Building Block Filtering Genetic Algorithm

Jun Lu, Boqin Feng and Bo Li

Xi'an Jiaotong University

Abstract. A Building Block Filtering Genetic Algorithm(bbf-GA) is proposed which introduces building block candidates filtering and exploiting to improve traditional GA**.** Various recognition functions are designed and tested by analyzing the features of building blocks during the evolution of GA search for symmetrical TSP, and one of them is adopted to filter building block candidates. A position representation for TSP and relevant bbf-based genetic operators are designed to exploit the building block candidates. The proposed TSP specialized position representation can decrease the computational workload of bbf-GA, such as edge comparison, computation of individual similarity, abstraction of uniform edge, and operations in bbf-based genetic operators. Experimental results show that comparing with traditional GA, Building Block Filtering Genetic Algorithm can improve the efficiency of search remarkably by reducing unnecessary search in GA.

1 Introduction

The GA's search strategy is commonly described by the pattern theorem and building block hypothesis. The building block hypothesis (Holland1975; Goldbery 1989) [1] states that the GA works well when short, low-order, highly-fit schemas recombine to form even more highly fit higher-order schemas. The ability to produce fitter and fitter partial solutions by combining building blocks is believed to be the primary source of the GA's search power, thus improving the ability of GA to exploit known building blocks in limited populations and to explore new building blocks at the same time is essential to improve the search of GA.

Numerous researchers have studied on defining and exploiting building block. Forest and Mitchell[2] designed a class of fitness landscapes (the "Royal Road" function) to measure the effects of genetic operators on building block in binary encoding mechanism. Wu *et al.* [3] compared two different GA representation schemes with a floating representation scheme and examine the differences in building block dynamics. Kemenade[4] compared and identified building block by calculating the difference of the fitness value caused by the change of allele in binary coding, and utilized it in the proposed three-stage GA.

Zhou peng *et al.* [5] applied reduction mechanism to find uniform partial solutions from local optimal solutions generated by heuristic method, then reduced the scale of the instance by multi-reduction algorithm, finally the solution of the original problem could be reverted after iterative operations. Schneider *et al.* [6] proposed an efficient parallel method to reduce the instance of TSP to a smaller one by finding backbones which are actual uniform partial solutions from local optimal solutions and eliminating them from original problem to get even better solutions in a very short time and a few observables of interest corresponding to this parallel approach.

These research works inspire us to find the way to guide the search of GA by filtering building block from similar parts among populations. In this paper, the search of instance of symmetrical Traveling Salesman Problem (TSP) is used to evaluate various recognition mechanisms, and one of them is adopted and a Building Block Filtering Genetic Algorithm (bbf-GA) is proposed. The structure of this paper is as follows. In next Section we introduce the chromosome representation for TSP (position representation) that we proposed, as well as compare and analyse various recogniztion functions and their filtering results. After that, the bbf-GA is described detailed in Section 3 and experimental results are presented in Sections 4. Finally, conclusions come in Section 5.

2 Recognition of Building Block

The Traveling Salesman Problem has been in the focus of studies for many years. In order to investigate the features of building block during the typical evaluation of a GA search, several testing instances in TSPLIB[7] are chosen.

As to TSP, Building blocks can be taken as the "best" edges. For a certain node, the "best" edge is not the shortest edge that takes it as vertex (greedy algorithm can hardly find the optimal solution), but the edge that synthesized with other "best" edge to make the shortest full-path. Thus, it is impossible to determine an edge is good or not just by the comparison of edges' length. Calculating the difference of the full-path length when the connected edge of a vertex is changing could identify best edges. However, the computational work of this is as hard as that of the solution search itself.

In the first place, we use traditional GA to solve TSP. The known optimal solution is inputted at the beginning of algorithm and all edges in the solution are taken as "best" edges, which are the building blocks. Then we look into the distribution and change of building blocks in each individual during running process of algorithm to find the way to filter building blocks. In order to improve the calculating efficiency, position representation is designed.

2.1 Position Representation

There have been many different representations used to solve the TSP problem using GA [8] such as ordinal representation(Grefenstette 1985), adjacency representation (Grefenstette 1985), metrix representation(Fox and McMahon 1992), edge representation (Homaifar and Guan 1993), and path representation etc. The most natural representation is path representation. For instance, path (1-3-2-6-5-4-1) can be represented as (1,3,2,6,5,4) directly. However in this representation, the individual has a cycle topology. The meaning of genic segment just shows the relationship between a node and its previous and next node, but is independent of its position in chromosome. Different individual, such as the four shown in the left column of Table 1, may represent the same path.

In algorithms based on path representation, it takes much time in recognizing the same edges in two individuals. Thus, we propose a position representation inspired by adjacency representation (Grefenstette 1985). In position representation, each individual is composed of two parts: right adjacency (RA) and left adjacency (LA), which means the subsequence node and previous node of the node that represented by genic position (in adjacency representation individuals only have right adjacency). E.g. path(1-3-2-6-5-4-1)can be represented as(RA)(3 6 2 1 4 5)and(LA)(4 3 1 5 6 2) where the 3rd position in(RA)is 2 which means edge (3-2), and the 2nd position in(LA) is 3 which also means edge (3-2).

The position representation of the four individuals is depicted in the right column of Table 1. For individuals in position representation, it only needs two operations to judge whether an edge in them is the same or not. E.g. the following comparison is used to judge whether edge (4-5) in individual 2 exists in individual 3 or not.

if(individual2.RA[4]==individual3.RA[4] || individual2.RA[4]==individual3.LA[4])

It's easy to find from Table 1 that six edges of those four individuals are all the same. Although position representation requires more memory, it reduces the computational work for the comparison of allele among individuals. What is more, it benefits bbf-based genetic operators depicted in Section 3.

Table 1. The comparison of path representation and position representation

position / individual	path representation						position representation					
	1	2	3	4	5	6	1	2	3	4	5	6
1	1	3	2	6	5	4	3	6	2	1	4	5
							4	3	1	5	6	2
2	1	4	5	6	2	3	4	3	1	5	6	2
							3	6	2	1	4	5
3	2	6	5	4	1	3	3	6	2	1	4	5
							4	3	1	5	6	2
4	6	2	3	1	4	5	4	3	1	5	6	2
							3	6	2	1	4	5

2.2 Recognition function of building block

Firstly, Simple Genetic Algorithm (SGA) is used to investigate the distribution of building blocks in the evolution of a GA search for Ludwig's drilling problem 280. Unfortunately, the results in early stage are depressed. When the population size is set to 400, only 30 edges are as same as those in the optimal solution even iterate to the 1000th generation. In order to reduce runtime, new individuals in each generation are optimized by 2-opt algorithm in probability Ph (that is so-called memetic algorithm). 2-opt algorithm can eliminate path crossover effectively, but cannot guarantee to find optimal solution [8]. The amount of building blocks among the population in the 20th generation during the evolution of a search is shown in Fig. 1 (where x-axis is the serial number of buiding blocks, which are the edges in the optimal solution and y-axis is the relevant appearing frequency in the population). We can see that some building blocks have a quite high appearing frequency at the beginning of evolution. Thus, it is possible to recognize building block by certain statistic methods to avoid useless and repeatable search for known building blocks. In order to find the relationship between the amount of building blocks among individuals and the length of the path of individuals, we analyze these two features of individuals. The results are shown in Fig. 2.

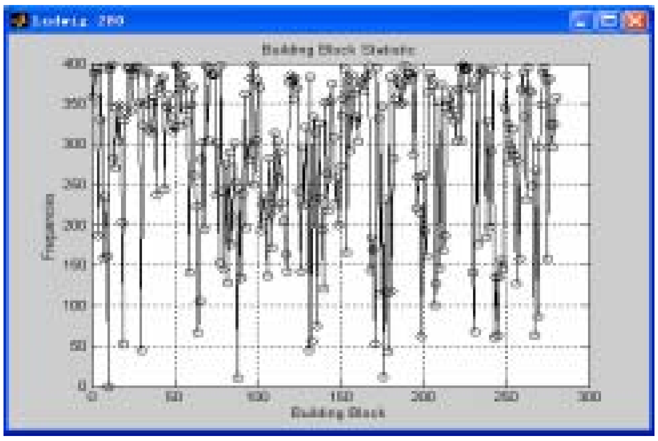

Fig. 1. Statistic of appearing frequencies for building blocks

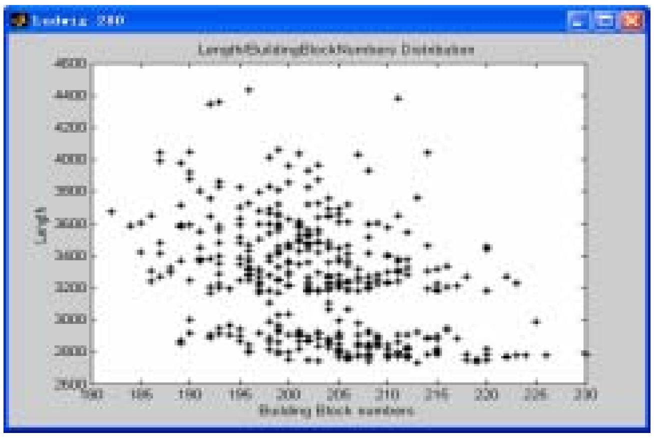

Fig. 2. Distribution of path length / build blocks in individuls

From Fig. 2 we can see that the amounts of building blocks among individuals in populations are ranged from 180 to 230, and the amount of building blocks and the path length do not have linear relationship. The individuals that have longer path length may include more building blocks on the contrary. As to TSP problem, an individual that has a worse edge achieves a longer path length no matter how many good edges it has.

Six statistic functions are designed to recognize building block from populations, in addition the recognizing effects are evaluated.

Let N be the size of population, f_i be the fitness of individual (fitness function is $f(\pi) = 76.5 \times L\sqrt{N} / D\pi$,where L means the side of the smallest square which can contain all the cities, N is the number of cities and $D\pi$ is the length of the path in the current permutation.), $\bar{f}$ be the average of fitness, n be the amount of individuals whose fitness are better than $\bar{f}$, E_i be the set of edges in individual i, $E_i^e = \begin{cases} 1 & e \in Ei \\ 0 & e \notin Ei \end{cases}$,then six recognition functions are designed as follows:

$$F_e^1 = \sum_{i=1}^{N} E_i^e / \mathrm{N} \quad \textbf{(1)}$$

$$F_e^2 = \sum_{i}^{n} E_i^e / \mathrm{n} \quad \textbf{(2)}$$

$$F_e^3 = \sum_{i}^{N} E_i^e * \left(\frac{fi}{\sum_{i}^{N} fi}\right) \quad \textbf{(3)}$$

$$F_e^4 = \sum_{i}^{n} E_i^e * \left(\frac{fi}{\sum_{i}^{n} fi}\right) \quad \textbf{(4)}$$

$$F_e^5 = \sum_{i}^{n} E_i^e * c(1-c)^{i-1} \qquad c \in [0.01, 0.05] \quad \textbf{(5)}$$

$$F_e^6 = \sum_{i}^{b} E_i^e / \mathrm{b} \qquad \mathrm{b} \in [2, \ \mathrm{N}] \quad \textbf{(6)}$$

2.3 Analysis of recognition functions

Probabilities are calculated by recognition functions for all edges in population in the generations during the evolution. Those edges whose probability exceed threshold $P_{\text{threshold}}$ are considered as building block candidates, and let T_C represents the amount of them. Compare building block candidates and building blocks (edges in known optimal solution) to find false building blocks (that are candidates who are not true building blocks), and let F_C represents the amount of these false building blocks. Then, the true recognition rate is: Rate=$(T_c\text{-}F_c)/T_c$.

When $P_{\text{threshold}}$ is set to 0.98,the comparison of the recognition results of function F^1 to F^5 is shown in Fig. 3.

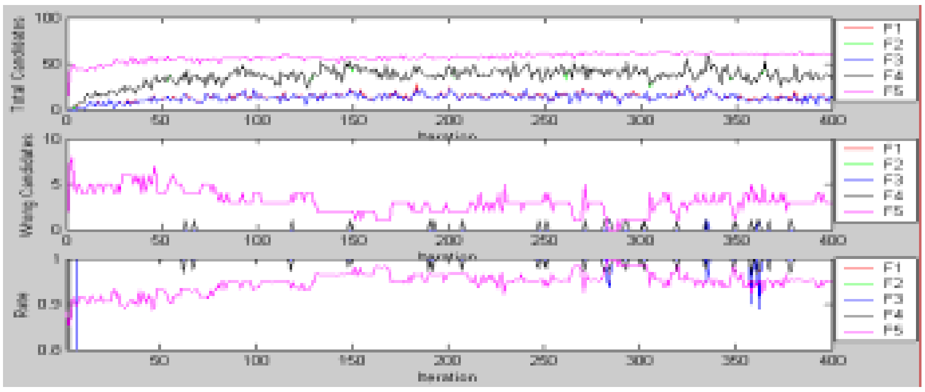

Fig. 3. The comparison of different functions ($P_{threshold}$ =0.98)

By analyzing a great deal of experimental data, we found that the recognition ability of function F^4 is the best, while F^5 is the worst whose false rate is the highest. The false rate of F^3 is the sub-highest, while the rest are similar. The comparison of recognition results with different thresholds of function F^4 is shown in Fig. 4.

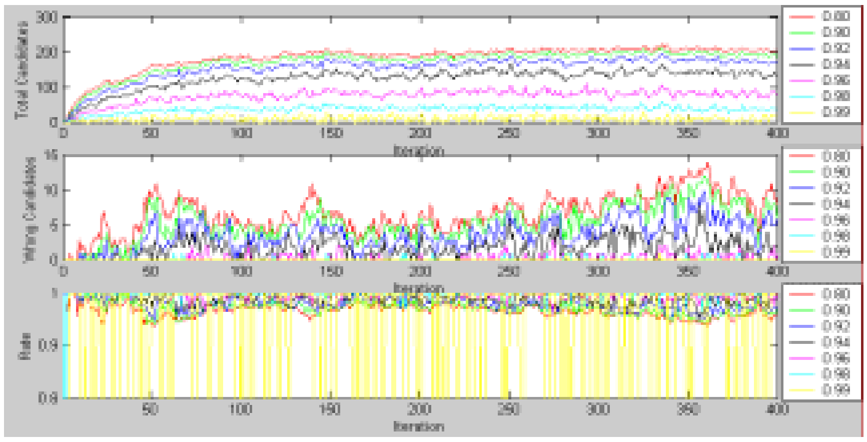

Fig. 4. Comparison of different thresholds for F^4 ($P_{threshold}$ is ranged from 0.80 to 0.99)

From Fig. 4 we can see that when the threshold is close to 1, it is hardly to find building block candidates, and when the threshold is lower than 0.96, the false rate is rather high.

The recognition results of function F^6 are shown in Fig. 5. When b, which means the number of the best individuals in population, is set from 2^1 to 2^7, and the threshold is 1. From Fig. 5 we can see that the smaller the number of statistic individuals is, the higher the false rate is. When b is set to 8 and at the 190th generation, although the true rate is 0.95, due to the larger number of building block candidates, the false building blocks are over 20. From the comparison of all mechanisms, we find that function F^4($P_{threshold}$ =0.98) and F^6($P_{threshold}$ =1 , b= 2^7) , the comparison of which is shown in Fig. 6, are better than others. As a result, we take F^6 ($P_{threshold}$ =1 , b= 2^7) as filtering function for that it needs less computational work.

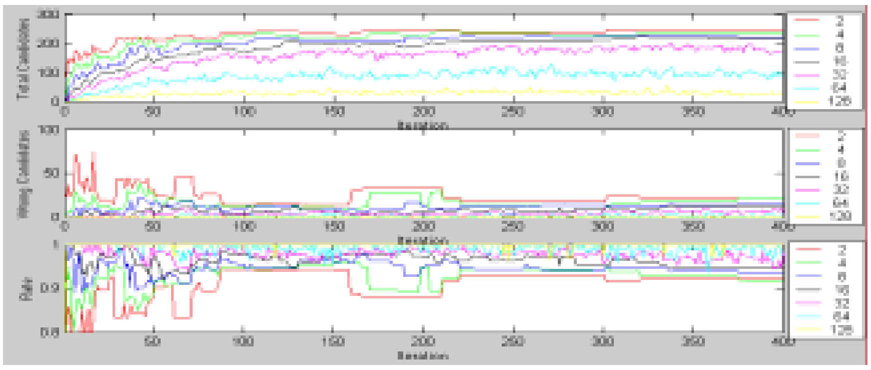

Fig. 5. Comparison of F^6 ($P_{threshold}$ =1) when b is set from 2^1 to 2^7

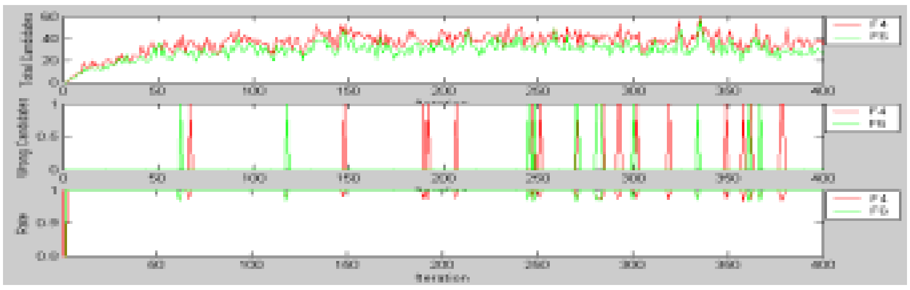

Fig. 6. Comparison of F^4 ($P_{threshold}$ =0.98) and F^6($P_{threshold}$ =1, b=2^7)

The comparison of different functions show that statistic based method can recognize building block in a high probability. But false recognition will appear no matter which function is used. In this case, eliminating the edges of building block candidates from original problem by reduction mechanism to reduce the scale of the problem will probably cause false reduction, which will result to failure to find the optimal solution.

However, the average probability of finding building blocks by the best functions is 0.98. These building block candidates can be preserved during the evolution and make the search of GA more effective.

3 Building Block Filtering Genetic Algorithm (bbf-GA)

3.1 The bbf-GA

In order to exploit the building blocks filtered, we propose bbf-GA as following:

```
Initial the parameters of GA;
Create initial population P(t) randomly;
Improve chromosomes by 2-opt algorithm in probability Ph;
Evaluate P(t);
While (not meeting the terminal condition){
    Calculate and abstract building block candidates from individuals of P(t);
    Implement crossover operation to P(t) in probability of Pc*(1-Pb) to get C1(t);
    Implement bbf-based crossover operation to P(t) in probability of Pc*Pb to get C2(t);
    Reproduce P(t) in probability of (1-Pc) to get C3(t);
    C(t) =C1(t)+C2(t)+C3(t);
    Implement mutation operation to C(t) in probability of Pm*(1-Pb);
    Implement bbf-based mutation operation to C(t) in probability of Pm*Pb;
    Implement 2-opt algorithm to C(t) in probability of Ph;
    Evaluate C(t);
    Generate P(t+1) based on the optimum individuals in P(t) and C(t);
    t=t+1;
}
```

In our algorithm the position representation is adopted, and each individual is represented as right adjacency (RA) and left adjacency (LA). Crossover operator adopts Ordered Crossover Operator[8] method proposed by Davis in 1985, which constructs an offspring by choosing a sub tour of one parent and preserving the relative order of cities of the other parent. Mutation operator adopts random multipoint mutation. The parameters of GA are: Pc(crossover probability), Pm(mutation probability), Ph(2-opt optimization probability) , N(the size of population).

In order to filter building block candidates, the recognition function is implemented to algorithm. Building block candidates are also represented as right adjacency (RA) and left adjacency (LA), where the position of non-building block is represented as -1. In addition, bbf-based crossover operator and mutation operator is designed to exploit building block candidates. The bbf-based genetic operators are used in probability of Pb, while normal operators are used in 1-Pb. From later experiments we can see that the search will cause rapid premature convergence when Pb is big enough. Due to the existence of false genic segment in building block candidates, Pb shouldn’t be too big. When Pb is set to 0, algorithm is equal to traditional GA actually.

3.2 Bbf-based genetic operators

In this paper, traditional genetic operators are mended to exploit building block candidates in individuals. The bbf-based crossover operator is depicted as following.

Input: parents P1,P2; building block candidates B
output: offspring O1
operation: choosing edges in parent P2 that either are between random position s1 to s2 or belong to building block candidates and the rest edges from parent P1 to generate offspring O1. The loss edges are generated randomly.

Algorithm description:

```
(1)  iCount=0;                  // count of passed node
(2)  O1[ ]=$\phi$                     // path of offspring
(3)  Set edges in P2 that neither are between position s1 to s2 nor belong to building
       block candidates to -1;
(4)  iCur=rand(N);    // begin with random node
(5)  while(iCount <N){    // analyze for each gene
(6)       if(P2.RA[iCur]!=-1){    / /P2 has right adjacency
(7)             iNext= P2.RA[iCur];
(8)             P2.RA[iCur]=-1;                 // segment can be used only once
(9)             P2.LA[iNext]=-1;
(10)      }
(11)      else if(P2.LA [iCur]!=-1){    //P2 has left adjacency
(12)            iNext= P2.LA[iCur];
(13)            P2.LA[iCur]=-1;                 //segment can be used only once
(14)            P2.RA[iNext]=-1;
(15)      }
(16)      else{        // edges that not belong to P2 are selected from P1
(17)            iNext= P1.RA[iCur];                    // select RA first
(18)            if(iNext$\in$O1[ ] || (P2.RA [iNext]!=-1 && P2.LA[iNext]!=-1) ){
(19)            // next node has been used, or is the vertex of two edges in P2
(20)                  iNext= P1.LA[iCur];                // select LA then
(21)                  if(iNext$\in$O1[ ] || (P2.RA[iNext]!=-1 && P2.LA[iNext]!=-1) )
(22)                  // next node has been used, or is the vertex of two edges in P2
(23)                  iNext=random usable node
(24)            }
(25)      }
(26)      O1[ ]+=iNext;
(27)      iCur=iNext;
(28) }
```

The purpose of step 3 in above algorithm is to eliminate neither selected edges nor building block candidates in parent P2 to generate offspring O1 cooperated with parent P1. The starting node is generated randomly, and next node is selected from P1 or P2 each time. To generate next node, the RA and LA (which means two edges representing different direction from current node) of parent P2 are selected firstly. If no usable node found, the RA and LA of parent P1 is used instead. If no valid node is found in both P1 and P2, next node is generated randomly (in step 23). In this case, it is necessary to estimate whether next node is used (which may cause cycle), or is the vertex of two edges in P2 (which may cause the loss of an edge in P2).

The algorithm preserves all edges that either in selected zone or belong to building block candidates in parent P2, as well as some edges of parent P1. From the process of algorithm we can see the advantage of position representation, that is, preserving certain edges (non-continuous edges are possible) effectively without much computational work.

The way to mend mutation operator is simple. After generating a node that should be mutated randomly, the RA and LA of this node are checked whether they are belonging to building block candidates. If so, generate a new node to avoid losing of these edges.

The bbf-based operators can preserve edges that belong to building block candidates in parents and avoid damage, reform and comparison to these nodes. Thus, the search is focus on the rest edges, which reduce unnecessary stochastic search and improve search efficiency of GA.

4 Experiment and Analysis

In our experiments, we set parameter values as followings: population size N=400, number of generations=400, crossover probability Pc=0.80, mutation probability Pm=0.03, local optimization (2-opt algorithm) probability ph=0.3. Traditional GA and bbf-GA (Pb=0.35, and bbf-based genetic operators are employed after 100th generation) are implemented 100 times each and the results are shown in Table 2.

Table 2. Comparison of Traditional GA and bbf- GA

Algorithm	Times of finding optimal solution	Average generations of finding optimal solution	variance of solution
Traditional GA	4	309	69.14505583
bbf-GA	28	213	29.08194754

From Table 2 we can see that the possibility of finding the optimal solution (2586.7696) by bbf-GA is increased remarkably, and the fluctuating of result is reduced. Another important data in our experiment is that a suboptimal solution (2587.8088) is found 48 times. We consider this as a result of that the false building block candidates cause convergence to suboptimal in high probability. The results also show that traditional GA has higher fluctuating and randomicity than bbf-GA. Fig. 7 is the distribution of building block candidates in one of these results (the red edges represent building block candidates filtered).

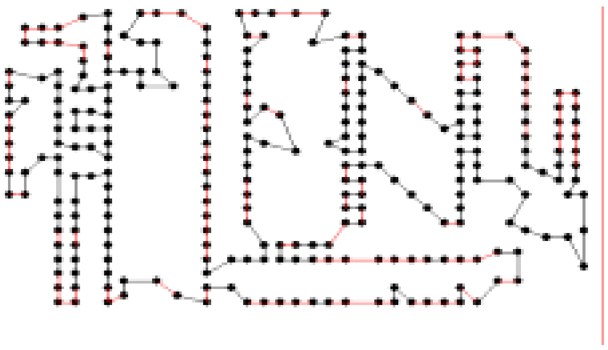

Fig. 7. Distribution of building block candidates

The bbf-GA cannot be improved by increasing Pb simply. When Pb is up to 0.85, the result is no better than traditional GA. By analyzing the recognizing process of building block candidates we can see that when Pb is big enough, building block candidates (including both true and false candidates) diffuse among population. It is clear to see from Fig. 8 that the amount of building block candidates and false candidates are increased significantly as well as recognition accuracy is reduced from the 100th generation when bbf-based genetic operators are employed.

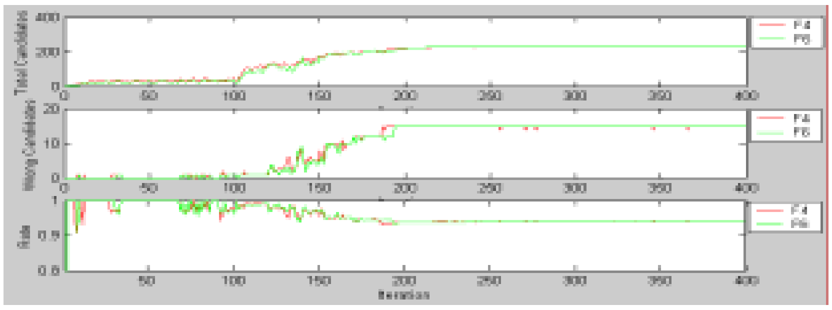

Fig. 8. Comparison of candidate building block recognition in evaluation process (Pb=0.85)

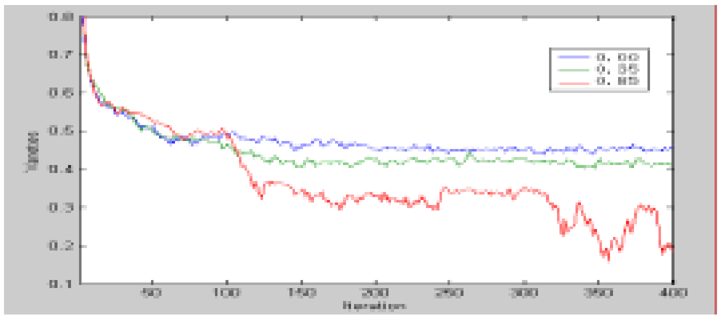

Fig. 9. Diversity comparison in evoluation process

By observing the individuals among population, we find that when Pb is big enough, the diversity of population will be destroyed, which leads to premature convergence. Fig. 9 shows the comparison of the average information entropy of different generations, where when Pb=0 (traditional GA), the diversity is the highest; when Pb=0.35, the diversity is decreasing slightly; When Pb=0.85, the diversity is decreased significantly from the 100th generation.

5 Conclusion

In order to reduce useless search of GA on parts that are already optimal and make the search more effective, a mechanism that uses statistic function to filter building block candidates in the evolution of GA search is proposed. By testing the recognition effect of 6 statistic functions, a bbf-GA is proposed, including the filtering of building blocks and the bbf-based genetic operators. The experimental results show that the recognition and utilization of building blocks can improve the efficiency of search significantly. The comparison between traditional GA and bbf-GA makes it clearly that local searching algorithm(2-opt) can generate a large amount of high quality partial solutions rapidly, as well as recognizing and preserving these partial solutions during the evolution of GA can take advantage of the parallel search ability of GA. In addition, position representation is proposed, which decreases the computational workload of bbf-GA, such as edge comparison, computation of individual similarity, abstraction of uniform edge, and operations in bbf-based genetic operators (especially for the exploitation of non-continuous edges).

References

1. J. H. Holland. Adaptation in Natural and Artificial System. Ann Arbor: The University of Michigan Press, 1975.
2. S. Forrest, M. Mitchell. Relative building-block fitness and the building-block hypothesis. In Foundations of Genetic Algorithms 2, 1992, pp.109-126.
3. Annie S. Wu and Kenneth A. De Jong An examination of building block dynamics in different representations. In the Proceedings of the 1999 Congress on Evolutionary Computation, 1999 pp. 715-721.
4. Cees H. M. van Kemenade. Building block filtering and mixing. Report SEN //Centrum voor Wiskunde en Informatica, Software Engineering, 1998.
5. Zhou Peng, Zhou Zhi, Chen Guoliang et al. A multilevel Reduction Algorithm to TSP. Journal of Software.2003,14(1):35-42.
6. J.Schneider, et al, Searching for backbones-an efficient parallel algorithm for the traveling salesman problem, Computer Physics Communications 96, 1996, pp.173-188.
7. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
8. P.Larranaga, C.M.H.Kuijpers, et al. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review, 13(2), 1999, pp. 129-170.