Improving GA Search Reliability Using Maximal Hyper-Rectangle Analysis

Chongshan Zhang
Computer Science Department
University of Georgia
Athens, GA 30602
01-706-534-2907
czhang@cs.uga.edu

Khaled Rasheed
Computer Science Department
University of Georgia
Athens, GA 30602
01-706-534-3444
khaled@cs.uga.edu

ABSTRACT
In Genetic algorithms it is not easy to evaluate the confidence level in whether a GA run may have missed a complete area of good points, and whether the global optimum was found. We accept this but hope to add some degree of confidence in our results by showing that no large gaps were left unvisited in the search space. This can be achieved to some extent by inserting new individuals in big empty spaces. However it is not easy to find the biggest empty spaces, particularly in multi-dimensional problems. For a GA problem, however, it is not necessary to find the exact biggest empty spaces; a sufficiently large empty space is good enough to insert new individuals. In this paper, we present a method to find a sufficiently large empty Hyper-Rectangle for new individual insertion in a GA while keeping the computational complexity as a polynomial function. Its merit is demonstrated in several domains.

Categories and Subject Descriptors
I.2.8 [**Computing Methodologies**]: Artificial Intelligence – *problem solving, control method, and search.*

General Terms
Algorithm, Performance.

Keywords
Genetic Algorithms, Optimization, Maximal Hyper-Rectangle.

1. INTRODUCTION
Genetic algorithm[9] search is a probabilistic search approach which is founded on the ideas of evolutionary processes. The GA procedure is based on the Darwinian principle of survival of the fittest. For the general GA, first an initial population is created containing a number of individuals. Each individual has an associated fitness measure, typically representing an objective value. Only individuals with relatively high fitness are selected to reproduce offspring by crossover and mutation, while those with lower fitness will get discarded from the population. The result is another set of individuals based on the previous population, leading to subsequent populations with better individual fitness in most cases. The GA approach has repeatedly proven to be a robust search and optimization method in numerous theoretical and practical domains.

GECCO'05, June 25-29, Washington, DC, USA.

However there are potential problems. If we strictly adhere to the Darwinian evolution paradigm, the population in every generation only weakly depends on the initial population, because all offspring are created from their parents. However, only the first generation is created randomly. It is inevitable as the GA converges that exploitation will replace the initial exploration. Nevertheless, it is hard to guarantee that this will not happen too soon leading to premature convergence. One way to reduce the risk is to ensure that some individuals are created randomly in every generation. It is usually observed that as the search progresses, the population gets gradually concentrated in one or a few regions which may or may not contain the global optimum. The capability of exploring new space will be weaker. Niching methods[8] attempt to reduce the risk using sharing, crowding or other diversity promotion techniques. The problem with these methods is that they are implicit rather than explicit in their attempt to cover the search space. Despite the use of Niching methods, there could be large regions of the search space that are never explored. To overcome these drawbacks, our idea is to create some individuals directly in each generation, placing them in big empty spaces to promote exploration. In doing this, the convergence may be slightly slower but the probability of premature convergence to local sub-optima is significantly reduced. By inserting some individuals in big empty spaces, our confidence in the results increases because no large gaps exist in the search space.

However, finding the biggest empty space in a multi-dimension space is not easy. The first step is to define empty space. We can use empty hyper-sphere, empty hyper-rectangle or other shapes. For example, if we adopt empty hyper-rectangle, it is still an infinite set. Liang-ping Ku et al.[1] proposed using Maximal Hyper Rectangle (MHR) to measure the empty space and tried to find the biggest MHR. In this paper, we adopt the concept of MHR to describe the empty space (Figure 1), defined as follows:

(1) All the sides of MHR are parallel to the respective axis, and orthogonal to the rest.

(2) MHR does not contain any points in its interior.

(3) No other MHR exists which either contains or falls entirely within the interior of the MHR. In other words, on each surface of the MHR, there is at lease one point.

We adopt the MHR concept because it is representative of all possible empty hyper-rectangles and the number of MHRs is finite. Secondly, it is easer to handle because the space and points are usually described in the Cartesian coordinate system.

The problem of finding empty hyper-rectangles has been studied repeatedly in the literature. In [1], the computational complexity to find the biggest MHR is $O(n^{2k-1}k^3(\lg n)^2)$, where n is the number of points and k is the number of dimensions in the dataset. Liu et al.[2] motivate the use of empty space knowledge for discovering constraints. Their proposed algorithm runs in $O(n^{2(k-1)}k^3(\lg n)^2)$. Even in low dimensions this algorithm is impractical for large values of n. In an attempt to address both the time and space complexity, they proposed only maintaining maximal empty hyper-rectangles which are larger than some size. However the number of MHRs may still be very large because it is difficult to correctly set the size, a priori, in some problems. Furthermore, many MHRs are largely overlapping. Edmonds et al.[3] proposed finding all MHRs by considering each 0-entry$<x, y>$ of M one at a time, row by row, where M is an $|X|\times|Y|$ matrix (for two dimensions), X and Y denote the set of distinct values in the data set in each of the dimensions. Their proposed algorithm runs in $O(n^{2(k-1)}k)$. However, it is not applicable for continuous spaces. Other approaches have been proposed that use decision tree classifiers to approximately separate occupied space from unoccupied space, then post-process the discovered regions to determine MHRs[4]. These methods do not guarantee that all maximal empty rectangles are found.

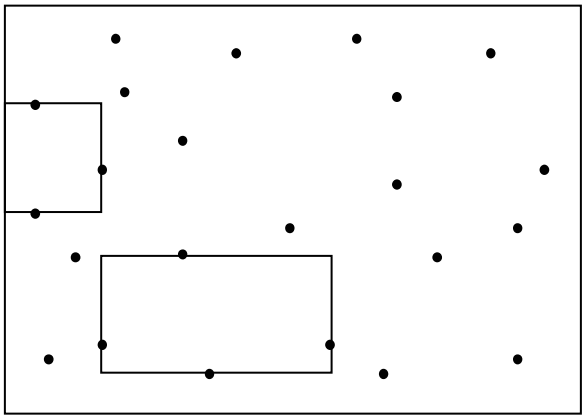


Figure 1. Two examples of MHRs in a 2-D space

Despite the extensive literature on this problem, none of the known algorithms are effective for large data sets. Fortunately, for the purpose of improving the GA reliability, we do not have to find the exact biggest MHR, a sufficiently big MHR is good enough.

Now the problem we will consider is as follows: Given a k-dimensional space S, where each dimension r is bounded by a minimum (S_{minr}) and a maximum (S_{maxr}) value. S contains n points (individuals). The problem is to find a sufficiently large MHR in which to insert a new individual.

The rest of the paper is organized as follows: In Section 2, we describe the proposed algorithm, and check the time complexity of the algorithm by setting points in S randomly and finding the calculation time for different numbers of points and dimensions. In Section 3, we apply this algorithm to some real GA problems. Section 4 presents the conclusion and future work.

2. THE PROPOSED APPROACH

The main idea of the algorithm is as follows: There are n points in a k-dimensional space S (then S has $2k$ surfaces), the middle point of two points P_i and P_j is M_{ij}. Near each M_{ij}, there is always a small empty hype-rectangle the center of which is M_{ij} and its volume is ≥ 0. We expand the empty rectangle with the same speed along all directions until some surface, for example, surface k_+, meets a point. Then along direction k_+, the expanding will stop, while along direction k_-, which is the opposite of k_+, the speed of expansion is doubled. The speed of expansion along other directions remains the same. The expansion continues until some surface meets a point. The expansion will stop when all surface meet point (Figure 2).

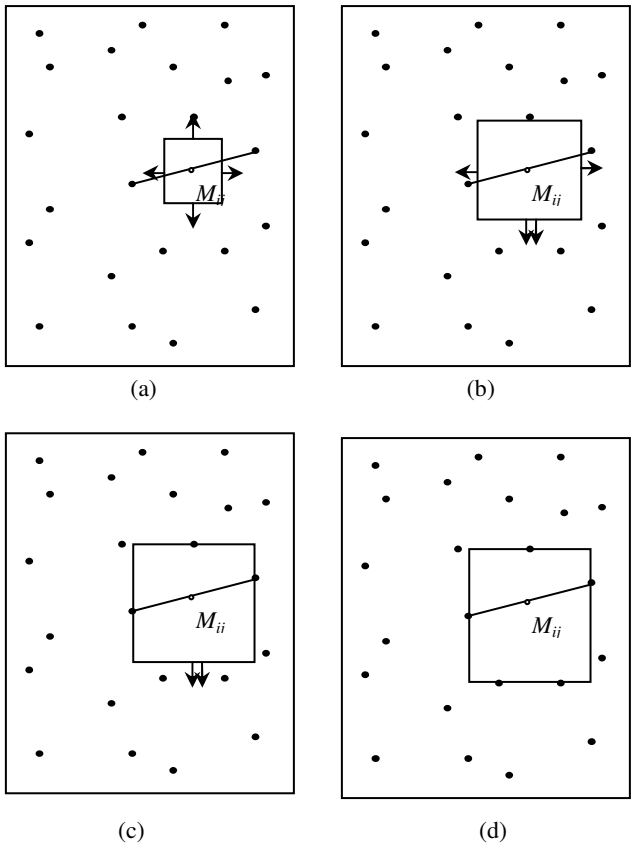


Figure2. Example to illustrate the expansion process of a MHR in a 2-D space

(a) an empty rectangle expands starting from a middle point; (b) one surface meets a point; (c) the other two surfaces meet two other points; (d) all surfaces meet points.

We use a variable "time t" to measure the expansion of the empty hype-rectangle. When t=0, the volume of the empty rectangle =0,

i.e. the distance between each surface and M_{ij}, d_r is 0. At the beginning, the speed along all directions, $v = 1$. If surface r_+ has met a point, we set $v_{r+} = 0$ and $v_{r-} = 2$. Then the distance between each surface and M_{ij}, $d_r = v_r * t$.

When the empty rectangle expands, we can increase t by a small amount (Δt) each time, and then check whether some surface met some point. However, if Δt is too small, we will have to check too many times. If Δt is too big, the time that some surface meets some points may lie between t and $t+\Delta t$.

To solve this problem, instead, we try to find which surface and point will meet first. With the current expansion speed, each point can be approached at some time t_i. We take P_i as an example. For each direction, we can find t_{ir} according to the distance between the current position of surface r and P_i and the speed v_{ir}. Suppose the maximum among t_{i1}, t_{i2}, …t_{i2k} is t_{ir}, then t_{ir} is the time (t_i) that the surface s of the empty rectangle expanding with the current speed meets P_i. To deal with the boundary, we add two points P_{n+1} and P_{n+2} to represent the boundary when we do the expansion, one with coordinates that are the positions of all lower surfaces of S, the other with coordinates that are the positions of all upper surfaces of S. To find the time the empty rectangle meets a surface, we consider the minimum among $t_{n+1,1}$, $t_{n+1,2}$, …$t_{n=1,k}$ rather than maximum. Suppose the minimum among t_1, t_2, …t_n, t_{n+1}, t_{n+2} is t_p, then point p is the point that the empty rectangle expanding with the current speed will first meet, and t_p is the meeting time.

After we find the point that the empty rectangle first meets, we update the position and expansion speed of all surfaces of the empty rectangle and continue the expansion process until all surfaces meet points. The expanded empty rectangle is a MHR, we call it Expanding MHR (EMHR). The $EMHR_{ij}$ is a big MHR expanding from M_{ij}. The maximum among all $EMHR_{ij}$ can be considered a sufficient MHR.

Although we did not investigate all MHRs because the number of MHRs is $O(n^{2k-2})$, we do expand in each MHR at least $k(2k-1)$ times because there is at least one point on each surface of each MHR.

In the course of a GA optimization, the number of points visited will increase gradually. If we run the above process every time a new point is created, the efficiency will be very low because some expansions may be repeated many times unnecessarily. In the real program, the algorithm is as follows. Given a k-dimensional space, we first start with no point in S. We consider all the vertices of S as initial points, so the only middle point is the center of S, and the EMHR is the whole S. We add the EMHR to the set of EMHRs. Then the n points are added to S, one point at a time. At each insertion, only the EMHRs which contain the newly added point will be updated by doing expansion from its relative middle point, because EMHRs which do not contain the newly added point will not change. We do new expansions from the middle points between the newly added point and other points, and add their EMHRs to the set of EMHRs. At each time, the above algorithm will have same results with expansions from all middle points between any two points. The MHR we require is the biggest among all EMHRs.

We do not have to keep track of all EMHRs. First, if two middle points are very close, their EMHRs will strongly overlap; thus we can discard one of them. Second, if an EMHR is very small, we can discard it. For example, if the volume of space S is V, the number of points so far is N, then if the volume of an EMHR is less than V/N, we can discard it, because it will never be the biggest EMHR.

The detailed algorithm is also described using pseudo-code (Figure 3).

Algorithm FindBigEmptySpace

Begin

Set the corner vertices of space S as initial points.

Set the Linked list of EMHRs (LE) = NULL.

EMHR(M) = Expending(M) // *Expansion starts from M. At this step, M is the center of S, and EMHR(M) is the whole S*

Insert(LE, EMHR(M)) // *Add EMHR(M) to LE*

When a new point P is inserted into S,

Check LE

If ($EMHR_i$ contains P), update $EMHR_i$

From the mid point M_i between P and every other point i,

$EMHR(M_i) = Expending(M_i)$

$Insert(LE, EMHR(M_i))$

Check LE

If(Two EMHRs strongly overlap), delete one of them

If($EMHR_i$ is not big enough) delete $EMHR_i$ from LE // *The above two steps may be done once every several points.*

Return biggest EMHR and its center.

End

Expending(M)

Do (while not all dimensions of hyper-rectangle (HR) have met points)

For each point P_i,

$Time_i = getTime(P_i)$ // *the time on which the expanding HR meets with P_i*

$DimAndP_j = minimun(Time_i)$ // *the point that the expanding HR will meet first (along some dimension)*

Update the expansion speeds.

EndDo

Figure 3. The pseudo-code of the proposed algorithm

We use the above algorithm to find a sufficiently large MHR to periodically add new points to a GA population. One reason is that the calculation to find the exact biggest MHR is too high, while for our purpose, a sufficiently large MHR is good enough. Another reason is that, when we try to find the big empty space in a GA trace, we may prefer more squared space rather than a narrow space (Figure 4). The above expanding algorithm tends to find more squared space, because the shorter side will be met in the expansion earlier.

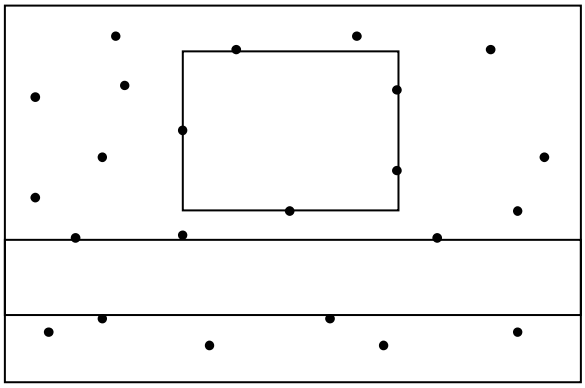


Figure 4. A more squared space is more interesting empty space

Analysis of complexity: for expanding from one point, the calculation to find the next point the expanding empty rectangle will meet is done in O(*nk*) because in order to find the time for one point when the expanding empty rectangle meet it, we have to check all of its coordinates, i.e. O(*k*), and we have to find the time for all the points (*n* of them). The expanding will stop when all surfaces meet points, so the total calculation for expanding from one point is O(nk^2). We do expansions from all middle points between any two points, so, the total calculation is O(n^3k^2).

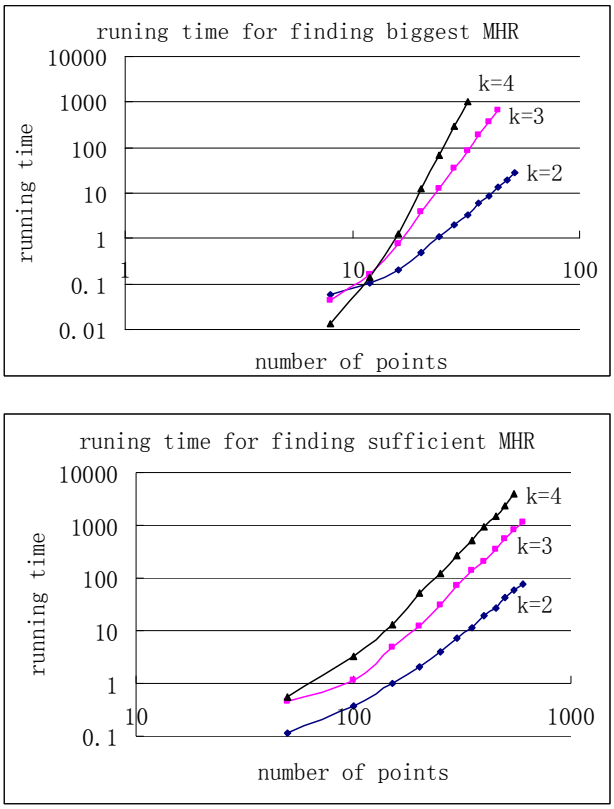


Figure 5. Running time for finding exact biggest MHR and sufficient big MHR (in logarithmic scale)

As pointed out earlier, we do not have to keep track of all EMHRs. If two middle points are too close, we only keep one of them and if an EMHR is too small, we discard it. The exact number of EMHRs to be discarded depends on the definitions of "too close" and "too small" but the real complexity may be lower than O(n^3k^2).

We compared the running time of the algorithm for finding the exact largest MHR and our proposed algorithm for finding a sufficiently large MHR with randomly generated data. For the sake of comparison, we find the biggest MHR by checking all MHRs, because we only try to show it is an exponential function. Figure 5 shows the results in terms of the actual CPU time.

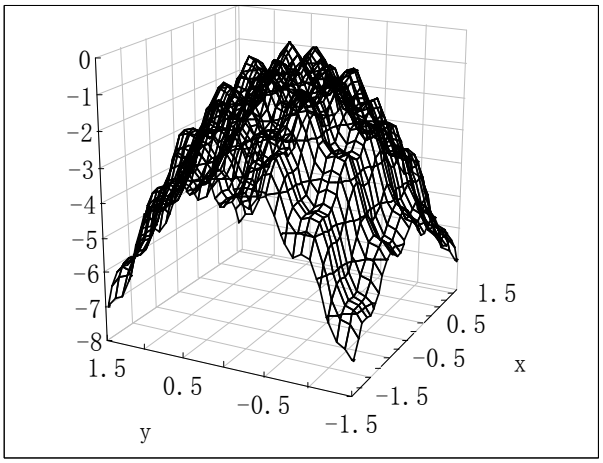
For finding the exact biggest MHR for k=4 and for only 30 points, the running time is several minutes! Furthermore, even for k=4, the running time increases at a very fast rate as n increases. When n increases, for example, from 16 to 32, the running time increases more than 700 times. From Figure 5, we can see that the slope of the curve increases when k increases. This means that the order is a function of k. On the other hand, the running time for finding a sufficiently large MHR using our algorithm, even for several hundred points, is a few seconds. When the number of points doubled, the running time is usually increased by 8 to 10 times and does not heavily depend on k. From Figure 5, we can see that the slope of the curves is nearly the same for different values of k, showing that our algorithm runs in polynomial rather than exponential time.

3. APPLICATION TO GA OPTIMIZATION

In this section, we apply our algorithm, inserting new individuals at big empty spaces, to some GA problems, to examine its effect on performance. We consider seven examples of continuous-parameter GA problems. Functions F1 and F2 are taken form previous studies[5, 6, 7]. F1 is symmetric while F2 is non-symmetric and both have many local optima. The functions are shown in Figures 6, 7. Note that we inverted the plots by adding a negative sign to the functions for the sake of clarity but all the functions used in this paper are minimized. The global optimum of F1 is at (0, 0) and the global optimum of F2 is at (0.375, 0.375). F3 is constructed based on F1 by copying a small range including the global optimum to another location, in order to see the effect of position of global minima. Similarly F4 is constructed based on F2. F5 is constructed using F1 and F2 with each function located at different ranges, so the whole function is not continuous. F6 is constructed based on F1, but expanding its dimensionality to four. F7 is constructed based on F6 in the same manner of constructing F3 from F1. The ranges of all functions are set to $-20 \le x_1, x_2 \le 20$. In the range, each function has thousands of local optima in addition to the global optimum.

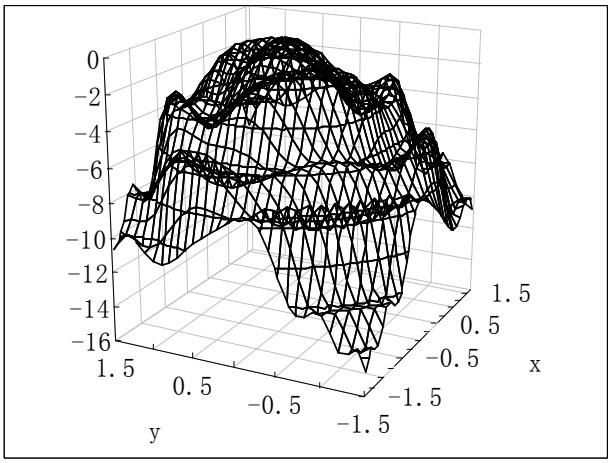
The initial population is generated randomly. Rank based selection is used as it is more robust in general. Then the population enters the following loop with fixed population size N.

Selection: part of the population ($r_s \cdot N$, where r_s<1, is the ratio of selection) are selected to propagate to the next generation and will also be allowed to produce offspring. The selection probability of an individual is $p_i = (rank_i)^{-c} / \sum_{i=1}^{N} (rank_i)^{-c}$, where c is a constant. The individual with highest fitness is guaranteed to be selected.



$$F_1(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

Figure 6. F1



$$F_2(x_1, x_2) =$$

$$\sum_{j=1}^{2}((x_1 - d_j)^2 + (x_1 - d_j)^2 - \cos(\pi((x_1 - d_j)^2 + (x_1 - d_j)^2)) + 1$$

Figure 7. F2 With $d_1 = 0$ and $d_2 = 0.75$

Crossover: part of the population ($r_c \cdot N$, where r_c<1, is the ratio of crossover) is created by crossover of two parents selected randomly from the above selected population. The crossover operator is $x_i = x_{i,1} + R*(x_{i,2} - x_{i,1})$, if $x_{i,2} > x_{i,1}$, $i = 1,2$, where *R* is a random number.

New individual insertions (by our method): the rest of the population ($(1 - r_s - r_c)N$) is created at big empty spaces, which are found according to our algorithm using all the points visited so far during the course of the optimization. The largest MHR is selected and a point is created at its center. This is repeated for the required number of individuals.

Mutation: each of the individuals in the selected population mutated with some probability P_m. This is done after the crossover step so if they took part in a crossover, it was with their original genes rather then their mutated forms. The mutation operator is non-uniform mutation [9] with $\Delta(t, y) = y \cdot R \cdot (1 - t/T)^b$, where *y* is the distance between the individual and the boundary, *R* is random number, *t* is the generation number, *T* is the maximum generation number, *b* is a system parameter determining the degree of non-uniformity.

In this paper, the parameters are set as r_s =50%, r_c=30% (in the comparisons when no insertions are done this becomes 50%), P_m=0.2, *c*=0.5, and *b*=2. We use a population size of 10 and terminate the process after 50 generations.

With the insertion of new individuals at big empty spaces, we expected more space to be covered. In other words, at the end of the GA process, the empty spaces should be smaller than without insertion. Indeed the experiments supported this expectation. Figure 8 shows the total individual distribution for F1. F2 has similar behavior.

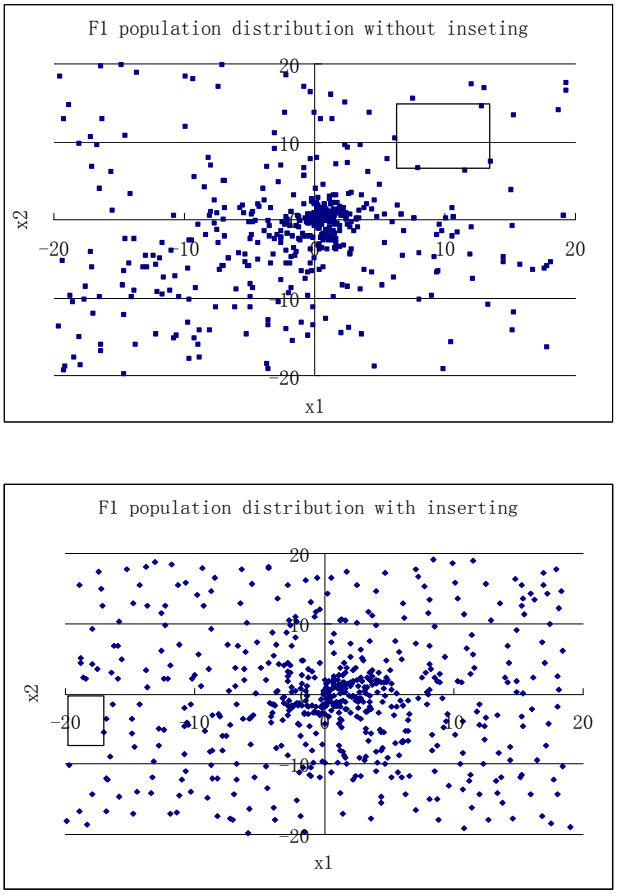


Figure 8. Population distribution for F1

Figure 8 shows that with the insertion of new individuals, the distribution of total points visited throughout the optimization is more uniform than that without insertion. After 50 generations, without inserting individuals at big empty spaces, the biggest empty square is 7.4×7.4, while with individual insertion; the

biggest empty square is 3.0×3.0. Thus more space is searched by using our algorithm.

One way to measure the success of a GA run is by checking if there are individuals within a tight ε-neighborhood of the optimum. In this paper, we consider the GA run to be successful if the individual with highest fitness is within the basin of attraction of the global optimum. The motivation behind this is to reduce the computation time. If the individual with highest fitness is close to the optimum and in its basin of attraction, usually it will go to the optimum gradually. The basin of attraction of F1 is $x_1^2/0.35+x_2^2/0.3<1.0$, and for F2, it is $(x_1+x_2-0.75)^2/0.4+(x_1-x_2)^2/0.5<1.0$.

We test the performance of our algorithm by checking how many runs are successful among 200 runs. The results are shown in Figures 9 and 10.

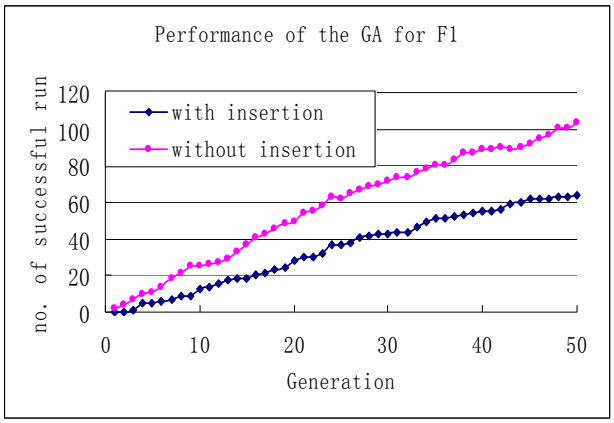


Figure 9 Performance of the algorithm for F1

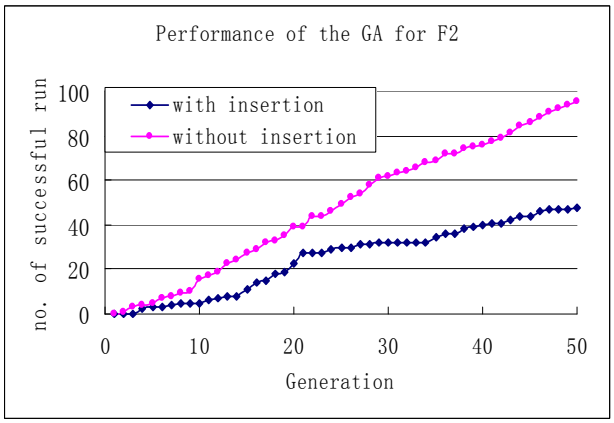


Figure 10 Performance of the algorithm for F2

From figure 9 and figure 10, we see that our method actually degrades the performance for F1 and F2. Although this may seem surprising it is understandable. Although F1 and F2 have many local optima, their main shapes are simple. Take F1 for example, its main shape is determined by the first and second terms of F1, $x_1^2+x_2^2$, therefore, the values near the center (0,0) of the space is small while the values near the corner and the boundary of the space is big. The local change is caused by the third and fourth terms of F1, $-0.3\cos(3\pi x_1)-0.4\cos(4\pi x_2)$. With the regular process of evolution, the population will concentrate near the center gradually because of the effect of selection, crossover and mutation. The range near the boundary of the space may have big empty spaces. So if we insert some individuals in big empty spaces, with high probability they will be near the boundary rather than the center of the space. This will explore more space, however, for F1, the global optimum is at (0, 0), the insertion has no contribution to success and will degrade the performance.

However, in practice, we do not know the location of the global optimum, and the function may have any shape. So it is necessary to explore more space in order to avoid missing some space where the global optimum lies.

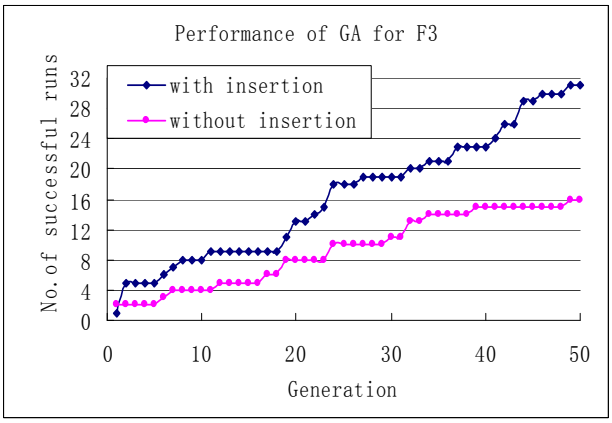


Figure 11. Performance of the algorithm for F3

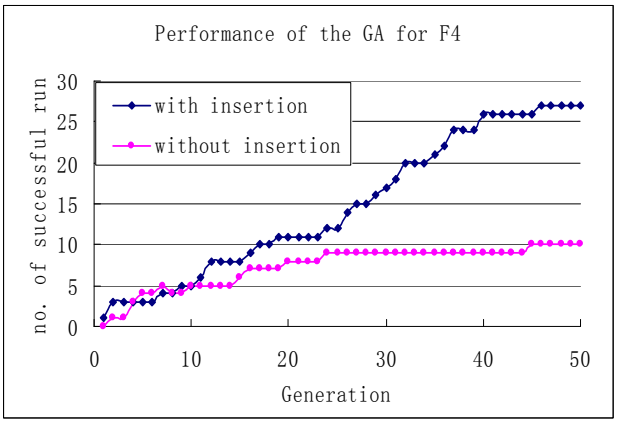


Figure 12. Performance of the algorithm for F4

To demonstrate this, we constructed F3 and F4 based on F1 and F2 respectively. To create F3, we copied a small range which contains the global optimum to another location, for example, (9-11, 9-11), and added a small value to the original function to

make the original global minimum a local minimum. The expression of F3 is as follows:

$$F_3(x_1, x_2) =$$
$$x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 + 1.0$$

where $|x_i - 10| \geq 1, i = 1,2$

$$F_3(x_1, x_2) = (x_1 - 10)^2 + 2(x_2 - 10)^2$$
$$- 0.3\cos(3\pi(x_1 - 10)) - 0.4\cos(4\pi(x_2 - 10)) + 0.7$$

where $|x_i - 10| \leq 1, i = 1,2$

F4 is constructed similarly.

We then checked the performance of the algorithm for F3 and F4. The results for F3 and F4 are shown in Figures 11 and 12 respectively. The figures show that for F3 and F4 the ratio of successful runs with individual insertions in big empty spaces is higher. If the global optimum is not in the range where most values are good, inserting new individuals will increase the probability of finding it.

We next considered a discontinuous function**.** F5 is constructed by putting F1 at the range (7-13, 7-13) with its origin at (10, 10) and the range ((-7)-(-13), (-7)-(-13)) with its origin at (-10, -10), F2 at the range ((-7)-(-13), 7-13) with its origin at (-10, 10) and the range ((7-13, (-7)-(-13)) with its origin at (10, -10). The function surface is shown in Figure 13. In its full range, the function is discontinuous. We further added to each continuous region a different constant (0, 4, 6, 12 respectively) so that the whole function has only one global optimum.

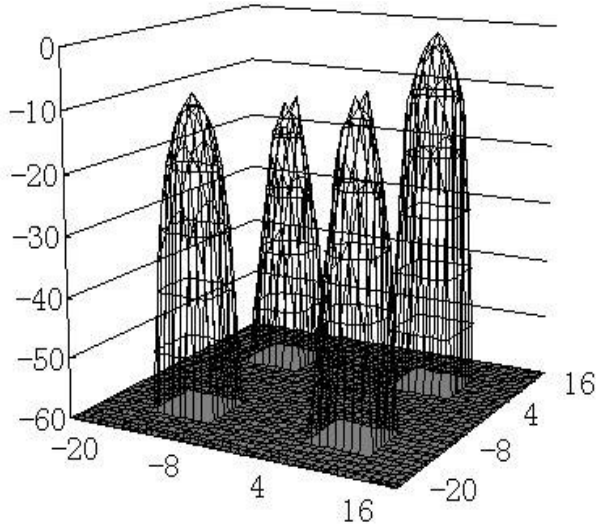


Figure 13. F5

Figure 14 shows the performance of the algorithm for F5. The figure shows that for F5 the ratio of successful runs with and without insertion of new individuals in big empty space is nearly the same. It means the contribution of inserting new individual in empty space is nearly the same as the cost of inserting.

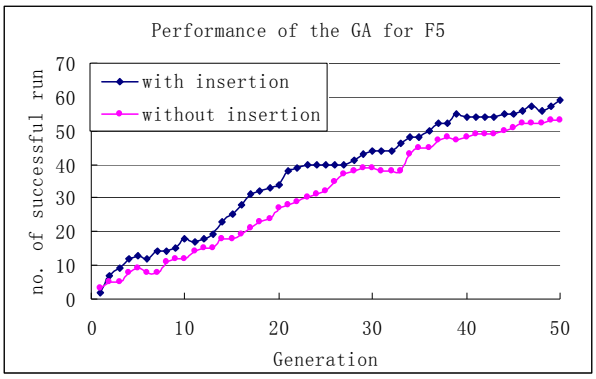


Figure 14. Performance of the algorithm for F5

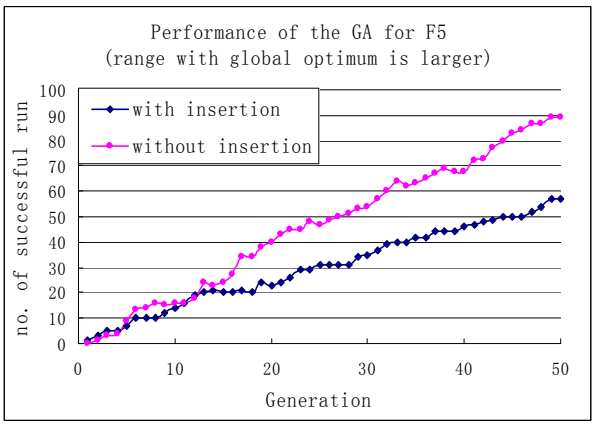


Figure 15. Performance of the algorithm for F5

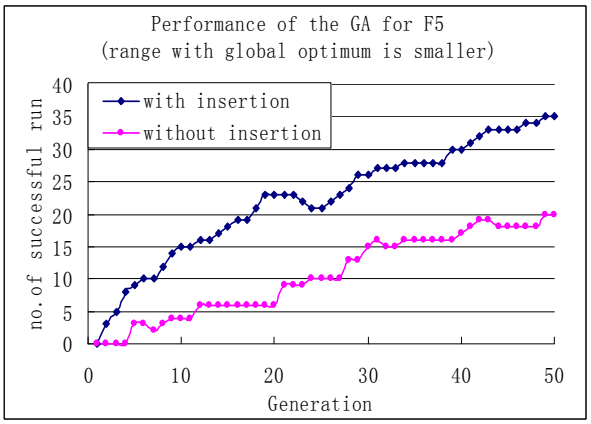


Figure 16. Performance of the algorithm for F5

We hypothesized that increasing the range of the peak containing the global optimum will degrade the contribution of individual insertions while decreasing that range will have the opposite effect. To test this we first increased the range with the global optimum from 6×6 to 18×18, and repeated the experiments. Figure 15 shows the results. We then decreased that range from 6×6 to 2×2 and the results are shown in Figure 16. The results clearly support our hypothesis. (Figure 15, 16)

All of the functions discussed so far were two-dimentional. Although we expected that the performance of the algorithm should be the same, we considered functions with higher dimension. F6 is constructed from F1 by expanding its dimension to four as follows,

$$F_6(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(\pi x_1) - 0.4\cos(1.5\pi x_2) + x_3^2 + 2x_4^2 - 0.3\cos(\pi x_3) - 0.4\cos(1.5\pi x_4) + 1.4$$

Its basin is $x_1^2/3.1 + x_2^2/2.1 x_1^2 + x_3^2/3.1 + x_4^2/2.1 < 1.0$. F7 is constructed from F6 in the same manner we constructed F3 from F1, also for the same reason. We used a population size of 20 and terminated the process after 50 generations in this case. We tested the performance of our algorithm by checking how many runs are successful among 200 runs. The results for F6 and F7 are shown in Figures 17 and 18 respectively.

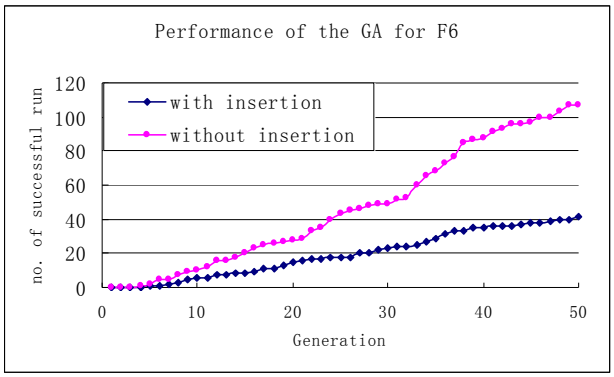


Figure 17. Performance of the algorithm for F6

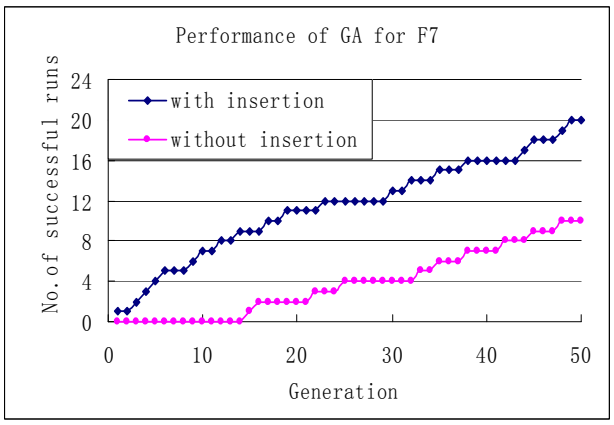


Figure 18. Performance of the algorithm for F7

From Figure 17 and 18, we can find that the performance for F6 and F7 is similar to F1 and F3 respectively, showing that our algorithm has same behavior for functions with higher dimension.

4. CONCLUSION

In this paper, we propose a modification for the classical Darwinian evolution metaphor commonly used in evolutionary optimization by periodically inserting new individuals at big empty spaces. To efficiently do this we propose an algorithm to find sufficiently large empty hyper-rectangles with polynomial time complexity. We show that it is efficient and scalable.

We conducted experiments to check its performance using several functions. The experimental results demonstrate that more space will indeed be searched by inserting new individuals in big empty spaces. Even when the global optimum is not located at the range where all or most points have good fitness, the GA with insertion of new individuals will have a higher probability of finding the global optimum. This is particularly useful in discontinuous and multi-modal optimization domains. The proposed method also provides a potential tool for measuring the reliability of a GA search (or any other search method for that matter) based on the size of the gaps in the search space. Further research is planned to identify such measures.

REFERENCES

[1] Ku, L., Liu, B., and Hsu, W. *Discovering Large Empty Maximal Hyper-Rectangle in Multi-Dimensional Space*. Technical Report, Department of Information Systems and Computer Science (DCOMP), National University of Singapore, 1997.

[2] Liu, B., Ku, L., and Hsu, W. *Discovering Interesting Holes in Data.* In Proceedings of IJCAI, pages930-935, Nagoya, Japan, 1997.

[3] Edmonds, J., Cryz, J., Liang, D., and Miller, R. J. *Mining for Empty Rectangles in Large Data Sets.* In Proceedings of Intl Conf on Database Theory (ICDT), pages 174--188, 2001.

[4] Liu, B., Wang, K., Mun, L.F., and Qi, X.Z. *Using Decision Tree Induction for Discovering Holes in Data.* In 5th pacific Rim International Conference on artificial Intelligence, Pages182-193, 1998.

[5] Bohchevshy, I.O., Johnson, M.E., and Stein, M.L. *Generalized Simulated Annealing for Function Optimization.* Technometrics 28(3), PP. 209-218, 1986.

[6] Fogel, D.B. *Evolutionary computation: toward a new philosophy of machine intelligence.* (Institute of Electrical and Electronics Engineers, Inc.).

[7] Ballester, P.J. and Carter J.N. *Real-parameter Genetic Algorithms for Finding Multiple optimal Solution in Multi-modal Optimization.* GECCO 2003, LNCS2723, PP. 706-717, 2003.

[8] Mahfoud, S. *A comparison of parallel and sequential Niching methods*. 6th Int. Conf. on Genetic Algorithms, pages 136-143. Morgan--Kaufmann,1996.

[9] Michalewics, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Third Edition, Springer, 1996.