEC 2013 test suite [32] is employed. The dimension of the functions is $D = 30$. The maximum number of function evaluations ($MaxFEs$) is chosen from 10,000 to 100,000. The memory threshold (MT) is set to be 10,000 units for cNrGA/CM/LRU and cNrGA/CM/R. Since MT is smaller than $MaxFEs$, some portion of history information must be pruned. Table 1 shows the loss of search information with each setting. For cNrGA, the setting is $MT = MaxFEs$ so that no search information is lost.

To evaluate the effect of the pruning fairly, for cNrGA, cNrGA/CM/LRU, and cNrGA/CM/R, we use the same parameters suggested in [10]. Thus the population size is set to 100 and the crossover rate is set to 0.5. In each run, to be fair, the three algorithms share the identical random process outcomes 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 trees, then inserts the new node representing the new solution into their own trees. 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 behaviors after that.

4.2 Results

4.2.1 Significance tests of the three algorithms

The detailed experimental results can be found in Appendix. The experimental results in Tables 5, 6, 7 and 8 in "Appendix" show that neither LRU nor R pruning strategy degrades the overall performance of cNrGA. The performance of cNrGA with R pruning varies slightly more than cNrGA with LRU pruning for different problems.

The performance of an algorithm is measured by statistics from 30 independent runs. Tables 5, 6, 7 and 8 in "Appendix" show the means, standard deviations, and the p values of Mann-Whitney U test (with significance level $\alpha = 0.05$) obtained by the three variants of cNrGA when the percentage of information lost varies from 0 to 90 % uniformly (refer to Table 1). The best mean results amongst the three algorithms for each problem are shaded in gray.

In total, 280 (28 benchmark functions $\times$ 10 different values of $MaxFEs$) sets of data are obtained from the experiments, where there are 252 comparisons and 28 for reference (the set of $MaxFEs = MT$). Among the 252 comparisons, 244 are found to have no significant difference, and the remaining eight cases are not one-sided.

cNrGA significantly outperforms cNrGA/CM/LRU 2 times (f1 with $MaxFEs = 40{,}000$ and 80,000), and significantly outperforms cNrGA/CM/R four times (f1 with $MaxFEs = 40{,}000$, f5 with $MaxFEs = 80{,}000$, f17 with $MaxFEs = 60{,}000$, and f23 with $MaxFEs = 100{,}000$). These data is in bold and marked † in Tables 5, 6, 7 and 8 in "Appendix". In contrast, cNrGA/CM/LRU significantly outperforms cNrGA 2 times (f5 and f28 with $MaxFEs = 100{,}000$), while cNrGA/CM/R does not significantly outperforms cNrGA in all cases. The data where cNrGA/CM/LRU significantly outperforms cNrGA is in bold and is marked ‡ in Tables 5, 6, 7 and 8 in "Appendix". Significant differences only occur in solving the above five functions (f1, f5, f17, f23 and f28), while for solving the other 23 functions, f2–4, f6–16, f18–22, and f24–27, no algorithm is found to be significantly different from the other two.

4.2.2 Comparison with RGA and SPSO 2011

We also compare these three versions of cNrGA with real-coded genetic algorithm (RGA) [26–28] and the standard particle swarm optimization (SPSO) 2011 [21]. RGA is selected as it is a GA dealing with continuous optimization and has a set of designer supplied fixed parameters, while cNrGA and its variants have only two parameters and the rest of the process is guided by history and the non-revisiting mechanism. SPSO 2011 is selected as PSO is an influential paradigm and SPSO 2011 is its latest standard version.

The parameters of SPSO are set to be the same as in [21]. The parameters of RGA are set as follows: BLX-α crossover [26] with α 0.4 and crossover rate 0.7, mutation rate 0.3,

tournament selection with tournament size 3. $MaxFEs$ is set 100,000 for all five algorithms, while the memory usage threshold is 10,000 for the two versions of cNrGA with constant memory.

Table 9 in "Appendix" shows the mean results and standard deviation. Table 10 in "Appendix" presents the result of Mann-Whitney U test (with significance level $\alpha = 0.05$). Problems f1–5 are unimodal, f6–20 are basic multimodal, and f21–28 are composition problems. Both cNrGA and cNrGA/CM/LRU outperform RGA significantly in 22 cases, while cNrGA/CM/R outperform RGA significantly in 20 cases. Both cNrGA/CM/LRU and cNrGA/CM/R outperform SPSO significantly in 12 cases, while cNrGA outperform SPSO significantly in 13 cases. In contrast, RGA only outperforms cNrGA/CM/R significantly in two cases, and SPSO outperforms the three variants of cNrGA significantly nine cases for each. It can be seen from the tables that all three versions of cNrGA are substantially better than RGA and slightly better than SPSO 2011. Qualitatively, all three versions of cNrGA are superior to SPSO 2011 in the multimodal and composition problems but are inferior in the unimodal problems.

4.3 Adaptive mutation

cNrGA employs the adaptive one-gene-flip mutation. On the one hand, uniform crossover is applied for coarse tuning of the suggested searching region (exploration oriented), while mutation is used for fine tuning (exploitation oriented). However, on the other hand, mutation brings new gene element to the population, which enlarges the diversity of population. Since parameter control is a difficult issue [30,33], one-gene-flip mutation is designed to be parameter-less. The mutation step size depends on the size of the corresponding sub-region. As a result, the step size of mutation adaptively decreases as the search moves on. The merit of this strategy is that it automatically moves the search emphasis from initially exploration prone to later on exploitation prone. Yet the demerit is, the more solutions it accumulates, the smaller sub-region would be allocated for each solution on average. A tiny sub-region is good only when it is exploiting around the actual global optimum, which is usually unknown.

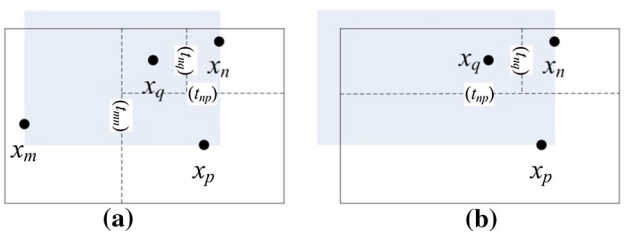


Fig. 6 An example of sub-region enlargement after pruning

Figure 6 gives an example in two dimensional search space. There are four solutions in the subspace, dividing the subspace into four sub-regions in Fig. 6a. Suppose the memory threshold has been reached, and time stamp t_{mn} is the smallest overall, implying the sub-region around the solution x_m is rarely visited. Then, applying the LRU pruning, the search information about x_m would be removed, so that the sub-regions of its neighbors, i.e., x_q and x_p, would be enlarged. In Fig. 6a, the gray shaded region in sub-region for adaptive mutation if any new solution is identical to solution x_q, while in Fig. 6b, the gray region is enlarged by pruning. Both LRU and R pruning maintain a certain sub-region size, and the average size for each sub-region is the size of the search space divided by MT. Note that the average sub-region size of cNrGA is the size of the search space divided by $MaxFEs$. LRU pruning forms a heterogeneous distribution for the sizes of sub-region, i.e., for the frequently visited areas, the solutions are probably kept. Thus the sub-regions are small in these areas. In contrast, for the rarely visited area, the solutions are prone to be pruned, and thus the sub-regions are prone to be sparse. In contrast, R pruning forms a homogeneous distribution for the sub-region size.

To test the effect on the mutation operation caused by LRU and R pruning, we analyze two indexes, the *number of mutation*, and the *number of effective mutation* when running the algorithms. We define a mutation to be effective if the fitness value of mutant is improved compared with its parent. In cNrGA, the elitism selection is employed, while effective mutation is more likely to tournament selection. The concept of effective mutation is only presented to reflect the effectiveness of adaptive mutation.

Because the entire history is recorded in cNrGA, while only a portion of the entire history is kept in cNrGA with constant memory, it is more likely that one newly generated solution will be a revisit of the former than in the later. In the case that the generation of offspring is totally random, then it is true. However, there are two other factors that need to be considered. The first factor is selection, which makes the population concentrates more on certain sub-regions, within which revisits are prone to happen. The second factor is that both pruning strategies will tend to keep more recent information. For LRU pruning, as its name suggests, all the recently used data is kept in the BSP tree. In R pruning, because of the to-prune-list, the recently generated leaf nodes are not included in the to-prune-list; they only avail themselves for random pruning when the list is exhausted and regenerated. Thus the new nodes have no chance to be pruned for some time. As a result, the current implementation of R pruning also has more tendency to delete information which is old or less used.

Considering the above two factors, cNrGA with both types of prunings preserve the nature of the original cNrGA of concentrating the search on recently used sub-regions. Therefore

Table 2 The ratio of the number of mutations and the number of effective mutations

	MaxFEs	f1	f2	f10	f18	f25	f28
(a) Ratio of the number of mutation							
cNrGA vs. cNrGA/CM/LRU	20,000	1.01	0.98	0.97	1.00	1.07	1.01
	50,000	0.99	0.99	0.98	0.99	1.03	1.01
	80,000	0.97	0.98	0.97	0.97	1.00	0.99
	100,000	0.97	0.99	1.02	0.97	0.97	0.96
cNrGA vs. cNrGA/CM/R	20,000	1.00	0.98	0.72	0.97	1.03	1.02
	50,000	0.97	0.97	0.93	0.99	1.01	1.00
	80,000	0.93	0.97	0.95	0.97	1.00	0.99
	100,000	0.95	0.97	0.97	0.94	0.98	0.99
(b) Ratio of the number of effective mutation							
cNrGA vs. cNrGA/CM/LRU	20,000	0.94	1.00	1.03	0.96	0.99	1.00
	50,000	1.00	0.97	1.03	0.99	1.01	1.01
	80,000	**0.83**	0.99	1.01	**0.60**	0.95	1.02
	100,000	**0.51**	1.00	1.00	**0.45**	0.97	1.02
cNrGA vs. cNrGA/CM/R	20,000	0.94	1.01	0.82	1.00	1.00	1.01
	50,000	1.03	0.98	0.98	1.03	1.00	0.99
	80,000	**0.80**	1.01	1.02	**0.62**	0.98	1.06
	100,000	**0.49**	0.99	1.04	**0.46**	0.97	1.01

the frequency of revisits in cNrGA with pruning would not be affected drastically. This observation is verified by the experimental results in Table 2a and b, which confirm that the revisit frequency is only slightly affected by pruning.

We randomly choose six functions with four different $MaxFEs$ settings to test the influence on mutation caused by pruning. In Table 2a, there are the ratios of the number of mutations. The ratio is calculated by the average number of mutations of cNrGA divided by the average number of mutations of either version with pruning. A value greater than one means that there is greater number of mutation in cNrGA and vice versa. For example for f1 when $MaxFEs$ is 20,000, after it reaches MT, the average number of mutations is 3651.17 for cNrGA and 3623.23 for cNrGA/CM/LRU respectively. Thus the first data cell shown in Table 2a is $3651.17/3623.23 = 1.01$.

Note that before and when the MT is reached, the three algorithms share the identical evolution progress in this experiment, for the sake of fairness. All the data are averaged from 30 independent runs.

Table 2b shows the ratios of the number of effective mutations, where eight cells (bold region) displaying numbers distinctly smaller than one can be found. Thus for f1 and f18, when the $MaxFEs$ are 80,000 and 100,000, the pruning operations slightly increase the numbers of mutation, which could be considered as noise, as shown in Table 2a. However, the numbers of effective mutations are significantly increased by both LRU and R pruning as shown in Table 2b. Especially, when $MaxFEs$ is 100,000, the number of effective mutation are doubled, compared with the original cNrGA. Thus although cNrGA/CM/LRU or cNrGA/CM/R have approximately the same number of mutations as cNrGA (as shown in Table 2a), there are twice the number of mutations in both cNrGA/CM/LRU and cNrGA/CM/R that improve the fitness in some cases (as shown in those bold in Table 2a). In other words, the mutation operation in the two pruning versions of cNrGA is twice as effective in solving some problems, while in solving other problems, the proportion of effective mutation is not significantly affected. This means the performance of mutation in the original cNrGA is at least maintained when pruning strategies are employed. Surprisingly, in a few cases, the mutation performance is significantly enhanced by the pruning.

To test the statistical significance of the above observations, the Mann-Whitney U test (with significance level $\alpha = 0.05$) is performed, and the eight data cells which are in bold in Table 2b, meaning that the effect of in the increase in the number of effective mutation is statistically significant. Excluding these eight data, the other elements can be considered as ratio 1.0 but slightly affected by a small random noise, since the difference is insignificant.

Although the definition of effective mutation is irrelevant to the elitism selection in these algorithms, the increase of the number of effective mutation supports the thesis that both LRU and R pruning operations can be interpreted as novel parameter-less adaptive mutation operators.

4.4 Running time

Clearly both LRU and R pruning strategies reduce the usage of memory; next let examine computational overhead reflected by time consumption. cNrGA makes a trade-off between the performance and time and memory overhead. A $MaxFEs$ of 40,000 is used in [7,9,10]. In this paper, we maintain a constant usage of memory in cNrGA/CM/LRU and cNrGA/CM/R. The overhead of memory is fixed. Thus the only other concern in cNrGA with constant memory is the time consumption. Next, we first analyze the theoretical time consumption of three algorithms, followed by experimental results.

For all cNrGAs with or without pruning strategy, the time consumption could be represented as follows,

$$T_{total} = T_{evo} + T_{eval} + T_{BSPtree} \tag{1}$$

where, T_{evo} is the cost of the genetic evolution, including initialization, crossover, mutation and selection. is the cost of evaluations. is equal for all the algorithms if they are performed with the same $MaxFEs$. $T_{BSPtree}$ is the time cost for BSP tree operations, including searching position to prune (insert), pruning (insertion), etc. Assuming M units of memory are available ($MT = M$) and the required number of evaluations is N ($MaxFEs = N$). The theoretical time consumption on the BSP tree for the three algorithms is as follows.

$$T_{BSPtree}^{(cNrGA)} = T_{exp}(N) \tag{2}$$

$$T_{BSPtree}^{(R)} = T_{exp}(M) + T_{p\&i}(N - M) + T_{ov} \tag{3}$$

$$T_{BSPtree}^{(LRU)} = T_{exp}(M) + T_{p\&i}(N - M) + T_{ov} + T_{sort} \tag{4}$$

where T_{exp} is the time cost of the BSP tree expansion, $T_{p\&i}$ is the time cost of pruning and insertion, T_{ov} is the overhead introduced by pruning. Because both pruning methods employ the to-prune-list, their costs are the same except for one term T_{sort}. It is the cost to keep the to-prune-list of LRU pruning in order, composed of two parts. The first part is the sorting of the to-prune-list immediately after its generation, and the second part is after each pruning and insertion, the up-prune-list is updated so that some unable-to-prune node (if any) is excluded from the list. Both parts of T_{sort} keep the to-prune-list sorted by their time stamps and make sure that all nodes in the list are able to be pruned.

Note that for a BSP tree with k leaf nodes, the total number of tree nodes are $2k - 1$, including leaf nodes and non-leaf (virtual) nodes. T_{exp} and $T_{p\&i}$ are defined as follows, assuming the generated BSP tree is balanced for simplicity:

$$T_{exp}(k) = \sum_{i=1}^{2k-1} (T_{vis} \cdot \log_2 i) + (2k - 1) \cdot T_{ins} \tag{5}$$

$$T_{p\&i}(k) = 2k \cdot (T_{pru} + T_{ins}) \tag{6}$$

where T_{vis}, T_{ins}, and T_{pru} are the time costs of *visiting*, *inserting*, and *pruning* a node respectively. The coefficient $2k$ in Eq. (6) means that each pruning and inserting is related to two accompanied offspring, though each time only one solution is pruned and inserted.

Comparing Eqs. (3) and (4), R pruning is always faster than LRU pruning.

Table 3 shows the experimental running time of functions f1, f12, f20 and f28, randomly selected from the 28 problems in the CEC 2013 test suite. As can be seen from the table, the original cNrGA runs the fastest. The running time of cNrGA/CM/LRU and cNrGA/CM/R are between two and three times greater than that of cNrGA.

Table 3 Comparison of running time excluding the initial 10,000 evaluations of cNrGA, cNrGA/CM/LRU, and cNrGA/CM/R (unit: seconds)

MaxFEs	f1			f12			f20			f28		
	cNrGA	cNrGA/CM/LRU	cNrGA/CM/R	cNrGA	cNrGA/CM/LRU	cNrGA/CM/R	cNrGA	cNrGA/CM/LRU	cNrGA/CM/R	cNrGA	cNrGA/CM/LRU	cNrGA/CM/R
2E+04	0.51	1.44	1.37	0.51	1.44	1.37	0.47	1.40	1.32	0.50	1.44	1.36
3E+04	1.04	2.75	2.63	1.03	2.78	2.66	0.95	2.74	2.57	1.02	2.76	2.64
4E+04	1.61	4.15	3.96	1.59	4.17	3.98	1.46	4.10	3.89	1.55	4.13	3.93
5E+04	2.21	5.55	5.32	2.15	5.53	5.29	1.98	5.46	5.18	2.10	5.49	5.26
6E+04	2.80	6.96	6.67	2.69	6.87	6.58	2.50	6.84	6.48	2.68	6.89	6.59
7E+04	3.52	8.44	8.11	3.26	8.25	7.89	3.04	8.21	7.80	3.18	8.18	7.84
8E+04	4.07	9.84	9.46	3.78	9.61	9.19	3.58	9.56	9.09	3.79	9.59	9.19
9E+04	4.82	11.31	10.94	4.38	10.99	10.52	4.15	10.94	10.42	4.34	11.02	10.55
1E+05	5.59	12.86	12.35	4.92	12.40	11.85	4.72	12.33	11.74	4.97	12.38	11.87
2E+05	13.73	27.48	26.45	10.36	25.32	24.36	10.10	25.15	24.15	10.64	25.48	24.40
3E+05	23.24	43.21	41.31	16.06	38.83	37.23	15.76	38.52	37.20	16.98	39.31	37.58

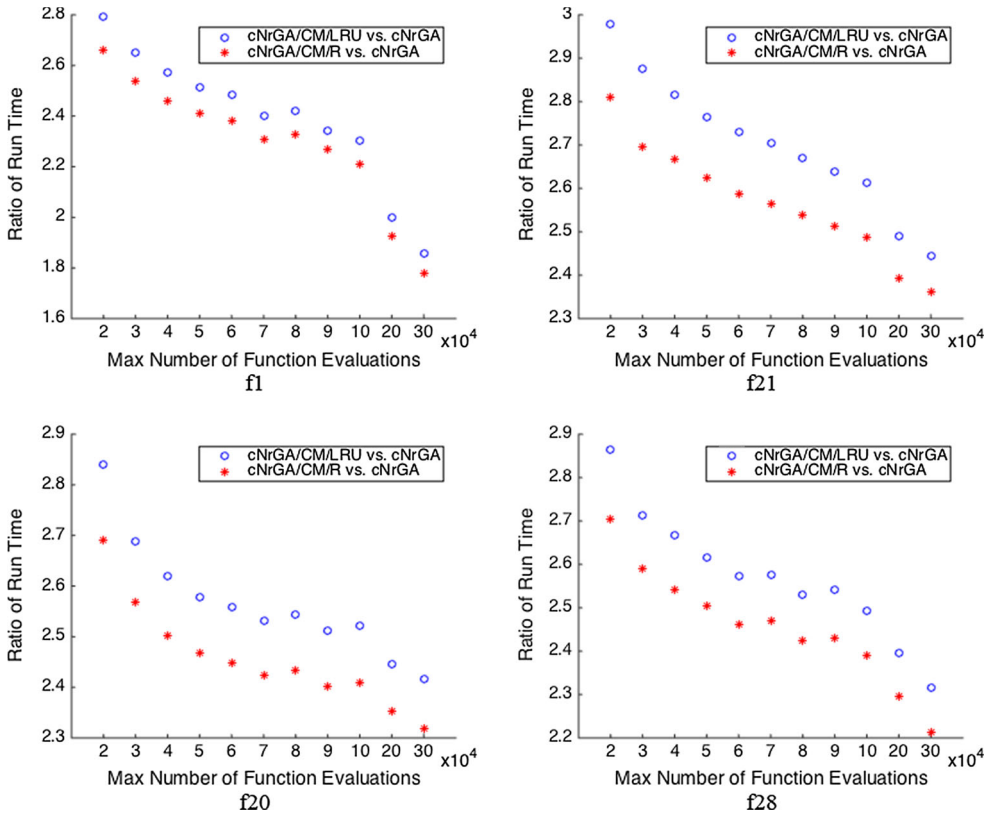


Fig. 7 Ratio of run time vs. maximum number of function evaluations ($MaxFEs$)

We further examine the algorithm running time extending the $MaxFEs$ from 100,000 to 300,000, and find that the ratio of time consumption of cNrGA with constant memory versus cNrGA is further reduced when the $MaxFEs$ is large. The detailed ratio of time consumption is plotted in Fig. 7, which shows (1) the ratio of cNrGA/CM/R vs. cNrGA is always less than cNrGA/CM/LRU vs. cNrGA. This means that cNrGA/CM/R is always faster than cNrGA/CM/LRU, (2) the tendency of the curves is generally decreasing, which means that both cNrGA/CM/LRU and cNrGA/CM/R are becoming more time efficient when the required number of evaluations is increased, and (3) all the ratios are greater than one, and between two and three, which means cNrGA/CM/LRU and cNrGA/CM/R increase the computational time two or three times, but could save nine times memory usage (by pruning 90 % of history information) without reducing the performance significantly.

4.5 The to-prune-list

The to-prune-list is proposed to address the technical problem that to identify one LRU node each time is computationally expensive. It is essential for LRU pruning. It is also recommended for R pruning, since it is empirically fast when the to-prune-list is applied for R pruning, though it changes somewhat the original meaning of R pruning. The following compares the computational cost of the pruning strategies with and without employing the to-prune-lists.

There are in total $(N - M)$ times pruning and inserting needed, to maintain the memory usage of the BSP tree to a constant M. Note that it is required to traverse all the $(2M - 1)$ nodes of the BSP tree to identify the LRU node, while for R pruning, it only requires $\log_2(2M - 1)$ to reach a leaf node randomly. Assuming all leaf nodes can be pruned, the complexities of searching random and LRU leaf nodes are given in Eqs. (7) and (8). Let τ be the number of times of (re-)generating to-prune-lists, then the complexities of using the to-prune-lists is given in Eqs. (9) and (10).

$$C_{rnd} = (N - M) \cdot \log_2(2M - 1) \tag{7}$$

$$C_{lru} = (N - M) \cdot (2M - 1) \tag{8}$$

$$C_{rnd}^{tpl} = \sum_{i=1}^{\tau} (2M - 1) = \tau \cdot (2M - 1) \tag{9}$$

$$C_{lru}^{tpl} = \sum_{i=1}^{\tau} (2M - 1 + C_{sort}(L_i)) = \tau \cdot (2M - 1) + \sum_{i=1}^{\tau} C_{sort}(L_i) \tag{10}$$

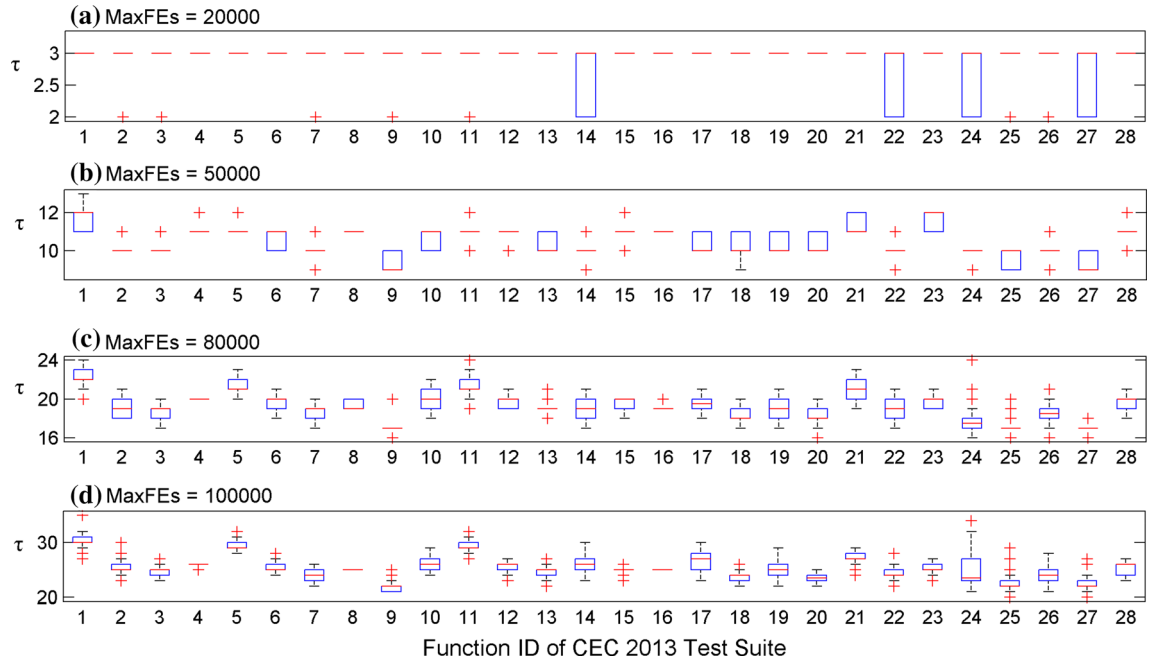


Fig. 8 Boxplots of experimental values of τ when the $MaxFEs$ is **a** 20,000, **b** 50,000, **c** 80,000, and **d** 100,000 respectively

where C_{rnd} is the complexity of randomly search leaf nodes on the BSP tree with M leaf nodes ($2M-1$ nodes in total) one after the other. C_{lru} is the complexity of searching LRU nodes one after the other. The superscript *tpl* in Eqs. (9) and (10) means using the to-prune-list. $C_{sort}(L_i)$ is the complexity of sorting a to-prune-list of length L_i, then $C_{sort} = O(L_i \cdot \log_2 L_i)$ [34,35]. The exact complexity C_{sort} depends on the sorting method employed.

Because τ and L_i are unknown in Eqs. (9) and (10), we study them experimentally. Figure 8 shows four boxplots, from up to down, when the $MaxFEs$ are (a) 20,000, (b) 50,000, (c) 80,000, and (d) 100,000 respectively. The blue box denotes that the central 50 % data lies within this section; the red bar is the median value of all 30 datasets; the upper and lower black bars are the greatest and the least values, excluding outliers; and finally the red pluses represent the outliers. $\bar{\tau}$, the average number of (re-)generating the to-prune-list, and $\bar{L}$, the average length of the to-prune-lists, for different values of $MaxFEs$ is shown in Table 4, accompanied by the standard deviations of τ and L_i of 30 independent runs.

In the case that the memory threshold is $M = 10,000$, and the maximum number of evaluations $MaxFEs = 100,000$, $\bar{\tau} = 25.39$ and $\bar{L} = 3963.31$. Approximately, let substitute $\tau = \bar{\tau}$, and $L_i = \bar{L}$ into Eqs. (9) and (10). According to Table 4, both the standard deviations of τ and L_i are relatively small, thus the sorting complexity can be approximated as Eq. (11). Practically in implementation, we use the Matlab function *sort* with default mode, i.e., 'ascend' to sort the to-prune-list. The algorithm employed in *sort* function is quicksort, and thus its time complexity is $C_{sort} = O\left(n \cdot \log_2 n\right) \approx 1.39 \cdot n \cdot \log_2 n$, where n is the number of numerical elements to be sorted [34,35].

Table 4 The average number and standard deviation of the times of (re-)generating the to-prune-list, and the average length and standard deviation of the to-prune-lists, when the $MaxFEs$ is 20,000, 50,000, 80,000, and 100,000 respectively

MaxFEs	2E+04	5E+04	8E+04	1E+05
$\bar{\tau}$	2.91	10.50	19.24	25.39
SD	0.15	0.41	0.81	1.18
$\bar{L}$	4852.97	4335.71	4082.26	3963.31
SD	94.21	111.22	140.31	165.76

$$\sum_{i=1}^{\tau} C_{sort}(L_i) = 1.39 \cdot \sum_{i=1}^{\tau} L_i \cdot \log_2 L_i \approx 1.39 \cdot \bar{\tau} \cdot \bar{L} \cdot \log_2 \bar{L} \tag{11}$$

Using Eqs. (7) and (9), we obtain $C_{rnd} = 1.29 \times 10^6$ and $C_{rnd}^{tpl} = 0.51 \times 10^6$. The usage of the to-prune-list on average reduces the time complexity by approximately twice in R pruning. As for the LRU pruning, using Eqs. (8) and (10), $C_{lru} = 1.80 \times 10^9$ and $C_{lru}^{tpl} = 2.18 \times 10^6$. The reduction of time complexity is approximately 800 times.

5 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.

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 when it reaches a predefined memory threshold. Thus in principle, cNrGA can be used for all kinds of search problems, especially for 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. Further, the to-prune-list is proposed which makes the pruning more efficient.

The experimental results reveal that these pruning strategies do not degrade the performance of cNrGA. Also they do not significantly reduce the number of mutation that is important for obtaining good solutions. With up to 90 % pruning, the performance of the two cNrGA variants and the original cNrGA does not show any significant difference. They also show that the pruning strategies may be legitimately considered as novel parameter-less adaptive mutation operators, and somewhat surprisingly, they even lead to better mutation conditions for some problems we tested. Moreover, the computational overhead of the proposed pruning strategies is reasonably small when the to-prune-list is used.

A neck to neck comparison of the three versions of cNrGA is also made with real-coded genetic algorithm (RGA) and standard particle swarm optimization (SPSO) 2011. Considering the number of significantly superior and inferior cases, it is found that all three versions of cNrGA are substantially better than RGA and slightly better than SPSO 2011. Qualitatively, all three versions of cNrGA are superior to SPSO 2011 in the multimodal and composition problems but are inferior in the unimodal problems.

From the above results, 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.

Acknowledgments 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 suggesting that pruning can be done randomly on the discrete version of NrGA.

Appendix

See Tables 5, 6, 7 and 8.

Table 5 Performance comparison of cNrGA, cNrGA/CM/LRU and cNrGA/CM/R. The best mean fitness values over three algorithms are shaded in grey. † means cNrGA is significantly superior to its variant with constant memory, while ‡ means cNrGA is significantly inferior

		f1	f2	f3	f4	f5	f6	f7
1E+04	**cNrGA**	-1229.92	6.36E+07	1.16E+10	1.09E+05	-861.13	-790.60	-694.73
	SD	56.63	2.26E+07	5.32E+09	1.57E+04	53.41	33.62	22.85
	cNrGA/CM/LRU	-1229.92	6.36E+07	1.16E+10	1.09E+05	-861.13	-790.60	-694.73
	SD	56.63	2.26E+07	5.32E+09	1.57E+04	53.41	33.62	22.85
	cNrGA/CM/R	-1229.92	6.36E+07	1.16E+10	1.09E+05	-861.13	-790.60	-694.73
	SD	56.63	2.26E+07	5.32E+09	1.57E+04	53.41	33.62	22.85
2E+04	**cNrGA**	-1392.74	3.91E+07	3.82E+09	9.70E+04	-988.35	-826.21	-715.25
	SD	2.90	1.76E+07	1.78E+09	1.44E+04	7.93	26.92	15.90
	cNrGA/CM/LRU	-1390.97	3.92E+07	3.79E+09	9.70E+04	-990.19	-827.67	-716.51
	SD	4.34	1.76E+07	1.98E+09	1.44E+04	3.79	27.48	17.92
	p-value	(0.0963)	(0.9823)	(0.9234)	(1.0000)	(0.9000)	(0.8073)	(0.8073)
	cNrGA/CM/R	-1389.63	3.89E+07	3.86E+09	9.69E+04	-989.23	-825.86	-715.65
	SD	12.58	1.70E+07	1.96E+09	1.44E+04	6.22	27.58	17.87
	p-value	(0.3403)	(0.9470)	(0.7283)	(1.0000)	(0.8883)	(1.0000)	(0.9470)
3E+04	**cNrGA**	-1398.04	2.78E+07	2.84E+09	9.14E+04	-998.30	-826.30	-718.62
	SD	1.67	1.21E+07	2.10E+09	1.24E+04	0.75	21.89	15.85
	cNrGA/CM/LRU	-1397.60	2.79E+07	2.94E+09	9.05E+04	-998.25	-825.56	-719.73
	SD	3.50	1.31E+07	2.42E+09	1.25E+04	0.80	20.60	13.28
	p-value	(0.7283)	(0.9352)	(0.9587)	(0.6734)	(0.7845)	(0.9941)	(0.8766)
	cNrGA/CM/R	-1397.98	2.77E+07	2.96E+09	9.01E+04	-997.81	-825.78	-717.44
	SD	1.72	1.19E+07	2.38E+09	1.59E+04	1.83	21.24	15.13
	p-value	(0.9470)	(0.9000)	(1.0000)	(0.5010)	(0.4204)	(0.8418)	(0.8187)
4E+04	**cNrGA**	-1399.66	3.09E+07	1.69E+09	8.30E+04	-999.24	-841.65	-720.23
	SD	0.42	1.05E+07	1.39E+09	1.14E+04	0.61	25.73	15.87
	cNrGA/CM/LRU	-1399.38	3.05E+07	1.81E+09	8.18E+04	-999.13	-839.76	-719.13
	SD	0.70	1.04E+07	1.58E+09	1.19E+04	0.64	26.34	15.70
	p-value	**(0.0026)**†	(0.9587)	(0.7394)	(0.6897)	(0.4376)	(0.7845)	(0.6952)
	cNrGA/CM/R	-1399.31	3.12E+07	1.79E+09	8.11E+04	-998.94	-839.56	-720.54
	SD	1.22	1.05E+07	1.71E+09	1.23E+04	1.72	26.19	13.35
	p-value	**(0.0040)**†	(0.7845)	(0.8418)	(0.5394)	(0.8650)	(0.7845)	(0.9470)
5E+04	**cNrGA**	-1399.77	2.69E+07	1.62E+09	7.05E+04	-999.54	-838.46	-721.80
	SD	0.24	1.33E+07	1.25E+09	1.15E+04	0.89	22.76	16.96
	cNrGA/CM/LRU	-1399.84	2.76E+07	1.75E+09	7.14E+04	-999.67	-837.67	-719.89
	SD	0.15	1.32E+07	1.32E+09	1.10E+04	0.21	23.83	17.80
	p-value	(0.2170)	(0.8303)	(0.7506)	(0.7061)	(0.8303)	(0.9234)	(0.6520)
	cNrGA/CM/R	-1399.77	2.68E+07	1.62E+09	7.06E+04	-999.75	-839.00	-722.18
	SD	0.24	1.24E+07	1.08E+09	1.09E+04	0.24	22.67	16.42
	p-value	(0.9470)	(0.9823)	(0.7506)	(0.9352)	(0.0877)	(0.8303)	(0.9941)
6E+04	**cNrGA**	-1399.68	2.12E+07	1.03E+09	6.67E+04	-999.87	-848.01	-724.35
	SD	1.43	1.05E+07	9.02E+08	1.11E+04	0.08	28.98	18.56
	cNrGA/CM/LRU	-1399.93	2.18E+07	1.27E+09	6.36E+04	-999.79	-847.56	-723.88
	SD	0.07	1.04E+07	1.17E+09	1.36E+04	0.24	28.67	17.98
	p-value	(0.9587)	(0.7283)	(0.8418)	(0.4419)	(0.5592)	(0.9941)	(0.9705)
	cNrGA/CM/R	-1399.92	2.18E+07	1.05E+09	6.56E+04	-999.80	-846.64	-723.76
	SD	0.05	1.02E+07	7.19E+08	1.10E+04	0.15	30.15	17.71
	p-value	(0.0993)	(0.7506)	(0.6843)	(0.7282)	(0.1537)	(0.9000)	(0.9352)
7E+04	**cNrGA**	-1399.84	2.02E+07	1.31E+09	5.85E+04	-999.69	-841.75	-725.25
	SD	0.61	9.96E+06	1.34E+09	1.32E+04	0.89	24.98	13.53
	cNrGA/CM/LRU	-1399.93	2.03E+07	1.18E+09	6.01E+04	-999.73	-840.89	-726.63
	SD	0.10	1.06E+07	1.16E+09	1.32E+04	0.82	24.55	14.30
	p-value	(0.5692)	(0.9352)	(0.7618)	(0.5493)	(0.3403)	(0.9352)	(0.8187)
	cNrGA/CM/R	-1399.47	2.00E+07	1.23E+09	5.74E+04	-999.79	-841.31	-725.18
	SD	1.48	8.91E+06	1.19E+09	1.17E+04	0.33	25.53	14.50
	p-value	(0.2116)	(0.8766)	(0.8883)	(0.6789)	(0.0537)	(0.9117)	(0.9000)
8E+04	**cNrGA**	-1399.98	2.01E+07	1.07E+09	4.84E+04	-999.93	-852.23	-722.53
	SD	0.05	1.07E+07	1.34E+09	1.00E+04	0.17	25.14	13.73
	cNrGA/CM/LRU	-1399.70	2.01E+07	1.18E+09	4.77E+04	-999.89	-852.45	-723.17
	SD	1.34	1.04E+07	1.37E+09	9.34E+03	0.24	24.72	14.65
	p-value	**(0.0051)**†	(0.9352)	(0.9705)	(0.8130)	(0.2398)	(0.9705)	(0.7394)
	cNrGA/CM/R	-1399.94	2.04E+07	1.13E+09	4.98E+04	-999.82	-851.94	-722.94
	SD	0.16	1.11E+07	1.22E+09	9.17E+03	0.38	24.63	13.41
	p-value	(0.0594)	(0.9352)	(0.8766)	(0.6100)	**(0.0351)**†	(0.8650)	(0.9117)
9E+04	**cNrGA**	-1399.92	1.88E+07	1.32E+09	4.67E+04	-999.95	-850.25	-724.80
	SD	0.26	8.51E+06	1.88E+09	1.09E+04	0.06	25.25	18.06
	cNrGA/CM/LRU	-1399.61	1.95E+07	1.18E+09	4.88E+04	-999.93	-852.52	-726.51
	SD	1.36	8.98E+06	1.43E+09	9.68E+03	0.11	25.13	16.21
	p-value	(0.8883)	(0.7394)	(0.6843)	(0.3366)	(0.2458)	(0.6735)	(0.8303)
	cNrGA/CM/R	-1399.88	1.89E+07	1.29E+09	4.85E+04	-998.82	-850.75	-724.12
	SD	0.37	8.24E+06	1.79E+09	8.16E+03	4.05	25.07	17.33
	p-value	(0.7506)	(0.8187)	(0.9823)	(0.5542)	(0.0877)	(0.9823)	(0.8073)
1E+05	**cNrGA**	-1399.87	2.00E+07	8.82E+08	4.41E+04	-999.83	-849.60	-724.57
	SD	0.25	7.57E+06	8.12E+08	1.04E+04	0.55	23.95	12.68
	cNrGA/CM/LRU	-1399.83	2.00E+07	1.07E+09	4.25E+04	-999.85	-848.81	-724.38
	SD	0.69	7.46E+06	8.30E+08	7.89E+03	0.38	24.85	13.18
	p-value	(0.2838)	(0.9941)	(0.3183)	(0.7450)	**(0.0122)**‡	(0.7845)	(0.9117)
	cNrGA/CM/R	-1399.95	2.02E+07	1.13E+09	4.07E+04	-999.93	-849.51	-724.10
	SD	0.10	7.37E+06	1.01E+09	9.00E+03	0.17	23.82	13.08
	p-value	(0.8073)	(0.9000)	(0.1809)	(0.2707)	(0.2062)	(0.9000)	(0.9234)

Table 6 Performance comparison of cNrGA, cNrGA/CM/LRU and cNrGA/CM/R. The best mean fitness values over three algorithms are shaded in grey. † means cNrGA is significantly superior to its variant with constant memory, while ‡ means cNrGA is significantly inferior

		f8	f9	f10	f11	f12	f13	f14
1E+04	**cNrGA**	-678.90	-559.07	-341.50	-362.79	-98.20	5.16	154.45
	SD	0.06	2.89	53.95	8.30	30.33	18.03	75.02
	cNrGA/CM/LRU	-678.90	-559.07	-341.50	-362.79	-98.20	5.16	154.45
	SD	0.06	2.89	53.95	8.30	30.33	18.03	75.02
	cNrGA/CM/R	-678.90	-559.07	-341.50	-362.79	-98.20	5.16	154.45
	SD	0.06	2.89	53.95	8.30	30.33	18.03	75.02
2E+04	**cNrGA**	-678.92	-566.76	-423.03	-391.53	-206.94	-44.14	-46.37
	SD	0.06	5.36	34.02	2.48	28.85	35.21	32.23
	cNrGA/CM/LRU	-678.92	-566.60	-420.75	-392.25	-209.73	-48.27	-52.74
	SD	0.06	5.82	37.36	2.20	22.35	37.33	18.35
	p-value	(1.0000)	(0.9470)	(0.8418)	(0.3555)	(0.9587)	(0.7062)	(0.5493)
	cNrGA/CM/R	-678.92	-566.57	-423.01	-391.46	-213.19	-46.89	-47.87
	SD	0.06	6.48	33.45	2.18	18.13	36.39	27.41
	p-value	(1.0000)	(0.9352)	(0.9470)	(0.7172)	(0.4643)	(0.6048)	(0.8883)
3E+04	**cNrGA**	-678.96	-570.21	-455.25	-396.67	-238.68	-64.41	-78.05
	SD	0.05	3.81	20.15	1.68	15.21	28.89	8.71
	cNrGA/CM/LRU	-678.96	-569.27	-455.32	-396.43	-239.43	-68.03	-74.87
	SD	0.05	4.26	19.22	1.17	12.53	25.44	23.60
	p-value	(1.0000)	(0.4290)	(0.9352)	(0.3711)	(0.8073)	(0.6100)	(0.8766)
	cNrGA/CM/R	-678.96	-570.21	-456.11	-396.61	-238.72	-65.00	-75.28
	SD	0.05	4.41	18.78	1.34	14.37	25.22	21.52
	p-value	(0.9000)	(0.7394)	(0.9941)	(0.8534)	(0.9000)	(0.9941)	(0.6414)
4E+04	**cNrGA**	-678.98	-570.13	-469.63	-398.03	-248.26	-87.35	-88.94
	SD	0.06	3.80	13.09	1.51	12.79	22.83	7.01
	cNrGA/CM/LRU	-678.98	-570.41	-468.72	-397.75	-247.60	-82.63	-88.87
	SD	0.06	4.04	12.32	1.73	13.86	29.07	3.85
	p-value	(0.9234)	(0.9823)	(0.6735)	(0.4825)	(0.9352)	(0.5895)	(0.1907)
	cNrGA/CM/R	-678.98	-570.60	-469.37	-397.91	-245.55	-84.01	-88.73
	SD	0.06	3.73	13.09	1.48	11.03	24.70	6.31
	p-value	(0.9234)	(0.6952)	(0.9470)	(0.4204)	(0.3329)	(0.7506)	(0.6735)
5E+04	**cNrGA**	-678.98	-571.24	-477.61	-398.65	-248.22	-70.76	-90.75
	SD	0.06	4.04	11.45	1.09	11.45	28.40	6.21
	cNrGA/CM/LRU	-678.98	-570.48	-475.72	-398.64	-247.35	-74.54	-91.66
	SD	0.06	3.45	13.05	1.21	12.77	32.27	5.66
	p-value	(0.9293)	(0.5201)	(0.3112)	(0.7394)	(0.9234)	(0.8534)	(0.5793)
	cNrGA/CM/R	-678.98	-570.97	-476.71	-398.45	-246.50	-75.12	-91.32
	SD	0.06	3.59	12.60	1.14	11.54	30.39	5.06
	p-value	(0.9293)	(0.7506)	(0.7618)	(0.3711)	(0.5201)	(0.4464)	(0.9705)
6E+04	**cNrGA**	-678.98	-570.88	-485.98	-398.86	-246.76	-81.04	-93.99
	SD	0.05	3.60	5.28	1.20	12.35	35.42	4.87
	cNrGA/CM/LRU	-678.98	-571.19	-486.29	-398.64	-246.21	-85.39	-93.58
	SD	0.05	4.01	5.45	1.16	12.40	30.08	3.95
	p-value	(0.9764)	(0.9234)	(0.6952)	(0.4204)	(0.8418)	(0.5793)	(0.3183)
	cNrGA/CM/R	-678.98	-570.91	-486.09	-398.64	-245.86	-83.40	-93.36
	SD	0.05	3.81	5.31	1.21	13.65	29.32	3.42
	p-value	(0.8824)	(0.9470)	(0.7283)	(0.2905)	(0.8303)	(0.9000)	(0.1494)
7E+04	**cNrGA**	-678.97	-571.54	-487.84	-398.62	-249.11	-75.40	-94.76
	SD	0.04	3.70	7.15	1.42	12.86	33.84	4.39
	cNrGA/CM/LRU	-678.97	-570.72	-487.92	-398.27	-248.36	-77.13	-94.67
	SD	0.05	2.73	6.31	1.85	12.75	34.08	5.43
	p-value	(0.8475)	(0.2973)	(0.9117)	(0.3711)	(0.7618)	(0.8303)	(0.5106)
	cNrGA/CM/R	-678.98	-572.37	-487.69	-398.71	-248.46	-76.84	-95.44
	SD	0.05	2.72	6.86	1.70	13.06	32.47	3.16
	p-value	(0.9058)	(0.3555)	(0.8650)	(0.9941)	(0.9000)	(0.7845)	(0.6952)
8E+04	**cNrGA**	-678.98	-570.76	-490.50	-398.67	-249.99	-86.76	-95.94
	SD	0.05	3.96	6.27	1.19	12.18	28.76	3.76
	cNrGA/CM/LRU	-678.98	-571.28	-489.81	-399.11	-249.19	-84.73	-91.81
	SD	0.04	2.92	7.26	1.06	13.69	28.71	22.45
	p-value	(0.7116)	(0.6100)	(0.9352)	(0.2340)	(0.8418)	(0.7731)	(0.9470)
	cNrGA/CM/R	-678.98	-570.72	-490.69	-398.85	-249.89	-85.83	-95.38
	SD	0.05	3.77	4.18	1.08	12.64	29.39	3.07
	p-value	(0.7958)	(0.8650)	(0.5493)	(0.9941)	(0.9941)	(0.8073)	(0.1373)
9E+04	**cNrGA**	-679.01	-570.92	-491.95	-398.86	-246.95	-86.19	-95.31
	SD	0.06	3.67	5.01	1.48	14.17	29.99	8.95
	cNrGA/CM/LRU	-679.01	-570.06	-492.00	-398.98	-247.67	-88.44	-95.46
	SD	0.06	4.43	5.03	1.16	13.14	28.78	4.24
	p-value	(0.8187)	(0.6309)	(0.8650)	(0.3953)	(0.9941)	(0.6952)	(0.5298)
	cNrGA/CM/R	-679.00	-570.58	-492.57	-398.99	-247.52	-85.60	-97.32
	SD	0.05	3.47	3.97	1.28	13.28	34.03	3.10
	p-value	(0.4687)	(0.8766)	(0.6735)	(0.7618)	(0.8883)	(0.9000)	(0.0905)
1E+05	**cNrGA**	-679.03	-571.93	-494.39	-398.40	-245.72	-87.57	-95.68
	SD	0.06	3.17	2.30	1.55	14.81	21.27	4.61
	cNrGA/CM/LRU	-679.02	-570.88	-494.49	-398.77	-245.41	-91.53	-96.62
	SD	0.07	3.93	2.00	1.31	15.56	19.68	2.35
	p-value	(0.5492)	(0.3329)	(0.9941)	(0.4918)	(0.9705)	(0.3953)	(0.9117)
	cNrGA/CM/R	-679.02	-571.39	-494.16	-398.60	-246.93	-88.81	-97.64
	SD	0.07	3.03	2.51	1.37	13.83	21.94	1.77
	p-value	(0.4778)	(0.6309)	(0.7172)	(0.9941)	(0.7394)	(0.8303)	(0.4204)

Table 7 Performance comparison of cNrGA, cNrGA/CM/LRU and cNrGA/CM/R. The best mean fitness values over three algorithms are shaded in grey. † means cNrGA is significantly superior to its variant with constant memory, while ‡ means cNrGA is significantly inferior

		f15	f16	f17	f18	f19	f20	f21
1E+04	**cNrGA**	8111.55	203.51	388.92	678.75	524.32	614.14	1323.28
	SD	381.97	0.43	11.82	19.60	18.08	0.71	133.10
	cNrGA/CM/LRU	8111.55	203.51	388.92	678.75	524.32	614.14	1323.28
	SD	381.97	0.43	11.82	19.60	18.08	0.71	133.10
	cNrGA/CM/R	8111.55	203.51	388.92	678.75	524.32	614.14	1323.28
	SD	381.97	0.43	11.82	19.60	18.08	0.71	133.10
2E+04	**cNrGA**	7905.97	203.20	344.03	647.58	503.80	612.98	1054.17
	SD	253.17	0.32	2.26	34.97	0.92	0.29	73.37
	cNrGA/CM/LRU	7905.97	203.20	343.98	640.73	503.89	612.95	1049.14
	SD	253.17	0.32	2.87	38.64	0.77	0.27	73.79
	p-value	(1.0000)	(1.0000)	(0.7283)	(0.5154)	(0.3112)	(0.5493)	(0.7394)
	cNrGA/CM/R	7893.34	203.20	344.42	647.74	503.92	612.99	1056.95
	SD	266.24	0.32	2.46	35.64	0.74	0.30	72.00
	p-value	(0.8708)	(1.0000)	(0.5997)	(0.9000)	(0.3953)	(0.9587)	(0.7731)
3E+04	**cNrGA**	7697.36	203.08	335.88	622.45	502.57	612.57	1004.11
	SD	310.14	0.44	1.23	39.83	0.81	0.38	76.23
	cNrGA/CM/LRU	7760.14	203.08	336.31	618.90	502.58	612.65	1005.68
	SD	277.93	0.44	1.53	41.14	0.73	0.37	75.12
	p-value	(0.5248)	(1.0000)	(0.3871)	(0.7618)	(0.8650)	(0.4508)	(0.8187)
	cNrGA/CM/R	7703.18	203.09	336.39	625.75	502.71	612.61	1007.28
	SD	299.80	0.44	1.37	40.94	0.69	0.41	75.29
	p-value	(0.9352)	(0.9764)	(0.2009)	(0.6627)	(0.1715)	(0.5592)	(0.5895)
4E+04	**cNrGA**	7520.08	203.04	333.73	586.39	502.00	612.13	1003.55
	SD	584.01	0.38	1.03	42.81	0.45	0.65	77.77
	cNrGA/CM/LRU	7423.63	203.04	333.97	585.00	501.97	612.15	1007.84
	SD	661.63	0.37	1.41	32.83	0.50	0.68	75.23
	p-value	(0.4507)	(0.9234)	(0.8073)	(0.8303)	(0.9587)	(0.9058)	(0.7062)
	cNrGA/CM/R	7467.35	203.03	333.96	592.97	501.99	612.17	1004.00
	SD	651.80	0.36	0.93	37.04	0.44	0.61	77.07
	p-value	(0.4076)	(0.9823)	(0.2838)	(0.3042)	(1.0000)	(0.7958)	(0.7618)
5E+04	**cNrGA**	7510.52	202.83	332.75	557.72	501.79	611.81	1009.90
	SD	376.54	0.35	0.70	27.87	0.63	0.64	86.29
	cNrGA/CM/LRU	7477.33	202.84	333.01	569.72	501.78	611.84	1008.94
	SD	419.73	0.34	0.71	37.63	0.44	0.54	87.13
	p-value	(0.9176)	(0.9941)	(0.2707)	(0.4643)	(0.5997)	(0.7845)	(0.6100)
	cNrGA/CM/R	7491.56	202.88	333.01	562.46	501.85	611.72	1009.65
	SD	418.68	0.32	1.11	30.09	0.45	0.67	86.58
	p-value	(0.9705)	(0.5641)	(0.6204)	(0.4733)	(0.3478)	(0.6843)	(0.7506)
6E+04	**cNrGA**	7027.74	202.93	332.20	551.92	501.78	611.81	975.25
	SD	1273.96	0.31	0.62	32.62	0.50	0.63	67.50
	cNrGA/CM/LRU	6870.52	202.95	332.42	564.44	501.65	611.73	979.44
	SD	1267.20	0.33	0.64	34.22	0.43	0.58	66.81
	p-value	(0.4463)	(0.9646)	(0.1154)	(0.1224)	(0.4643)	(0.6843)	(0.7731)
	cNrGA/CM/R	6912.04	202.93	332.63	567.04	501.69	611.76	977.74
	SD	1266.14	0.31	0.94	35.81	0.53	0.56	65.59
	p-value	(0.6099)	(0.8766)	**(0.0315)**†	(0.1055)	(0.3632)	(0.7731)	(0.7394)
7E+04	**cNrGA**	6931.76	202.83	332.09	543.22	501.44	611.52	992.47
	SD	1130.15	0.29	1.28	21.07	0.37	0.62	75.51
	cNrGA/CM/LRU	7075.70	202.80	331.96	549.34	501.55	611.75	993.96
	SD	944.06	0.31	0.81	25.39	0.44	0.57	75.75
	p-value	(0.8360)	(0.6842)	(0.7958)	(0.3790)	(0.2581)	(0.1453)	(0.9470)
	cNrGA/CM/R	6855.25	202.82	331.70	548.22	501.59	611.72	994.23
	SD	1250.09	0.29	0.50	23.86	0.45	0.67	75.50
	p-value	(0.8245)	(0.7787)	(0.4119)	(0.6414)	(0.1624)	(0.1907)	(0.9587)
8E+04	**cNrGA**	5882.96	202.75	331.69	543.84	501.42	611.47	1018.14
	SD	1716.14	0.31	0.70	34.25	0.35	0.71	93.20
	cNrGA/CM/LRU	6356.92	202.74	331.64	547.56	501.39	611.46	1018.78
	SD	1511.99	0.32	0.48	30.60	0.30	0.68	92.66
	p-value	(0.3255)	(0.8302)	(0.8766)	(0.3711)	(0.8883)	(0.8650)	(0.5692)
	cNrGA/CM/R	6053.14	202.73	331.59	545.11	501.38	611.41	1020.13
	SD	1668.68	0.31	0.49	31.96	0.39	0.68	91.73
	p-value	(0.6361)	(0.7116)	(0.7283)	(0.7506)	(0.6952)	(0.8650)	(0.4918)
9E+04	**cNrGA**	5863.06	202.72	331.60	545.18	501.34	611.52	980.99
	SD	1613.16	0.32	0.77	32.07	0.47	0.70	74.17
	cNrGA/CM/LRU	6195.90	202.75	331.45	542.52	501.28	611.62	979.64
	SD	1530.38	0.37	0.55	26.00	0.41	0.69	73.01
	p-value	(0.5298)	(0.8015)	(0.3183)	(0.8650)	(0.7283)	(0.5395)	(0.7172)
	cNrGA/CM/R	6131.57	202.75	331.57	541.40	501.33	611.54	981.55
	SD	1514.18	0.33	0.67	24.28	0.39	0.54	73.71
	p-value	(0.5642)	(0.7449)	(0.9941)	(0.9117)	(0.8883)	(0.9941)	(0.4464)
1E+05	**cNrGA**	6096.50	202.73	331.34	525.57	501.34	611.72	1010.46
	SD	1512.14	0.27	0.46	23.85	0.41	0.62	78.14
	cNrGA/CM/LRU	5554.30	202.77	331.43	532.48	501.31	611.55	1013.48
	SD	1674.53	0.28	0.66	18.89	0.37	0.61	79.36
	p-value	(0.2643)	(0.5893)	(0.8650)	(0.1907)	(0.9352)	(0.3953)	(0.9352)
	cNrGA/CM/R	5663.53	202.71	331.49	528.56	501.26	611.50	1010.98
	SD	1653.66	0.32	0.43	19.85	0.31	0.69	78.51
	p-value	(0.3219)	(0.8824)	(0.1055)	(0.3871)	(0.5011)	(0.1907)	(0.4643)

Table 8 Performance comparison of cNrGA, cNrGA/CM/LRU and cNrGA/CM/R. The best mean fitness values over three algorithms are shaded in grey. † means cNrGA is significantly superior to its variant with constant memory, while ‡ means cNrGA is significantly inferior

		f22	f23	f24	f25	f26	f27	f28
1E+04	**cNrGA**	1242.49	9220.74	1270.26	1389.21	1464.95	2362.81	2578.22
	SD	125.46	396.17	8.85	7.40	74.65	118.72	172.39
	cNrGA/CM/LRU	1242.49	9220.74	1270.26	1389.21	1464.95	2362.81	2578.22
	SD	125.46	396.17	8.85	7.40	74.65	118.72	172.39
	cNrGA/CM/R	1242.49	9220.74	1270.26	1389.21	1464.95	2362.81	2578.22
	SD	125.46	396.17	8.85	7.40	74.65	118.72	172.39
2E+04	**cNrGA**	979.70	8787.41	1265.08	1385.85	1474.58	2259.81	2135.26
	SD	27.91	470.44	8.68	5.90	75.41	100.12	333.66
	cNrGA/CM/LRU	978.12	8769.79	1265.18	1385.50	1475.75	2256.76	2172.97
	SD	19.31	473.13	8.78	5.88	75.67	98.19	336.24
	p-value	(0.7845)	(0.9058)	(0.9941)	(0.9352)	(0.8130)	(0.9352)	(0.4643)
	cNrGA/CM/R	976.03	8813.54	1265.82	1385.88	1474.90	2254.65	2140.94
	SD	15.25	459.01	8.76	5.33	75.46	112.01	305.74
	p-value	(0.9234)	(0.8015)	(0.6735)	(0.9234)	(0.9058)	(0.6952)	(0.7506)
3E+04	**cNrGA**	942.78	8346.75	1266.95	1383.01	1475.36	2211.67	1950.22
	SD	29.78	797.79	10.16	6.49	79.74	77.19	366.48
	cNrGA/CM/LRU	943.27	8124.01	1267.20	1382.65	1467.33	2218.92	1953.42
	SD	31.50	1105.81	10.42	6.50	79.45	92.91	345.53
	p-value	(0.6735)	(0.5443)	(0.9352)	(0.8073)	(0.8650)	(0.8650)	(0.7958)
	cNrGA/CM/R	942.67	8481.45	1266.82	1383.26	1475.34	2220.07	1952.08
	SD	31.67	498.37	9.92	6.50	79.80	85.08	327.63
	p-value	(0.8534)	(0.9117)	(0.9234)	(0.8073)	(0.9646)	(0.7172)	(0.5493)
4E+04	**cNrGA**	933.53	7978.01	1263.20	1383.20	1443.64	2256.42	1784.59
	SD	20.47	1286.58	9.80	7.24	70.41	65.73	165.61
	cNrGA/CM/LRU	931.12	8137.84	1263.25	1382.98	1443.70	2250.57	1823.45
	SD	28.27	1059.05	8.66	7.41	70.59	73.28	238.04
	p-value	(0.8418)	(0.9411)	(0.9587)	(0.9000)	(0.9941)	(0.7731)	(0.9823)
	cNrGA/CM/R	928.81	7973.34	1263.09	1383.24	1444.21	2259.42	1787.18
	SD	29.03	1078.89	9.22	7.49	71.41	64.81	150.52
	p-value	(0.2772)	(0.5106)	(0.8534)	(0.8650)	(0.9234)	(0.9234)	(0.4918)
5E+04	**cNrGA**	924.27	7737.93	1263.48	1380.31	1447.88	2239.04	1816.23
	SD	28.96	1027.08	7.00	5.94	71.85	75.35	323.03
	cNrGA/CM/LRU	923.93	7538.69	1263.90	1379.95	1447.87	2222.40	1811.65
	SD	27.90	1220.18	6.99	6.73	71.94	59.99	323.32
	p-value	(0.7506)	(0.5642)	(0.9000)	(0.8883)	(1.0000)	(0.3255)	(0.5298)
	cNrGA/CM/R	927.06	7911.40	1264.19	1380.54	1448.07	2235.13	1864.60
	SD	27.34	1104.80	7.05	6.30	72.16	66.59	372.91
	p-value	(0.2116)	(0.2804)	(0.7172)	(0.7958)	(0.9058)	(0.6627)	(0.9234)
6E+04	**cNrGA**	926.22	7403.49	1262.66	1382.19	1457.64	2228.86	1783.00
	SD	22.14	1298.15	8.99	6.55	75.06	83.78	247.11
	cNrGA/CM/LRU	926.18	6623.89	1262.14	1382.09	1457.16	2227.23	1790.88
	SD	21.64	1708.82	9.08	6.44	74.68	80.09	275.33
	p-value	(0.8883)	(0.1039)	(0.8073)	(0.9470)	(0.8592)	(0.9705)	(0.5106)
	cNrGA/CM/R	926.10	6860.50	1261.85	1382.20	1457.87	2215.42	1764.05
	SD	18.91	1626.94	8.86	6.44	75.19	87.63	233.18
	p-value	(0.6100)	(0.1737)	(0.5895)	(0.9352)	(0.9176)	(0.6309)	(0.8303)
7E+04	**cNrGA**	916.72	6784.56	1264.46	1384.08	1448.44	2259.86	1743.57
	SD	33.52	1543.71	7.44	8.45	73.43	83.56	148.82
	cNrGA/CM/LRU	920.29	6162.81	1264.52	1383.92	1448.51	2252.91	1740.40
	SD	41.03	1840.78	7.27	7.90	73.51	88.21	145.00
	p-value	(0.8073)	(0.2612)	(0.8766)	(0.8883)	(0.9941)	(0.7062)	(0.5997)
	cNrGA/CM/R	919.92	6297.60	1264.66	1384.17	1448.52	2258.86	1760.84
	SD	45.70	1619.75	7.38	8.05	73.50	83.34	178.76
	p-value	(0.4733)	(0.2311)	(0.9117)	(0.9705)	(0.8073)	(0.9470)	(0.8883)
8E+04	**cNrGA**	911.57	6218.06	1265.20	1381.94	1475.53	2213.64	1771.82
	SD	26.20	1785.58	7.66	6.19	81.02	88.80	229.64
	cNrGA/CM/LRU	913.22	6047.19	1264.87	1382.21	1475.60	2208.53	1750.25
	SD	29.77	1679.74	8.45	6.35	81.01	89.14	192.02
	p-value	(0.5298)	(0.7283)	(0.9823)	(0.8766)	(0.9470)	(0.9000)	(0.1907)
	cNrGA/CM/R	916.01	6059.08	1264.46	1381.85	1476.09	2214.68	1713.86
	SD	35.79	1876.48	8.24	6.51	81.59	91.12	9.24
	p-value	(0.8650)	(0.8534)	(0.7618)	(0.9705)	(0.8187)	(0.9117)	(0.1413)
9E+04	**cNrGA**	917.07	5759.15	1262.37	1382.91	1450.60	2256.58	1755.18
	SD	27.31	1585.09	8.21	6.58	73.10	97.88	233.16
	cNrGA/CM/LRU	918.01	5526.98	1262.22	1382.71	1445.06	2261.47	1784.73
	SD	30.40	1512.44	8.50	6.86	70.23	99.47	274.89
	p-value	(0.9823)	(0.6100)	(0.9587)	(0.9587)	(0.8016)	(0.8534)	(0.3255)
	cNrGA/CM/R	917.93	5541.16	1262.68	1382.49	1445.07	2251.66	1797.95
	SD	28.99	1387.49	8.40	6.25	70.01	100.72	275.20
	p-value	(0.9587)	(0.6843)	(0.8534)	(0.7958)	(0.7227)	(0.8883)	(0.6100)
1E+05	**cNrGA**	915.51	4620.27	1263.72	1383.17	1458.02	2224.21	1804.87
	SD	30.47	868.03	8.92	6.30	76.76	87.34	314.38
	cNrGA/CM/LRU	917.12	4786.66	1263.07	1383.41	1458.48	2216.35	1799.51
	SD	26.57	1030.02	8.85	7.08	77.40	84.71	286.56
	p-value	(0.1958)	(0.6843)	(0.6952)	(0.7845)	(0.7958)	(0.6414)	**(0.0421)**‡
	cNrGA/CM/R	915.21	5439.27	1264.18	1382.65	1458.57	2237.45	1813.62
	SD	27.97	1249.86	8.86	7.08	77.39	79.74	326.80
	p-value	(0.6309)	**(0.0144)**†	(0.9117)	(0.6735)	(0.6309)	(0.6520)	(0.0905)

Table 9 Mean results (mean) and standard deviation (SD) of cNrGA/CM/LRU cNrGA/CM/R, real-coded GA and SPSO 2011

	cNrGA	cNrGA/ CM/ LRU	cNrGA/ CM/R	RGA	SPSO 2011
f1					
Mean	−1399.87	−1399.83	−1399.95	−1397.75	−1400
SD	0.25	0.69	0.1	2.01	0
f2					
Mean	2.00E+07	2.00E+07	2.02E+07	2.83E+07	4.76E+05
SD	7.57E+06	7.46E+06	7.37E+06	8.58E+06	2.00E+05
f3					
Mean	8.82E+08	1.07E+09	1.13E+09	1.81E+09	1.71E+08
SD	8.12E+08	8.30E+08	1.01E+09	1.77E+09	2.22E+08
f4					
Mean	4.41E+04	4.25E+04	4.07E+04	2.86E+04	1.63E+04
SD	10371.83	7894.01	8997.42	8106.26	5375.7
f5					
Mean	−999.83	−999.85	−999.93	−997.04	−1000
SD	0.55	0.38	0.17	0.84	0
f6					
Mean	−849.6	−848.81	−849.51	−827.98	−857.66
SD	23.95	24.85	23.82	32.16	29.45
f7					
Mean	−724.57	−724.38	−724.1	−706.11	−726.91
SD	12.68	13.18	13.08	34.08	23.18
f8					
Mean	−679.03	−679.02	−679.02	−679	−679.05
SD	0.06	0.07	0.07	0.09	0.06
f9					
Mean	−571.93	−570.88	−571.39	−569.79	−575.23
SD	3.17	3.93	3.03	4.39	4.64
f10					
Mean	−494.39	−494.49	−494.16	−463.27	−499.73
SD	2.3	2	2.51	10.08	0.15
f11					
Mean	−398.4	−398.77	−398.6	−385.02	−287.17
SD	1.55	1.31	1.37	3.56	39.06
f12					
Mean	−245.72	−245.41	−246.93	−172.89	−199.27
SD	14.81	15.56	13.83	23.63	29.2
f13					
Mean	−87.57	−91.53	−88.81	−36.7	−33.31
SD	21.27	19.68	21.94	29.2	36.17
f14					
Mean	−95.68	−96.62	−97.64	1032.17	5250.77
SD	4.61	2.35	1.77	297.7	885.49

Table 9 continued

	cNrGA	cNrGA/ CM/ LRU	cNrGA/ CM/R	RGA	SPSO 2011
f15					
Mean	6096.5	5554.3	5663.53	5683	6003.29
SD	1512.14	1674.53	1653.66	620.64	704.74
f16					
Mean	202.73	202.77	202.71	202.6	202.29
SD	0.27	0.28	0.32	0.53	0.33
f17					
Mean	331.34	331.43	331.49	355.17	476.01
SD	0.46	0.66	0.43	5.99	37.81
f18					
Mean	525.57	532.48	528.56	632.8	599.11
SD	23.85	18.89	19.85	21.38	19.25
f19					
Mean	501.34	501.31	501.26	505.99	511.12
SD	0.41	0.37	0.31	1.55	5.36
f20					
Mean	611.72	611.55	611.5	613.28	614.21
SD	0.62	0.61	0.69	0.86	1.35
f21					
Mean	1010.46	1013.48	1010.98	1044.9	1007.85
SD	78.14	79.36	78.51	63.44	115.1
f22					
Mean	915.51	917.12	915.21	1915.99	5879.36
SD	30.47	26.57	27.97	318.26	947.74
f23					
Mean	4620.27	4786.66	5439.27	7180.95	7118.24
SD	868.03	1030.02	1249.86	958.14	1084.37
f24					
Mean	1263.72	1263.07	1264.18	1256.46	1269.73
SD	8.92	8.85	8.86	10.61	12.67
f25					
Mean	1383.17	1383.41	1382.65	1388.07	1388.44
SD	6.3	7.08	7.08	13.24	9.39
f26					
Mean	1458.02	1458.48	1458.57	1479.53	1488.28
SD	76.76	77.4	77.39	85.38	78.68
f27					
Mean	2224.21	2216.35	2237.45	2268.23	2202.28
SD	87.34	84.71	79.74	112.52	67.11
f28					
Mean	1804.87	1799.51	1813.62	1825.81	1961.9
SD	314.38	286.56	326.8	37.76	758.68

Table 10 Significance tests of the three variants of cNrGA vs. real-coded GA and SPSO 2011 [+ means cNrGA (or its variant) is significantly superior, while − means significantly inferior]

	cNrGA		cNrGA/CM/LRU		cNrGA/CM/R	
	RGA	SPSO 2011	RGA	SPSO 2011	RGA	SPSO 2011
f1	+	−	+	−	+	−
f2	+	−	+	−	+	−
f3	+	−	+	−	+	−
f4	+	−	+	−	−	−
f5	+	−	+	−	+	−
f6	+	0	+	0	+	0
f7	+	0	+	0	+	0
f8	0	0	0	0	0	0
f9	0	−	0	−	0	−
f10	+	−	+	−	+	−
f11	+	+	+	+	+	+
f12	+	+	+	+	+	+
f13	+	+	+	+	+	+
f14	+	+	+	+	+	+
f15	0	0	0	0	0	0
f16	0	−	0	−	0	−
f17	+	+	+	+	+	+
f18	+	+	+	+	+	+
f19	+	+	+	+	+	+
f20	+	+	+	+	+	+
f21	+	−	+	−	+	−
f22	+	+	+	+	+	+
f23	+	+	+	+	+	+
f24	+	+	+	+	−	0
f25	0	+	0	0	0	+
f26	+	0	+	0	+	0
f27	0	0	0	0	0	0
f28	+	+	+	+	+	+
Superior	22	13	22	12	20	12
Inferior	0	9	0	9	2	9

References

1. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
2. Davis L (ed) (1991) Handbook of genetic algorithms, vol 115. Van Nostrand Reinhold, New York
3. Friedrich T, Hebbinghaus N, Neumann F (2007) Rigorous analyses of simple diversity mechanisms. In: Proceedings of the 9th annual conference on genetic and evolutionary computation (GECCO). ACM, New York, pp 1219–1225
4. Ronald S (1998) Duplicate genotypes in a genetic algorithm. In: Proceedings of the IEEE world congress on computational intelligence (WCCI), pp 793–798
5. Povinelli RJ, Feng X (1999) Improving genetic algorithms performance by hashing fitness values. In: Proceedings of the artificial neural networks in engineering (ANNIE), pp 399–404
6. Kratica J (1999) Improving performances of the genetic algorithm by caching. Comput Artif Intell 18(3):271–283
7. Yuen SY, Chow CK (2009) A genetic algorithm that adaptively mutates and never revisits. IEEE Trans Evol Comput 13(2):454–472
8. Glover F, Laguna M (1997) Tabu search. Kluwer, Norwell
9. Chow CK, Yuen SY (2010) Continuous non-revisiting genetic algorithm with random search space re-partitioning and one-gene-flip mutation. In: Proceedings of the IEEE congress on evolutionary computation (CEC) , Barcelona. doi:10.1109/CEC.2012.6252926
10. Chow CK, Yuen SY (2012) Continuous Non-revisiting Genetic Algorithm with Overlapped Search Sub-region. In Proceedings of the IEEE congress on evolutionary computation (CEC), Brisbane, QLD, p 1–8. doi:10.1109/CEC.2010.5586046
11. Yuen SY, Chow CK (2008) A non-revisiting simulated annealing algorithm. In: Proceedings of the IEEE congress on evolutionary computation (CEC), pp 1886–1892
12. Chow CK, Yuen SY (2008) A non-revisiting particle swarm optimization. In: Proceedings of the IEEE congress on evolutionary computation (CEC), pp 1879–1885
13. Du J, Rada R (2012) Memetic algorithms, domain knowledge, and financial investing. Memet Comput 4(2):109–125
14. Young CN, LeBrese C, Zou JJ, Leo CJ (2013) A robust search paradigm with enhanced vine creeping optimization. Eng Optim 45(2):225–244
15. Akay B, Karaboga D (2012) A modified artificial bee colony algorithm for real-parameter optimization. Inf Sci 192:120–142
16. Jadon SS, Bansal JC, Tiwari R, Sharma H (2015) Accelerating artificial bee colony algorithm with adaptive local search. Memet Comput 7(3):215–230
17. Gandomi AH, Yang XS (2012) Evolutionary boundary constraint handling scheme. Neural Comput Appl 21:1449–1462
18. Wang Y, Li HX, Huang T, Li L (2014) Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. Appl Soft Comput 18:232–247
19. Wang Y, Wang BC, Li HX, Yen GG (2015) Incorporating objective function information into the feasibility rule for constrained evolutionary optimization. IEEE Trans Cybern (in press). doi:10.1109/TCYB.2015.2493239
20. Chu W, Gao X, Sorooshian S (2011) Handling boundary constraints for particle swarm optimization in high-dimensional search space. Inf Sci 181:4569–4581
21. Zambrano-Bigiarini M, Clerc M, Rojas R (2013) Standard particle swarm optimisation 2011 at CEC-2013: a baseline for future PSO improvements. In: Proceedings of the IEEE congress on evolutionary computation (CEC), pp 2337–2344
22. Hansen N (2006) The CMA evolution strategy: a comparing review. In: Proceedings on towards a new evolutionary computation, pp 75–102
23. Hansen N (2011) The CMA evolutionary strategy: a tutorial. In: Technical report. http://www.lri.fr/~hansen/cmatutorial.pdf. Accessed 14 June 2015
24. Mack CA (2011) Fifty years of Moore's law. IEEE Trans Semicond Manuf 24(2):202–207
25. Lou Y, Yuen SY (2015) Non-revisiting genetic algorithm with constant memory. In: Proceedings of the IEEE systems, man, and cybernetics (SMC), pp 1714–1719
26. Eshelman LJ, Schaffer JD (1992) Real-coded genetic algorithms and interval-schemata. In: Proceedings of the international conference on genetic algorithms (ICGA), pp 187–202
27. Lihu A, Holban S, Popescu O-A (2012) Real-valued genetic algorithms with disagreements. Memet Comput 4(4):317–325
28. Heris SMK (2015) Implementation of real-coded genetic algorithm in MATLAB. http://www.yarpiz.com. Accessed 23 Aug 2015

29. Chow CK, Yuen SY (2011) An evolutionary algorithm that makes decision based on the entire previous search history. IEEE Trans Evol Comput 15(6):741–769
30. Leung SW, Yuen SY, Chow CK (2012) Parameter control system of evolutionary algorithm that is aided by the entire search history. Appl Soft Comput 12(9):3063–3078
31. www.ee.cityu.edu.hk/~syyuen/Public/Code.html. Accessed 28 Dec 2015
32. Liang JJ, Qu B-Y, Suganthan PN, Hernández-Díaz AG (2013) Problem definitions and evaluation criteria for the CEC 2013 special session and competition on real-parameter optimization. In: Technical report 2012, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou and technical report, Nanyang Technological University, Singapore
33. Karafotias G, Hoogendoorn M, Eiben AE (2014) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput 19(2):167–187
34. Sedgewick R (2002) Algorithms in Java, parts 1–4. Addison-Wesley, Boston
35. Knuth DE (1998) The art of computer programming: sorting and searching. Pearson Education, London