# Exploiting Building Blocks in Hard Problems with Modified Compact Genetic Algorithm

Kamonluk Suksen
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
6071401321@student.chula.ac.th

Prabhas Chongstitvatana
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
prabhas.c@chula.ac.th

## ABSTRACT

In Evolutionary Computation, good substructures that are combined into good solutions are called Building Blocks. In this context, Building Blocks are common structure of high-quality solutions. This paper describes an algorithm that exploits building blocks (BBs) with Compact Genetic Algorithm (cGA) in order to solve difficult optimization problems. cGA is a second generation of Genetic Algorithm that contains the model of the solution in terms of probability vectors representing probability density function of solutions. The main idea is to update the probability vectors as a group of bits that represents BBs thus avoiding the disruption of the BBs. A comparison to the plain cGA is made. The experiments are carried out on Trap-function and TSP problems. The results show the effect of this heuristic. It is most effective when the problem instants have common structures that can be identify as Building Blocks.

## KEYWORDS

Genetic algorithm, genetic programming, combinatorial optimization, evolutionary programming

## 1 INTRODUCTION

Genetic Algorithms (GAs) has now become one of the most flexible techniques to solve complex optimization problems using the ideas of natural selection and genetics. The GA [9,14] is a simulation of the genetic state of a population of individuals. GA evolves candidate solutions to an optimization problem towards better solutions. The evolution starts from a population of randomly generated individuals, and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population based on their fitness, and modified to form a new population. The cGA [12], an algorithm that mimics the order-one behavior of a simple GA with a given population size, selection rate under tournament selection and uniform crossover. The cGA reduces its memory requirements because it is not necessary to store n bits for each gene position. The cGA represents the population as a probability distribution over the set of solutions. It only needs to keep the proportion of ones (and zeros), a finite set of $n$ numbers that can be stores with $\log_2 n$ bits. However, as the problem's difficulty increase, higher selection rate must be used to produce new chromosomes that are as good as the ones already in the population. Higher selection pressure based on the cGA is used to solve soft-decision decoding effectively [6].

The example of GA-hard problem is trap function [1]. Trap function is an adversary function for studying BBs and linkage problems in GAs [11]. The general $n$-bit trap functions are defined as:

$$F_n(b_0 \ldots b_{n-1}) = \begin{cases} fhigh; & if\ u = n \\ flow - u\frac{flow}{n-1} & otherwise, \end{cases} \quad (1)$$

Where $bi \in \{0, 1\}$, $u = \sum_{i=0}^{n-1} bi$, and $f_{high} > f_{low}$. Usually, $f_{high}$ is set at n and $f_{low}$ is set at $n$-1.

The reason this is called a GA-hard problem or deceptive problem is that the GAs gets rewarded incrementally for each 0 it adds to the problem, but the best solution consists of all 1s. In swapping genes between parents in the simple GA, it will often disrupt good combinations and the average fitness in the population decrease after crossover. The crossover operator mixes and also breaks the BBs because the cut point is chosen at random (see Fig. 1). Similarly, the order-1 probability represents the population as a probability distribution over the set of solutions. Yet, generating new solutions using probability distribution leads to poor solutions. That is, both the simple GA and the cGA, fail to produce new chromosomes that are as good as the ones already in the population. A building block crossover can be developed in the GA with the purpose that the crossover operator needs to understand related genes, and not break up the combinations they represent. A building block only swap whole solutions to sub-problems, instead of single genes.

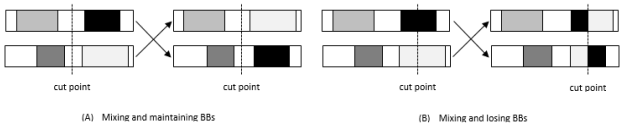


**Figure 1: The solutions are mixed by the crossover operator. The BBs are shadowed. The random cut point is selected and the tails of its two solutions are swapped to get new solutions. In case (A), the solutions are mixed while maintaining the BBs. In case (B), the BBs are disrupted.**

There are strategies in the GA literature use the bit-reordering approach to pack the dependent bits close together, for instance, inversion operator [11], messy GAs [10], symbiotic evolution [18], recombination strategy adaption [19], adaptive linkage crossover [19], and linkage learning [9], Recently, the Hybrid Linkage Crossover (HLX) operator and incorporating it into the

Differential Evolution was presented [7]. The bit-reordering approach does not explicitly identify BBs, but it successfully achieves the optimal solution. Several papers do BBs identification explicitly to find a partition of bit positions. For example, Table 1 infers the partition:

$$\{ \{0,1,2\}, \{3,4,6\}, \{7,8,9\}, \{10,11,12\}, \{13,14,15\} \} \quad (2)$$

In the case of non-overlapping BBs, partition is a clear representation [3,4,13,15-17]. The bits governed by the same partition subset are passed together to prevent BB disruption.

**Table 1: A Population of Highly Fit Individuals**

| Individual No. | $b_0b_1b_2$ | $b_3b_4b_5$ | $b_6b_7b_8$ | $b_9b_{10}b_{11}$ | $b_{12}b_{13}b_{14}$ | Fitness |
|---|---|---|---|---|---|---|
| 1 | 111 | 000 | 111 | 111 | 000 | 13.0 |
| 2 | 000 | 000 | 111 | 000 | 111 | 12.0 |
| 3 | 000 | 111 | 000 | 111 | 000 | 12.0 |
| 4 | 000 | 000 | 000 | 111 | 000 | 11.0 |
| 5 | 000 | 000 | 000 | 000 | 000 | 10.0 |

The fitness is the sum of five three-bit trap functions. "111" is the optimum for three-bit trap function. "000" gives more contribution to the fitness than that of "001", "010", "011", "100", "101", and "110". As a result, the highly fit population is composed of "000" and "111".

Moreover, identification of BBs is practically impossible in realworld problems. BBs on design variables of Interior Permanent Magnet Synchronous Motor (IPMSM) are identified and GA is applied to optimal design of IPMSM using information of identified BB [20].

There are many strategies of identifying the BBs. We will only refer to the building-block identification by simultaneity matrix (BISM) [8]. The algorithm consists of two parts: simultaneity matrix construction (SMC) and partitioning (PAR) algorithms. The SMC construct the matrix according to a set of solutions and counts a pair of two-bit BBs that are complement to each other. Then, PAR searches for a partition for the matrix. The partition is exploited in solution recombination so that the bits governed by the same partition subset are passed together.

This paper presents a new hybrid algorithm that exploits BBs with the cGA in order to solve difficult optimization problems. Let us assume that we have already know the BBs of $n$ bits. The cGA with BBs construct a probability vector ($P$) of each building block, instead of probability vector ($P$) of each single bit. The solutions are randomly generated from $P$ of each building block at each generation and $P$ of each building block is then updated base on their solutions.

The remainder of the paper is organized as follows. Section II, we will begin by briefly reviewing the working of the cGA. Then, we describe the cGA exploited by using BBs in Section III. Section IV, computer simulation compares the two algorithms, both in terms of solution quality and speed. We will then modify the cGA with BBs for the travelling salesman problem (TSP) in Section V and presents the comparison results with the cGA applied to the TSP in Section VI. At the end of the paper in Section VII, some conclusions are drawn.

## 2 COMPACT GENETIC ALGORITHM

Compact genetic algorithm represents the population as a probability vector over the set of solutions and is operationally equivalent to the order-one behavior of the simple GA (sGA) with uniform crossover. The vector contains each bit with a real number from 0.0 to 1.0 representing the probability of that bit to be one. This reduces the storage of the population to just the storage of the probability vector.

Here is a short description of the steps in cGA. In the initialization step a random population is generated. An appropriate encoding of the candidate solution is dependent on the problem. The second step is to sample two candidates from the population. Each member of the population is then evaluated and we calculate a *'fitness value'* for that individual by using the fitness function. The third step is to allow the two candidates to compete and determine a winner and a loser by comparing their fitness values. The winner's chromosome will be used to update the probability vector so that the distribution will converge towards the best fit solution. This is an iterative process until we reach a termination condition.

The cGA is approximately equivalent to the sGA with uniform crossover: it achieves solutions of comparable quality with approximately the same number of function evaluations. As a problem with higher order building blocks, cGA with both higher selection rates and larger population sizes should allow it to solve problems. Such an increase to the selection pressure helps the cGA to converge to better solutions since it increases the survival probability of higher order building blocks. Although the cGA mimics the order-one behavior of a sGA with uniform crossover, it was not proposed as an alternative algorithm. According to its authors, it can be used to quickly assess the *"difficulty"* of a problem. A problem is easy if it can be solved with a cGA exploiting a low selection rate. On the other hand, if it requires raising the selection rate to solve the problem, it should be considered as difficult.

## 3 COMPACT GENETIC ALGORITHM WITH BUILDING BLOCKS

The cGA with BBs are a set of *l*-bit binary string and a number of *n* bits in BBs. The set of *l*-bit binary string is distributed into *l*/*n* BBs and the one BB consist of $2^n$ sub-solutions. It is assumed that all bits in each BBs are dependent. The set of *l*-bit binary string denoted by:

$$S = \{s_0, s_1, \ldots, s_{l-1}\} \quad (3)$$

where $s_i$ is the *i*th string. The number of bits in BBs is *n* bits where $1 < n \leq l$. Therefore we can partition for the solution to the sub-solutions every the *n*th string. The set of BBs *l*-bit binary string denoted by:

$$S = \{ \{s_0, s_1, \ldots, s_{n-1}\}, \{s_n, s_{n+1}, \ldots, s_{2n-1}\}, \ldots, \{s_{2n}, s_{2n+1}, \ldots, s_{l-1}\} \}. \quad (4)$$

The joint probability of these *n* bits in each BBs having total $2^n$ sub-solutions. The initial probability value for each sub-solution is $1/2^n$. For example, a 30-bit trap functions problem can be formed by grouping together each three genes into a sub-solution and these three genes are related. This problem will have 10 of BBs. The joint probability of these three bits having total 8 sub-solutions and the initial probability vector for each sub-solution is 1/8 as shown in Table 2.

**Table 2: The Joint Probability of Three Bits and Initial Probability Vector for a Sub-solution**

| Sub-solution | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| Initial Probability Vector | .125 | .125 | .125 | .125 | .125 | .125 | .125 | .125 |

The joint probability of these three bits having total 8 sub-solutions including "000", "001", . . ., "111". The initial probability vector is 1/8 for each sub-solution. Fig. 2 gives pseudo code of the cGA with BBs.

Algorithm cGA with BBs is shown in Fig. 2. Step 1 initialize the probability vector *P* of BBs which input is set of *l*-bit binary string and a number of bits in BBs. Step 2 generate two individuals from probability vector of each BBs. Step 3 let them compete based on their fitness value. Step 4 BBs is then updated based on these solutions. Step 1 to 4 will be iterative until the terminating condition is met. So that in step 6 the distribution will converge to a population that fits the solution requirement. The time complexity of cGA with BBs is $O( I2^{n+1}/n )$ for one generation where *n* is a number of bits in BBs and *l* is length of bit binary string.

```
1) Initialize the probability vector p which input is set of l-
   bit binary string and n is a number of bits in BBs
       numberOfBBs = l / n;
       For i := 1 to numberOfBBs do
           For j := 1 to 2^n do
               generateAllBinaryBit(n);
                   p[j] := 1 / 2^n;

2) Generate two individuals from p
       a := generate(p);
       b := generate(p);

3) Let them compete
       Winner, loser := compete(a, b);

4) Update p towards the better one
       For i := 1 to numberOfBBs do
           For j := 1 to 2^n do
               if winner[j] ≠ loser[j] then
                   if winner[j] = 1 then
                       p[j] := p[j] + 1/β
                   else p[j] := p[j] - 1/β

5) Check if the vector p has converged
       For i := 1 to l do
           if p[i] > 0 and p[i] < 1 then
               return to step 2;

6) The probability p represents the final solution

   cGA with BBs parameters
       β: population size.
       l: bit length.
```

**Figure 2: Pseudo code of the cGA with BBs.**

For a deceptive problem such as *n*-bit trap functions problem, it will take a long time to solve problems with higher order BBs. Therefore we limit the size of trap. 10 copies of a 3-bit trap functions are concatenated to form a 30-bit trap functions. In the cGA, all the $p_i$ start with 1/2 and $p_i$ is dependent on one bit. For an order-3 schema, the survival probability is 1/8. Therefore the selection rate should greater than 8 that is enough to combat the disruptive effects of crossover. However, a selection rate of $s = 2$ in the cGA with BBs is enough to converge to reach the optimal solution because the algorithm maintains related genes by updating probability vector of BBs so that it does not break up the combinations.

## 4 EXPERIMENTAL RESULTS FOR TRAP FUNCTION

This section presents simulation results and compare the effectiveness between the cGA with BBs and the original cGA, both in terms of solution quality and in the number of function evaluations taken. For solution quality, we count the number of correct BBs. All experiments are averaged over 50 runs. The cGA uses tournament selection with $s$ = 8 while the cGA with BBs uses tournament selection with $s$ = 2. All runs end when the population fully converges that is all positions of $P$ become binary (0 or 1). Fig. 3 and 4 show the results of the experiments on the 10 copies of a 3-bit trap function. Figure 3 plots the solution quality (number of correct BBs at the end of the run) for the various population sizes. Fig. 4 plots the number of function evaluations taken until reaching the convergence state known for different population sizes.

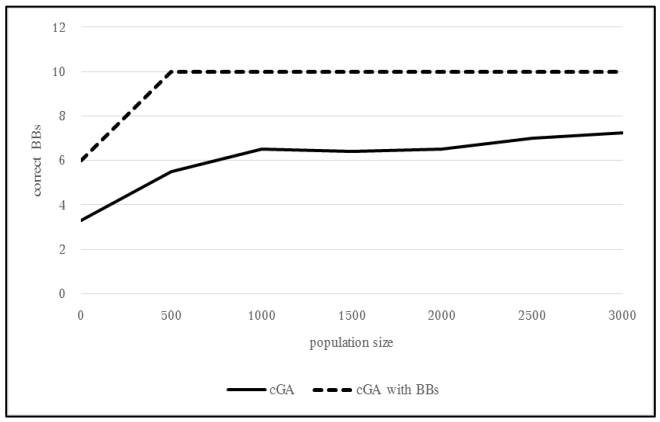


**Figure 3. Solution quality comparison achieved by the cGA and the cGA with BBs on the 10 copies of a 3-bit trap function. The solid line is for the cGA and the dashed line is for the cGA with BBs.**

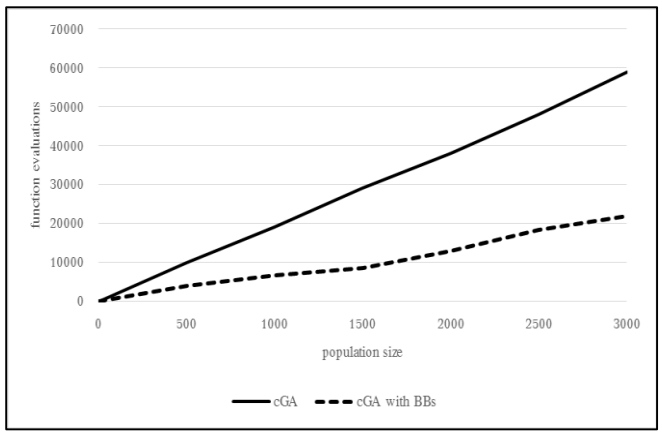


**Figure 4. Performance comparison achieved by the cGA and the cGA with BBs on the 10 copies of a 3-bit trap function. The solid line is for the cGA and the dashed line is for the cGA with BBs.**

Fig. 3 shows that the solution quality of the cGA with BBs is averaged 1.6 times better than the cGA and Fig. 4 shows that the performance of cGA with BBs is averaged 2.8 times less number of evaluations than the cGA for the various population sizes on a deceptive problem.

## 5 COMPACT GENETIC ALGORITHM WITH BUILDING BLOCKS FOR TRAVELLING SALESMAN PROBLEM

The traveling salesman problem (TSP) is the most well-known NP-hard combinatorial optimization problem. The TSP problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one, visiting each exactly once, and returning to the same city such that the total distance traveled is minimized [5]. A combinational crossover technique based on genetic algorithm and utilize the concept of heritable BBs is employed in the search for optimal or near-optimal TSP solution [2]. In this paper, in order to design the cGA with BBs for the TSP, we adopted the path-representation model which represents a feasible tour as one of the *k!* possible permutations of the $k$ cities [21]. Total number of feasible edges between cities is

$$\text{Total number of feasible edges} = \frac{k-1}{2}(k) \quad \text{where k cities} \qquad (5)$$

Total number of feasible edges represents a set of $l$-bit binary string. The initial probability value $P_{ij}$[] (one for each bit) is set to 0.5, where $P_{ij}$ is the probability vector for edge between city i and city j. For instance, Table 3 infers the number of feasible edges for 6 cities and the initial probability vector.

**Table 3: The feasible edges of 6 cities (A,B,C,D,E,F) that represents 15-bit binary string and the initial probability vector for each bit**

| $Edge_{ij}$ | $E_{AB}$ | $E_{AC}$ | $E_{AD}$ | $E_{AE}$ | $E_{AF}$ | $E_{BC}$ | $E_{BD}$ | $E_{BE}$ |
|---|---|---|---|---|---|---|---|---|
| $P_{ij}$ | .5 | .5 | .5 | .5 | .5 | .5 | .5 | .5 |

| $Edge_{ij}$ | $E_{BF}$ | $E_{CD}$ | $E_{CE}$ | $E_{CF}$ | $E_{DE}$ | $E_{DF}$ | $E_{EF}$ |
|---|---|---|---|---|---|---|---|
| $P_{ij}$ | .5 | .5 | .5 | .5 | .5 | .5 | .5 |

Let us assume that we have $k$ cities to visit. We need to have instances of problems that contain BBs so we arrange the $k$ cities into $n$ BBs. So that each BB contains the $k/n$ cities. Then, we find the best path of $k/n$ cities in each BB and the path is not a loop. To find the best path for each BB, the steps are as follows:

1. From Table 3, an $E_{AB}$ is randomly selected and define city A is the starting city from that path. Then, city A will be inserted in the tour N as the starting city.
2. To generate feasible tour: the next feasible path must not be city in tour N and must have city B in the path. City B is set to the current city

3. Generate feasible tour by randomly selecting and update traversed city in tour N and the current city.
4. Repeat step 2-3 until traverse all cities in the BB and the path is not a loop.

Then, we find the best route to connect each BB into one loop. So that we can visit each one of the cities starting from a certain one, visiting each exactly once, and returning to the same city. Generate two individuals from the probability vector and find out the best one, updating the probability vector towards the better one same as step 4 of the cGA with BBs. The above steps will be iterative until the terminating condition is met. The time complexity of cGA with BBs for TSP is $O( 2n^2k )$ for one generation where $n$ is a number of BBs and $k$ is a number of cities for each BB. The algorithm of the cGA with BBs for TSP is shown in Fig. 5.

```
Initialize( vector )
bestFitness = INT_MAX
repeat
    for i = 1 to 2 do
        routeCount = 0
        for block = 1 to n - 1 do
            for routeIndex = 1 to cityInBlock - 1 do
                validRouteVecto = validateBlockRoute
                                  (currentRoute,vector,routeArray)
                currentRoute = generateRoute
                                  (validRouteVector )
                routeArray[routeCount] = currentRoute
                                         routeCount++
            end for
        end for
            for routeIndex = 1 to numBlock do
              validRouteVector =validateBlockConnectionRoute
                                  (currentRoute,vector,routeArray)
              currentRoute = generateRoute( validRouteVector )
              routeArray[routeCount] = currentRoute
              routeCount++
            end for
        candidate[i] = routeArray
        fitness[i] = computeLength( candidate[i] )
    end for

    if( fitness[0] < fitness[1] ) then
        winner = candidate[0], loser = candidate[1]
    else
        winner = candidate[1], loser = candidate[0]
        winnerFitness = computeLength( winner )

    if ( bestFitness > winnerFitness ) then
        bestFitness = winnerFitness
        best = winner
    updateVector(vector, winner, loser)
until ( bestFitness == BEST_PROBLEM_FITNESS OR
generation == MAX_GENERATION )
output( best , bestFitness )
```

**Figure 5. Pseudo code of the cGA with BBs for TSP**

## 6 EXPERIMENTAL RESULTS FOR TSP

This section presents simulation results and compare the effectiveness between the cGA with BBs for TSP and the cGA, in terms of solution quality. All experiments are averaged over 50 runs. The tournament selection with $s = 2$. All runs end when the population fully converges that is all positions of $P$ become binary (0 or 1). Figure 6 show the results of the experiments on the TSP for U.S. cities (13 cities). Fig. 6 plots the solution quality (number of correct bits at the end of the run) for the various population sizes.

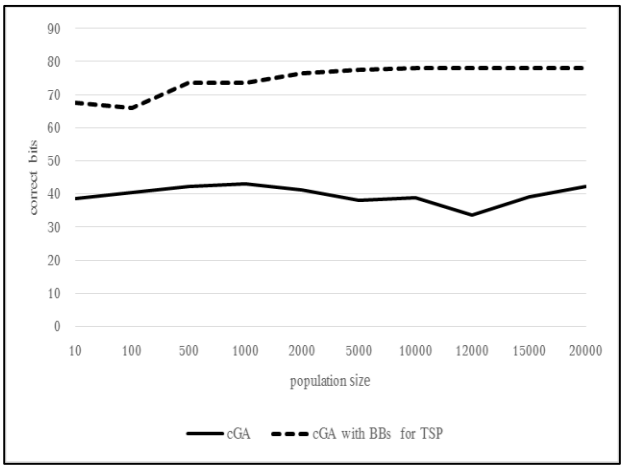


**Figure 6. Solution quality comparison achieved by the cGA for TSP and the cGA with BBs for TSP on the TSP to find the shortest route through the U.S. cities (13 cities). The solid line is for the cGA for TSP and the dashed line is for the cGA with BBs for TSP.**

Fig. 6 shows that the solution quality of the cGA with BBs for TSP is averaged 1.89 times better than the cGA.

## 7 CONCLUSIONS

In this paper, we show that the cGA with BBs can be successfully deal with difficult order-n optimization problems such as the trap function and the TSP. The optimal solution can be achieved by composing BBs. Our algorithm can maintain BBs by using probability distribution for the BB instead of individual bit; genes being considered order-$n$ BB where $n$ is a number of bits in a BB. So that the BBs are maintained in solution recombination. We introduce tournament selection with the probability vector distribution of each of the BBs. We applied our algorithm to solve a $n$-bit trap function problem. A comparison to the cGA that using higher selection pressure with randomly generated individuals is made. Empirical results show that the cGA with BBs is more effective than the cGA, both in terms of solution quality and speed. Moreover, we modified the cGA with BBs for TSP. The algorithm was evaluated on small problem instances. The results achieved were satisfactory if compared to the cGA. The cGA with BBs successfully delivers the optimal solution while the cGA fail to solve the problem.

## REFERENCES


[1] Ackley DH (1987) A connectionist machine for genetic hillclimbing. Kluwer, Boston.

[2] Al-Dallal, A. Using Genetic Algorithm with Combinational Crossover to solve Travelling Salesman Problem. in 2015 7th International Joint Conference on Computational Intelligence (IJCCI). 2015.

[3] Aporntewan C, Chongstitvatana P (2003) Building-block identification by simultaneity matrix. In: Cant-úPaz E et al (eds) Proceedings of genetic and evolutionary computation conference. Springer, Berlin Heidelberg New York, pp 1566–1567.

[4] Aporntewan C, Chongstitvatana P (2004) Simultaneity matrix for solving hierarchically decomposable functions. In: Deb K et al (eds) Proceedings of genetic and evolutionary computation conference. Springer, Berlin Heidelberg New York, pp 877–888.

[5] Aporntewan, C. and P. Chongstitvatana, Building-block Identification by Simultaneity Matrix. Soft Computing, 2007. 11(6): p. 541-548.

[6] Berkani, A., A. Azouaoui, and M. Belkasmi. Soft-decision decoding by a compact genetic algorithm using higher selection pressure. in 2015 International Conference on Wireless Networks and Mobile Communications (WINCOM). 2015.

[7] Cai, Y. and J. Wang, Differential evolution with hybrid linkage crossover. Information Sciences, 2015. 320: p. 244-287.

[8] G. Reinelt, The Traveling Salesman: Computational Solutions for TSP Applications. Berlin, Germany: Springer-Verlag, 1994.

[9] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA (1989).

[10] Goldberg DE, Korb B, Deb K (1989) Messy genetic algorithms: motivation, analysis and first results. In: Wolfram S (ed) Complex systems, vol 3, no 5. Complex Systems Publications, Inc., Champaign, pp 493–530.

[11] Harik GR (1997) Learning linkage. In: Belew RK, Vose MD (eds) Foundation of genetic algorithms 4. Morgan Kaufmann, San Francisco, pp 247–262.

[12] Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. In Proceedings of the International Conference on Evolutionary Computation 1998 (ICEC '98), IEEE New York (1998) 523–528.

[13] HarikGR(1999) Linkage learning via probabilistic modeling in the ECGA. Technical Report 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Champaign, IL.

[14] Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arboor, MI (1975).

[15] Kargupta H (1996) The gene expression messy genetic algorithm. In: Proceedings of the IEEE international conference on evolutionary computation. IEEE Press, Piscataway, pp 814–819.

[16] Kargupta H, Park B (2001) Gene expression and fast construction of distributed evolutionary representation. In: Whitley D (ed) Evolutionary computation, vol 9, no 1.MIT, Cambridge, pp 43–69.

[17] Munetomo M, GoldbergDE(1999). Linkage identification by nonmonotonicity detection for overlapping functions. In: Whitley D (ed) Evolutionary computation, vol 7, no 4. MIT, Cambridge, pp 377–398.

[18] Paredis J (1995) The symbiotic evolution of solutions and their representations. In: Eshelman LJ (ed) Proceedings of the 6th international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 359–36.

[19] Smith J, Fogarty T (1996) Recombination strategy adaptation via evolution of gene linkage. In: Proceedings of the IEEEinternational conference on evolutionary computation. IEEE Press, Piscataway, pp 826–831.

[20] Son, B., et al. Genetic algorithm adopting building block identification applied to optimal design of IPMSM. in 2016 IEEE Conference on Electromagnetic Field Computation (CEFC). 2016.

[21] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed. Berlin, Germany: Springer-Verlag, 1996.