



## The use of explicit building blocks in evolutionary computation

Chalermsub Sangkavichitr & Prabhas Chongstitvatana

**To cite this article:** Chalermsub Sangkavichitr & Prabhas Chongstitvatana (2016) The use of explicit building blocks in evolutionary computation, International Journal of Systems Science, 47:3, 691-706, DOI: [10.1080/00207721.2014.901580](https://doi.org/10.1080/00207721.2014.901580)

**To link to this article:** <http://dx.doi.org/10.1080/00207721.2014.901580>



Published online: 22 Apr 2014.



Submit your article to this journal [↗](#)



Article views: 47



View related articles [↗](#)



View Crossmark data [↗](#)

## The use of explicit building blocks in evolutionary computation

Chalermsub Sangkavichitr\* and Prabhas Chongstitvatana

*Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand*

*(Received 23 August 2012; accepted 12 April 2013)*

This paper proposes a new algorithm to identify and compose building blocks. Building blocks are interpreted as common subsequences between good individuals. The proposed algorithm can extract building blocks from a population explicitly. Explicit building blocks are identified from shared alleles among multiple chromosomes. These building blocks are stored in an archive. They are recombined to generate offspring. The additively decomposable problems and hierarchical decomposable problems are used to validate the algorithm. The results are compared with the Bayesian optimisation algorithm, the hierarchical Bayesian optimisation algorithm, and the chi-square matrix. This proposed algorithm is simple, effective, and fast. The experimental results confirm that building block identification is an important process that guides the recombination procedure to improve the solutions. In addition, the method efficiently solves hard problems.

**Keywords:** genetic algorithm; estimation of distribution algorithm; building block; linkage; multi-parent recombination

### Introduction

Evolutionary computation (EC) is a class of algorithms that are suitable for problems with a very large search space (Yu & Gen, 2010). The classic algorithm in EC is the genetic algorithm (GA) (Goldberg, 1989), which was inspired by the process of genetic breeding in nature. GAs have four main processes. First, the solution is encoded into a specific pattern, such as a binary string, called a chromosome or an individual and a number of encoded solutions, called a population, are generated. Second, good chromosomes are selected from the population. Third, selected chromosomes are mated using a crossover operator. Fourth, mated chromosomes are mutated. The fourth process is an optional step. New offspring are produced after these processes are completed.

As in nature, the evolution process will improve solutions from generation to generation. The EC algorithms are based on the premise that better solutions can evolve through selection and genetic operations (Goldberg & Sastry, 2010).

In GAs, the solutions are improved based on the assumption that substructures of the good solutions exist in the good individuals. If the substructures can be identified and combined correctly, the new solutions will be better (Goldberg & Sastry, 2010). The schema theorem (Schaefer, 2007) tells us that the properties of good substructures are 'short, low order, and highly fit'. These substructures are called building blocks (BBs). The BB hypothesis (BBH) states that the short, low-order, highly fit schemata recombine to form higher order schemas of conceivably higher

fitness. This hypothesis is used to explain the mechanism of GAs. To perform the recombination, a crossover operator is used. The crossover operation was claimed to construct rather than disrupt BBs (Schaefer, 2007). Generally, one-point crossover is sufficient to solve problems. However, several crossover methods have been introduced to improve the effectiveness of the recombination, such as the two-point, multiple-point, and uniform crossover methods (Sivanandam & Deepa, 2008). Different methods work well for different types of problems. The mutation operation was reported to provide a new source of genetic material (Schaefer, 2007). However, setting the appropriate mutation rate is difficult. We only know that it should be low. Conversely, the crossover rate should be high (Eiben & Smith, 2011; Lobo, Lima, & Michalewicz, 2007). The canonical GA is called the simple GA (sGA). It uses fitness-proportion selection, single-point crossover, and one-bit mutation. The sGA algorithm is easy to use, but it cannot solve hard problems effectively (Sivanandam & Deepa, 2008). However, several GAs have been successfully applied to a wide range of real world problems, such as Wang (2009), Cheung, Cheung, Tobar, Caram, and Garcia (2011), Alabsi and Naoum (2012), and Ozcan and Esnaf (2013).

Multiple parent recombination operators have been studied (Eiben, 2000; Eiben, Rauee, & Ruttkay, 1994; Ting & Chen, 2007). Rather than limiting the gene from two parents, the gene pool from the population was used. Some forms of explicit blocks of genes that come from multiple parents were introduced in many pioneering works (Levenick, 1995; Smith & Fogarty, 1995; Syswerda, 1993).

\*Corresponding author: Email: [chalermsub.san@kmutt.ac.th](mailto:chalermsub.san@kmutt.ac.th)

These ‘blocks’ were used by recombination operators to compete for a position over certain loci. In Syswerda (1993), an explicit model was created by counting the number of alleles in each bit position contributed by the selected population. This information was used to perform recombination of genes. Smith and Fogarty (1995) used the linkage between blocks of genes of varying size to generate offspring.

A parallel concept to BBs is linkage learning. Linkage is a relation between decision variables. While BBs are defined as a group of genes, linkage emphasises more on the relationship between genes. One important factor in the success of evolutionary algorithms is a good coding scheme that puts genes in the same BBs together thus providing tight linkage. Many linkage learning methods have been proposed to discover good BBs, for example, linkage learning GA and its variance (Chen, 2006; Chen & Goldberg, 2002; Chen & Lim, 2008). A good survey of the linkage learning can be found in Chen, Yu, Sastry, and Goldberg (2007).

If the BBs are mixed properly, the quality of the result will be improved (Goldberg & Sastry, 2010). Therefore, the BBs must be identified. BB identification (BBI) (Aporntewan & Chongstitvatana, 2005) is a method that explicitly identify BBs using correlation between genes. This knowledge is then used to partition genes into groups. Once the BBs are identified, the appropriate crossover is used to recombine BBs. Although there are many ways to identify the BBs, a majority of the methods are estimation of distribution algorithms (EDAs) (Larrañaga & Lozano, 2001; Pelikan, Goldberg, & Lobo, 2002; Pelikan, Sastry, & Cantú-Paz, 2006). The algorithms that employ some data structure (so-called ‘model’) to organise the decision variables are also considered to use linkage. They include the probabilistic model-building GAs or EDAs (Coffin & Smith, 2008), where the probabilistic models are used to represent relationship of decision variables. In the EDAs, knowledge is shared using a distribution model for the population, and new offspring are sampled from the model. However, most EDAs require some knowledge to identify the relationships between individuals in a population and to build a model. Because BBs are explicitly extracted in terms of probabilistic models, EDAs can solve hard problems effectively (Larrañaga & Lozano, 2001; Pelikan et al., 2006). The knowledge sharing in both the model building and model sampling processes is the main advantage of the EDA method.

The messy GA (mGA) is another approach that uses explicit BBs in the early stage (Goldberg, Deb, Kargupta, & Harik, 1993; Goldberg, Korb, & Deb, 1989). The mGA worked by building up increasingly longer and better strings from shorter BBs. It progressed in two phases: primordial and juxtapositional. The first phase built up the population of small and highly fit candidate. The second phase puts them together to form solutions. The juxtaposition is

performed by ‘cut’ and ‘splice’ operators that work on messy encoding similar to the crossover operators work on a fixed length string. A newer generation of mGA is the gene expression mGA (Kargupta, 1996) that simplified the encoding of strings and incorporate linkage learning phase which resulted in a large reduction of running time. A recent study proposed a simplified BB, called a ‘Fragment’ (Sangkavichitra & Chongstitvatana, 2010). The substructures of BBs are defined as the substructures that two highly fit individuals have in common. These substructures are the fragments. According to the BBH, the Fragments are the BBs because they can be recombined into higher order BBs (Goldberg & Sastry, 2010). The sGA with fragment crossover (sGA-FC) method is introduced. It is based on the sGA and a special crossover (Sangkavichitra & Chongstitvatana, 2010). The FC tries to maximise the schema exchange using multiple-point crossover scheme.

In this paper, we propose a new simple method that was inspired by multi-parent recombination, which can share alleles among multiple chromosomes and EDAs, which can identify and compose the BBs explicitly. The proposed method is named the ‘BBs identification and composition’ (BBIC) algorithm. Two well-known BB validation problems were used to demonstrate the behaviour of the proposed algorithm. The capabilities of the BBIC algorithm were benchmarked using a set of hard problems (Collard, Gaspar, Clergue, & Escazut, 1998; Finger, Stutzle, & Lourenco, 2002). The results were compared with the sGA-FC, the chi-square matrix (CSM) (Aporntewan & Chongstitvatana, 2004), the Bayesian optimisation algorithm (BOA) (Pelikan, 2005), and the hierarchical BOA (hBOA) (Pelikan, 2005). The proposed method outperforms these algorithms in terms of the number of function evaluations and the execution time.

The paper is organised as follows. The BB definition is presented in Section 2. Section 3 describes the BBIC algorithm. The experiment and results are explained in Section 4. The benchmark problems and results are presented in Section 5. Section 6 discusses a plausible mechanism that explains the behaviour of the algorithm. Finally, concluding remarks about the proposed method are given in Section 7.

### Building block

Generally, GAs are simple to use but the underlying mechanism is difficult to understand. The schema theorem was proposed to explain the effect of each operator (selection, crossover, and mutation) on the evolution. The effect of recombination process on BBs is not easily understood because it has an indirect impact on BBs. This paper introduces a simpler form of the BB. The proposed BB is defined as a contiguous substructure of a chromosome and is a part of a subschema. The defined BB can be interpreted as a basic form of the explicit subschema. Throughout this article,

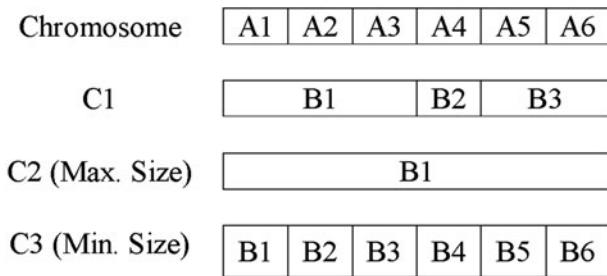


Figure 1. An example of possible patterns of BBs in a chromosome (A = Allele, C = Chromosome, B = Building Block).

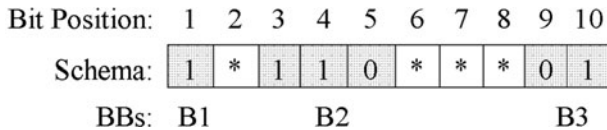


Figure 2. An example of BBs in a schema (B = Building Block).

we will use this definition of BB. The BBs in a chromosome can take many possible patterns as shown in Figure 1. An example of a BB is shown in Figure 2. The minimum size of a BB is one allele, and the maximum size is equal to the length of the chromosome.

### BBIC: building blocks identification and composition algorithm

In this section, we describe the proposed algorithm. The BBIC algorithm consists of two main parts: BBI and BB composition (BBC). The BBI process extracts the substructures that are common between good individuals (selected chromosomes), and the BBC process assembles them to create new offspring. The main objective of the BBIC algorithm is to capture and combine the sharing knowledge between good individuals. The main advantages of the algorithm are its simplicity and its performance.

#### Building block identification

Typically, GAs do not explicitly recognise BBs. The main operator that acts on BBs is the crossover, which processes BBs in an implicit manner (Schaefer, 2007). There are many crossover methods, each suitable for a different type of problem. Most of the methods recombine the schemata without knowledge about the BBs. However, they work well because the crossed chromosomes are good solutions that have passed a selection process and are expected to contain the ideal BBs (Goldberg & Sastry, 2010). The crossover operation mixes the chromosomes. Normally, these methods differ in the number of cross points and the number of parents. In the exchange, the crossover operator only manipulates the substructures or alleles that differ between chromosomes. The similar substructures remain in the same place, only the distinct parts are moved or separated. This

exchange mechanism causes schema disruption. The different patterns of disruption in each crossover method bias the results in different ways (Smit & Eiben, 2010). Substructures that are common between chromosomes are unchanged during crossover. The common substructures are candidate BBs. We consider the similarity of bits that are in the same position of two chromosomes. This information can reveal the boundary of the potential BBs. The distinct parts should not be disturbed because they might be part of the BBs. We define the BB as the substructures that are common and uncommon between any two chromosomes. Both types of BB are retained to maintain the diversity in each generation. Evolution mechanisms (selection and recombination) process these BBs to improve the quantity and quality of the solutions.

The BBI process is illustrated by an example in Figure 3. Note that the string is indexed from left to right starting at position 1. Given two 10-bit chromosome sequences, C1 = (1,0,1,1,0,1,0,1,0,1) and C2 = (1,1,1,1,0,0,1,0,0,1), the BBs of C1 and C2 are B1 = (1), B2,1 = (0), B2,2 = (1), B3 = (1,1,0), B4,1 = (1,0,1), B4,2 = (0,1,0), and B5 = (0,1) consecutively.

A common schema is defined as any identical contiguous parts between two chromosomes, and the uncommon schema is defined as the difference. The BB can be interpreted as contiguous subsequences that are common and uncommon between two chromosomes. The length of each BB must be greater than or equal to one bit. The definitions of the order and length of the BB are similar to those used in GAs. The order of a schema is the number of fixed bit positions. The BB comprises all of the fixed bit positions; therefore, the size of a BB is the order, e.g., in Figure 3, B3 = (1,1,0) and order (B3) = size(B3) = 3. The length of a schema is the distance between the first and last fixed bit positions, e.g., in Figure 3, B3 begins at position 3 and ends at position 5 and length(B3) = 5 - 3 + 1 = 3. For BBs, the short and low-order schemata have a high potential to survive, and they will be recombined to create better solutions.

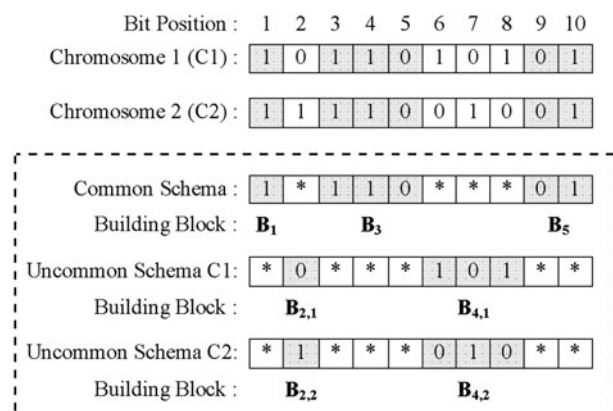


Figure 3. Building block identification method.

$D$  denotes a population.

$R$  denotes an individual (chromosome).

$B$  denotes common and uncommon subsequence (Building Blocks).

$T$  denotes an archive of common subsequences.

**Algorithm** BBIC {Building Block Identification and Composition algorithm}

$D_0 \leftarrow$  Generate  $R$  individuals (the initial population) at random

**Repeat** for  $g = 1, 2, \dots$ , until the stopping criterion is met

$D_{g-1}^N \leftarrow$  Select  $N < R$  from  $D_{g-1}$  according to selection method.

$B \leftarrow$  Find the common and uncommon substructures (BBs) between  $D_{g-1}^N$ .

{Building Block Identification}

$T \leftarrow$  Label  $B$  and record. {Building Block Composition}

$D_g^N \leftarrow$  Sample  $R_g$  individuals (the new population) from  $T$ .

$D_g^N \leftarrow$  Mutate  $R_g$  individuals from  $D_g^N$  (optional).

**EndRepeat**

**End.**

Figure 4. Pseudocode for the BBIC algorithm.

The BBs can be considered as independent contiguous sub-schema that are tightly linked.

### Building block composition

The knowledge sharing in the recombination process and the creation of new offspring process provides the main advantage of the multi-parent crossover GAs (Eiben, 2002) and EDAs. These methods gain exploration power to find possible patterns of good solutions and enhance the quality of the solutions (Toussaint, 2003). The BBIC algorithm maintains the diversity of the BBs obtained in the identification process by keeping them in an archive. The BBC demonstrates that the BBs are mixed in an explicit manner. There are many ways to compose BBs, but a simple method is sufficient. The BBs are selected one by one from the archive to create a new offspring. They are combined sequentially starting from the first position, as shown in Figure 7. In the archive, many BBs begin at the same position. If there is no guidance as to which BB is better, a random

selection will avoid biases. When the first BB is selected for the first position, the next BB is concatenated to it, and the process is repeated until a new chromosome is formed. If there is no BB that starts at the current position, a random binary value (0 or 1) is used at that position. This situation might occur if the diversity of the population is too low because the population size is too small or the population is nearly converged.

An overview of the BBIC algorithm (Figure 4) is as follows. First, the good solutions are selected from the population. Based on the schema theorem and the BBH, the chance of survival for a schema increases in a chromosome that has fitness above average. Thus, the above-average chromosome will be selected using the  $n$ -Best selection method (select  $n$  best individuals). Second (Figure 5), BBs are identified using the common schema partitioning criterion mentioned previously. In this step, every individual will be collated to explore every possible BB. Third, all BBs are labelled using their beginning position in the original individual. An example is shown in Figure 7. The BB B2



```

R denotes an individual (chromosome).
B denotes common and uncommon subsequences (Building Blocks).
Algorithm Building_Block_Identification
{The comparison of individual  $R_i$  to  $R_1, \dots, R_{i-1}$ }
  For  $i = 2$  to (population_size / 2) do
    For  $j = 1$  to ( $i-1$ ) do
       $B \leftarrow \text{Compare } R_i \text{ to } R_j$ 
      (search for common and uncommon substructures)
    EndFor
  EndFor
End.

```

Figure 5. Pseudocode for building block identification.

begins at position 4 in the chromosome and is stored in an archive. Fourth (Figure 6), the BBs in the archive are randomly selected one by one to compose the new chromosome from the first position to the last position. This process is repeated until the new population is fulfilled.

The time complexity of the BBI and the BBC processes are  $O((n)^2 \cdot l)$  and  $O(n)$ , respectively, where  $l$  is the chromosome length and  $n$  is the number of selected chromosomes. There are two basic parameters, the population size and the

number of selected chromosomes in the  $n$ -Best selection method. Compared to other GAs, or EDAs, it is easy to tune the algorithm. The mutation process is optional.

Figure 7 shows an example of the BBI and the BBC process. Two chromosomes,  $C1 = (1010101010)$  and  $C2 = (1011011001)$ , are compared using the common schema partitioning criterion. Six BBs are found. There are two common parts and two different parts between  $C1$  and  $C2$ . The BBs  $B1 = (101)$  and  $B4 = (10)$  are the common parts, and  $B2 = (010)$ ,  $B3 = (101)$ ,  $B5 = (10)$ , and  $B6 = (01)$  are the different parts. The starting positions of BBs  $B1$ – $B6$  are 1, 4, 4, 7, 9, and 9, respectively. The BBs are labelled and archived into a composition table according to the beginning positions. The results are obtained once all pairs of the selected individuals have been collated. They are shown in the next composition table in Figure 7. The next process is BBC, which creates new offspring for the next generation. In the example, two new chromosomes were produced,  $N1$  and  $N2$ . Chromosome  $N1$  was started by randomly selecting a BB for the position one. The BB  $B31$  was selected. The length of  $B31$  is two, thus the next BB must begin at the position three. The BBs  $B12$ ,  $B32$ , and  $B5$  were randomly selected to complete  $N1$ . Chromosome  $N2$  was created in the same manner as  $N1$ , but the last position has no BB in the archive. Therefore, a 0 or 1 bit is randomly generated.

```

R denotes an individual (chromosome).
 $B_i$  denotes a set of common and uncommon subsequences (Building Blocks) that
  begin with the bit at position  $i$ .
C denotes a Fragment
Algorithm Building_Block_Composition
{The composition of the new individual  $R$ }
   $i = 1$  {first position}
  While  $i < \text{individual\_size}$  do
    if  $B_i \neq \emptyset$  then {if has one or more BBs}
       $C \leftarrow \text{random}(B_i)$ 
       $i \leftarrow i + \text{size}(C)$  {assign next position}
    else {if there is no member}
       $C \leftarrow \text{random}(0,1)$  {random from 0 or 1}
       $i \leftarrow i + 1$ 
    endif
     $R \leftarrow R + C$  {concatenation}
  EndWhile
   $R \leftarrow \text{mutation}(R)$  {optional}
End.

```

Figure 6. Pseudocode for building block composition.

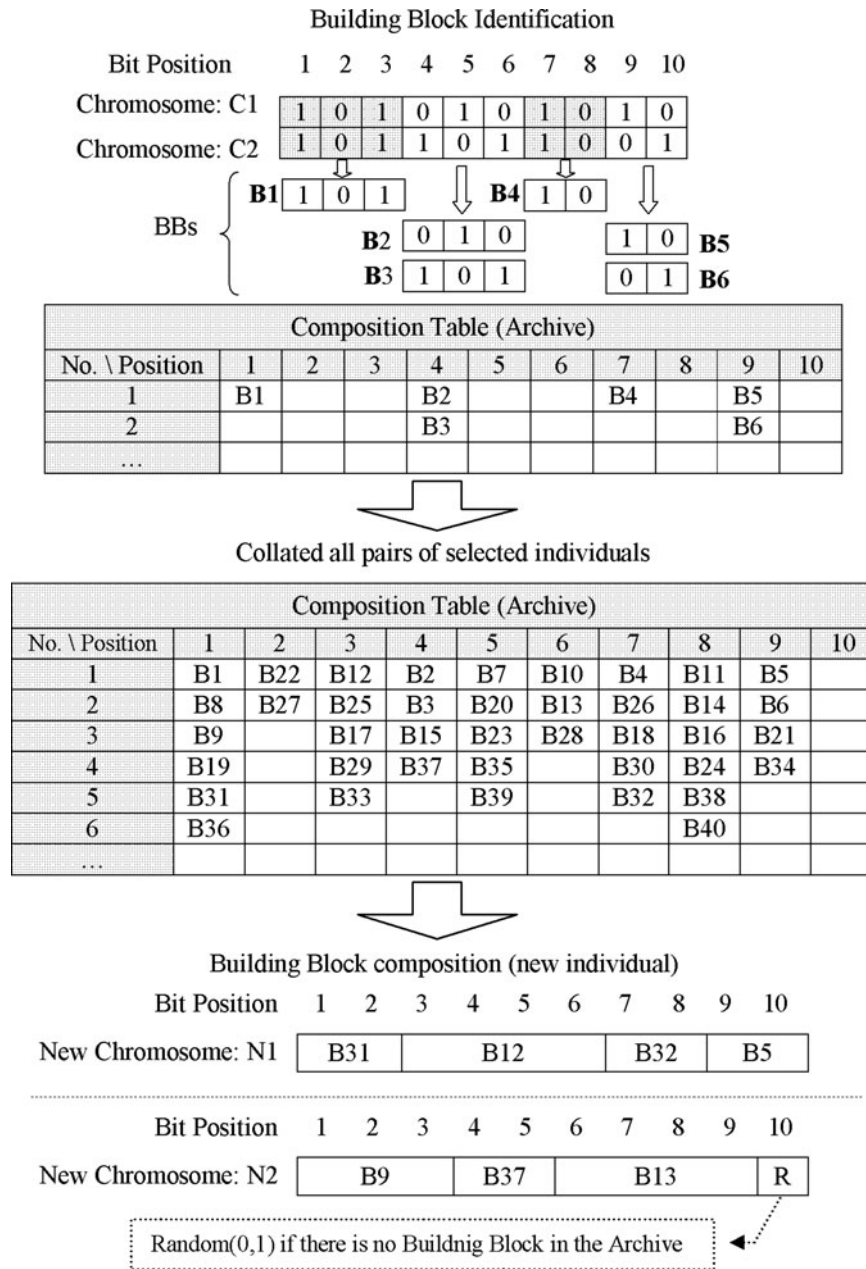


Figure 7. Building block identification and composition methods.

### Experimental settings and results

There are two parts of the experiment. The first part exhibits the behaviour of BBs processing by comparing the BBIC algorithm to the sGA. The second part compares the performance of the BBIC algorithm to a group of competent algorithms. They are sGA-FC, CSM, BOA, and hBOA. The sGA is used as the basic reference. The details of the first part of the experiments are described in the following sections.

### Test problems

Most of the problems used in the experiment are synthetic functions. The problems are classified into two categories: non-deceptive and deceptive. A deceptive problem lures the algorithm away from the ideal solution. Generally, deceptive problems are more difficult to solve than non-deceptive problems. It is difficult to claim what algorithm is suitable for a particular class of problems. Nevertheless, an experiment can be conducted to support the statement. In

this experiment, the Royal Road function (non-deceptive) (Howard & Sheppard, 2004) and the Trap-5 function (deceptive) (Beaudoin, Verel, Collard, & Escazut, 2006) were used. These two functions belong to the class of additively decomposable functions (ADFs).

The Royal Road function was designed to test the ability of GAs to compose BBs. The general  $k$ -bit Royal Road is defined as

$$E_k(b_1, \dots, b_k) = \begin{cases} f; & \text{if } u = k \\ 0; & \text{otherwise,} \end{cases} \quad (1)$$

where  $b_i$  is in  $\{0,1\}$ ,  $u = \sum_{i=1}^k b_i$ , and  $f = k$ .

ADFs denoted by  $E_{m \times k}$  are defined as

$$E_{m \times k}(k_1 \dots k_m) = \sum_{i=1}^m E_k(k_i), \quad k_i \in \{0,1\}^k. \quad (2)$$

Variables  $m$  and  $k$  are varied to produce a number of test functions. This problem is difficult because a hint about the BBs is not provided. The optimal solution is composed of all ones. This problem is representative of problems that have a simple BB structure.

The well-known Trap functions were designed to study BBs and linkage problems in GAs. The general  $k$ -bit trap functions are defined as

$$E_k(b_1, \dots, b_k) = \begin{cases} f_{\text{high}} & ; \text{ if } u = k \\ f_{\text{low}} - ((u \times f_{\text{low}})/(k - 1)) & ; \text{ otherwise,} \end{cases} \quad (3)$$

where  $b_i$  is in  $\{0,1\}$ ,  $u = \sum_{i=1}^k b_i$ , and  $f_{\text{high}} > f_{\text{low}}$ . Usually,  $f_{\text{high}}$  is set to  $k$  and  $f_{\text{low}}$  is set to  $k - 1$ . The Trap problem is defined using Equation (2).

The Trap functions fool the gradient-based optimisers to favour zeros, but the optimal solution is composed of all ones.

### Measurement

To be able to visualise the behaviour of BBs during processing, the BBs were classified into three classes: pure BB, mixed BB, and non BB. An example is shown in Figure 8. Pure BB means that the pattern in a chromosome corresponds to the ideal BBs in the problem. Mixed BB means that there is at least one allele (one bit) of the ideal BBs in the structure, thus mixed BB can be regarded as a substantial source of genetic material in the recombination process (and also a source of diversity). Non BB means that no allele of the ideal BBs is in the structure. This class can be considered as a barrier to achieving the desired BBs. The number of BBs classified as pure BB depends on the particular problem and its encoding length. Pure BB is used to indicate the performance of algorithms.

<b>Desired BB</b>	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1		
Pure BB	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1		
Mixed BB	<table><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1		
Mixed BB	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1		
Non BB	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0		

Figure 8. Examples of BB classification.

In the experiment, the binary encoding length of Royal Road and Trap-5 functions were 64 and 60 bits, respectively. The numbers of BBs classified as pure BB were 8 and 12 for the Royal Road and Trap-5 problems, respectively. The number of BBs was measured in each generation over the entire population. For example, if a population size was 100 for the Trap-5 60-bit problem, there were 12 (BBs)  $\times$  100 (chromosomes), equal to 1200 BBs (pure + mixed + non BBs), in each generation.

The BBs were used to illustrate the schema processing and the schema construction. There are two types of BBs: common and difference (uncommon). The common BBs are the similar and contiguous bits found in two good chromosomes. The different BBs are uncommon contiguous bits found in good chromosomes. The numbers of common and different BBs were calculated from each BBI operation. For example, there were two common BBs and four uncommon BBs in Figure 7. A change in numbers of the BBs in each type is an indicator of the behaviour of the BBs during processing. For example, when the size of common BBs grows, it indicates that the BBs have been combined into larger BBs. This usually happens when the evolution has been converged to a local optimum. The number of BBs can be a good indicator of the diversity in the population. The ratio between the common and different BBs indicates the competition between alternate schemata.

### Conditions

All tested problems were performed with 30 independent runs in both success and failure cases. All algorithms were required to find the optimal solution in all of the 30 runs for both cases. In the success cases, the minimum population size was set to achieve the optimum in all of the runs. The failure case, the maximum population size was reported. The numbers of Function Evaluation did not exceed 1 million for the Royal Road problem, and 50,000 for the Trap-5 problem. The BBIC algorithm uses the  $n$ -Best selection and draws only the good chromosomes that have



Table 1. Experimental parameter settings and results.

Parameters		Success 30 run				Failure 30 run	
		sGA		BBTC		sGA	BBTC
Problems	Size (bit)	#pop	#FEs	#pop:#sub	#FEs	#pop	#pop:#sub
Royal Road	64	1200	11,900	2000:200	13,437	200	200:100
Trap-5	60	2300	28,400	1000:100	7834	300	100:50

Note: #pop denotes the population size, #sub denotes the subpopulation (the selected individuals for identification process), and #FEs denote the number of function evaluation.

above average fitness. The maximum number of the selected chromosomes, or subpopulation, was limited to half of the population size. The sGA employs the tournament selection method, and the tournament size was four. The crossover rate of the sGA was set to 1.0 for the best result. The mutation operator was not allowed in either algorithm because the aim of the experiment was to test the capability of the schema processing (the BB recombination), and thus it is better to avoid another source of genetic ma-

terial. These parameter settings were the same for all test problems. In the failure case, the behaviour of the sGA and the BBIC algorithm was shown only for the first 100 generations.

The performance of GAs is compared using the number of function evaluation (#FEs). All results are averaged over the 30 runs using the same parameters setting. The parameters setting and results of the Royal Road and the Trap-5 problems are shown in Table 1.

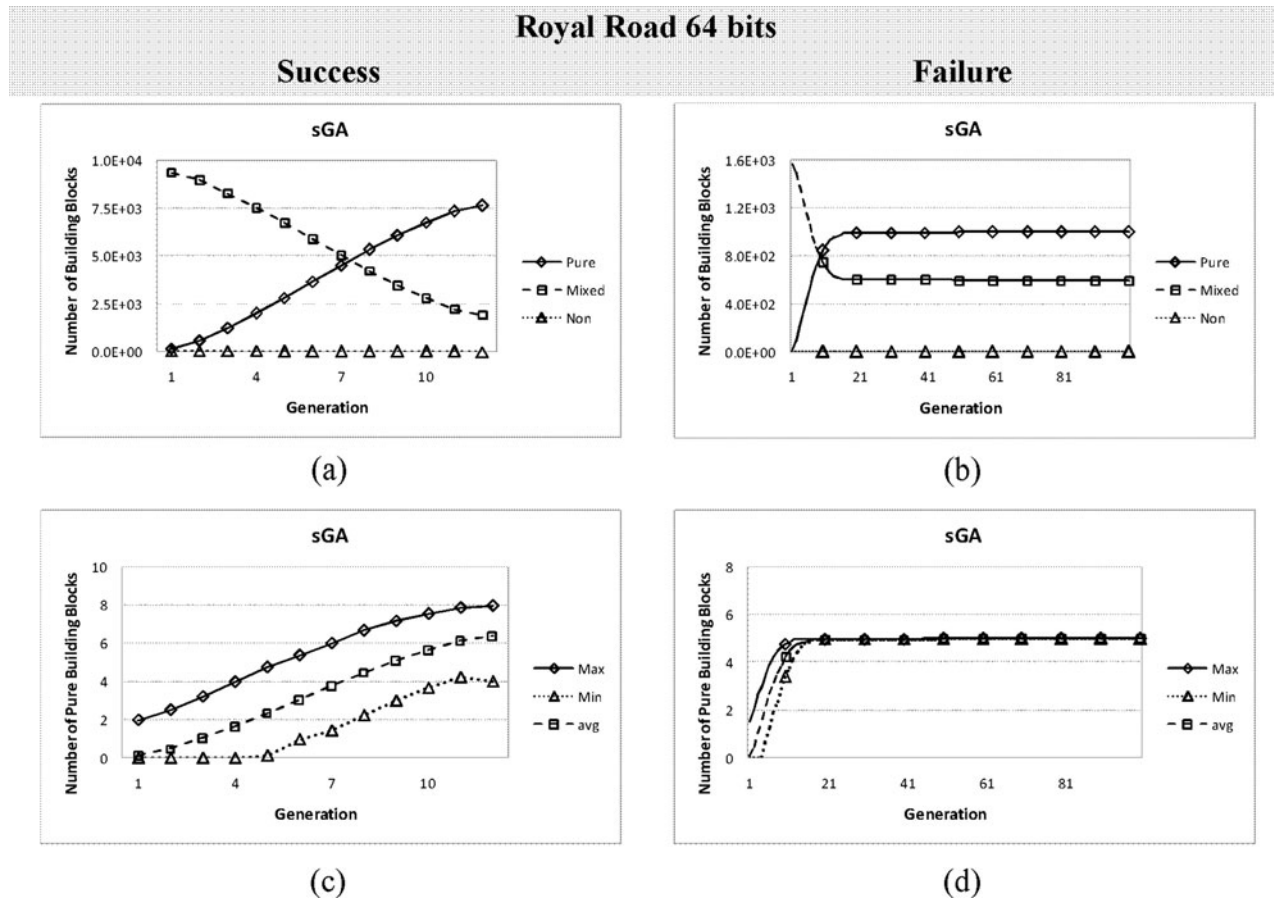


Figure 9. The sGA results for the Royal Road 64-bit problem: (a) and (b) the number of BBs in each generation, and (c) and (d) the number of ideal BBs (pure BBs) in each generation.

## Results

From the viewpoint of performance index (#FEs), the BBIC algorithm is inferior to the sGA in the Royal Road problem, but it outperforms the sGA in the Trap-5 problem. The population sizes in the success and failure cases are considered upper and lower bounds of the initial source of the diversity because there was no mutation or other source of genetic material during the process.

By design, the success case requires a minimum population size to achieve the optimal solution in all of the runs. This is regarded as an upper bound of the diversity that is required to obtain reliable results from the experiment. The failure case requires a maximum population size so that the optimal solution is not attained in all of the runs. This setting implies that if there is a larger population, the optimal solution can be found at least once. The population size in the failure case can be interpreted as a lower bound of the diversity that existed in the population. The behaviours of BBs during processing in both problems are discussed in the following paragraphs.

## Behaviour of BBs

The success of an algorithm is affected by the diversity of the population (to provide the genetic material). In the success case, the population size of the BBIC algorithm

is larger than the sGA in the Royal Road problem, and it is smaller than the sGA in the Trap-5 problem. In the failure case, the population size of the BBIC algorithm is equal to the sGA in the Royal Road problem, and it is smaller than the sGA in the Trap-5 problem. The size of the subpopulation was set to the maximum value (half population size) in the failure case to provide the highest diversity. This setting guarantees the lower bound of the population size that cannot find the optimal solution in all of the runs. These facts indicate that the BBIC algorithm is better than the sGA at maintaining the diversity in the Trap-5 problem, and it is worse than the sGA in the Royal Road function.

The behaviour of the sGA in the Royal Road problem is shown in Figure 9. The different types of BBs illustrate the change or transformation during the evolution process. Because the Royal Road problem is not deceptive, only the pure BBs and the mixed BBs compete in Figure 9(a) and 9(b). The number of pure BBs shows the quality of the solution. There are eight BBs in this problem. In the success case, the number of pure BBs increases continuously until the optimal solution is obtained because there is sufficient diversity. Figure 9(c) and 9(d) shows the convergence pattern of the entire population. In the failure case, the number of pure BBs is saturated at five from the 21th generation onwards.

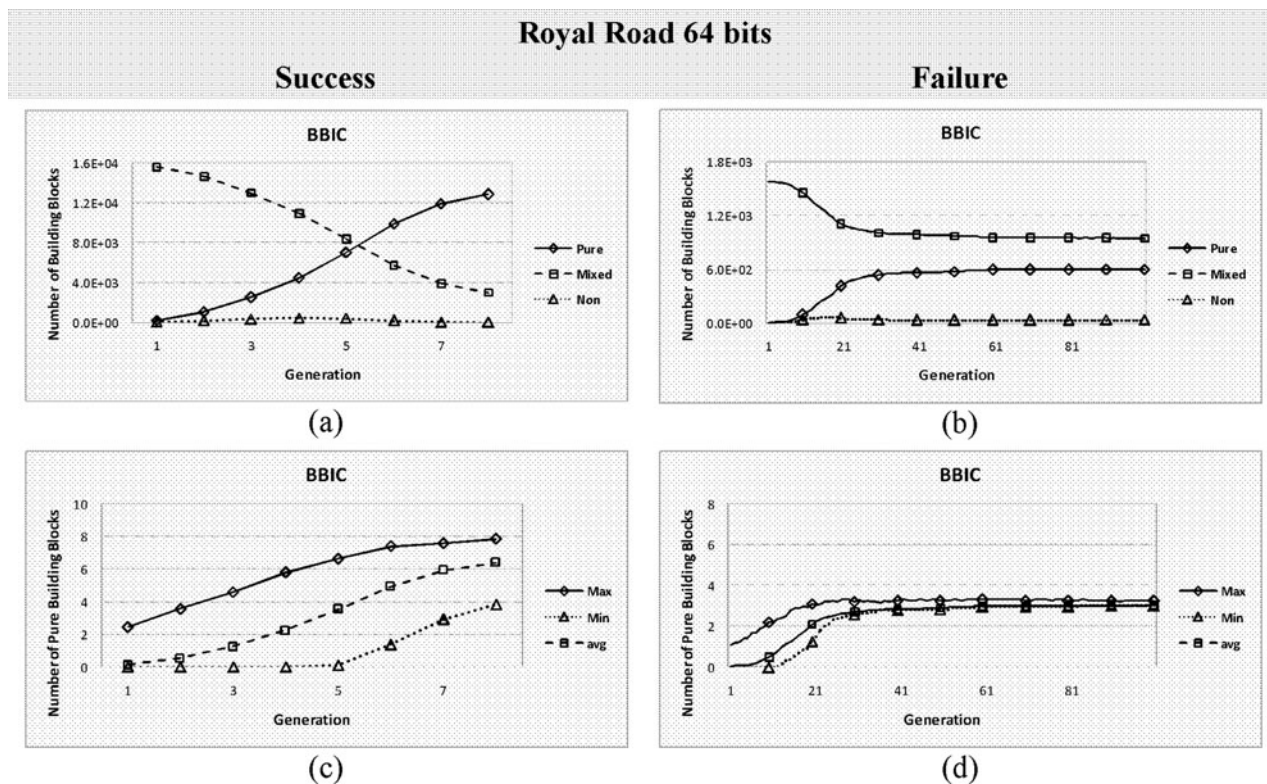


Figure 10. The BBIC algorithm results for the Royal Road 64-bit problem: (a) and (b) the number of BBs in each generation, and (c) and (d) the number of ideal BBs (pure BBs) in each generation.

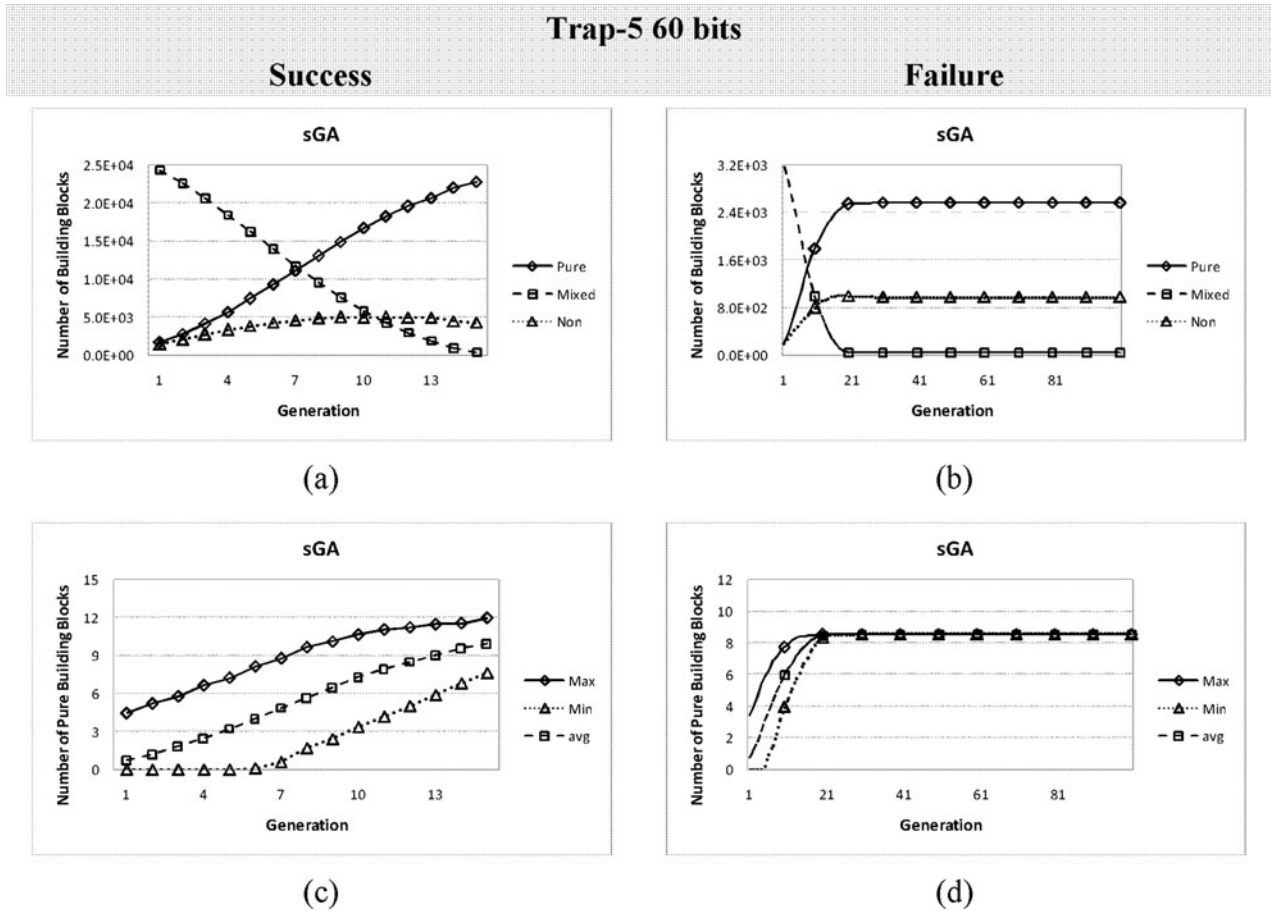


Figure 11. The sGA results for the Trap-5 60-bit problem: (a) and (b) the number of BBs in each generation, and (c) and (d) the number of ideal BBs (pure BBs) in each generation.

For the BBIC algorithm, the behaviour is similar to the sGA (Figure 10(a)–(d)). There is a slight difference in the failure case. This result can be observed in Figure 10(b), which shows that the number of mixed BBs is always higher than the number of pure BBs. The algorithm cannot learn enough to achieve good solutions, so the quality of the solutions in Figure 10(d) is inferior to the sGA (the BBIC algorithm achieved only three pure BBs versus five pure BBs in the sGA).

The results from the Trap-5 problem are shown in Figures 11 and 12. In Figure 11, the overall behaviour of the sGA for this problem is similar to Figure 9, except that there are non BBs in the competition. This result is due to the deceptive impact of the Trap function.

For this problem, the BBIC algorithm performed better than the sGA because of the recombination power of the BBC process. Comparing the quality of solutions of the sGA (Figure 11(c)) and the BBIC algorithm (Figure 12(c)), the number of pure BBs in the BBIC algorithm is higher than the sGA (for example, at the 10th generation). This is also noticeable in the failure case (BBIC in Figure 12(b));

the mixed BBs are maintained at a higher level than in the sGA (Figure 11(b)).

### Competition among BBs

The data from the experiments on the BBIC algorithm are used to illustrate the competition among BBs. The average size of the BBs and the number of BBs are good indicators of the competition between common and different BBs. Figure 13(a)–(d) illustrates the competition behaviour in the Royal Road problem. Figure 14(a)–(d) illustrates the Trap-5 problem. In both figures, CB denotes the average common BB size, DB denotes the average different BB size, #CB denotes the number of common BBs, and #DB denotes the number of different BBs.

The average sizes of BBs in Figure 13(a) and 13(b) and the number of BBs in Figure 13(c) and 13(d) indicate the convergence. These two values are opposite. If the size of BBs increases, the number of BBs decreases. In the success case (Figure 13(a)), only the common BBs are gradually developed while the different BBs remain constant. In the



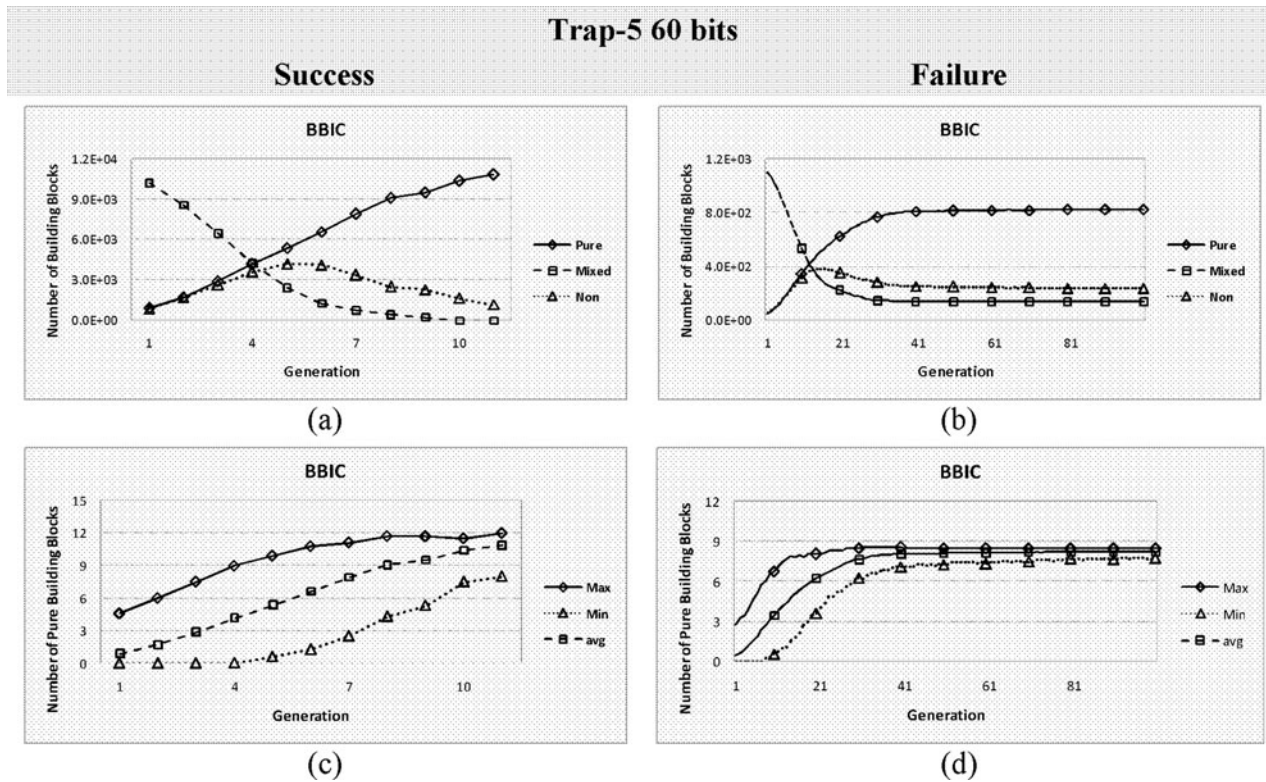


Figure 12. The BBIC algorithm results for the Trap-5 60-bit problem: (a) and (b) the number of BBs in each generation, and (c) and (d) the number of ideal BBs (pure BBs) in each generation.

failure case (Figure 13(b)), the average size of the different BBs grows slightly because there are several mixed BBs (Figure 9(b)). The number of BBs reduces rapidly until the 31th generation at which point rate of reduction slows down (Figure 13(d)). The rates of convergence of the success and failure cases are different.

For the deceptive problem, there was a prominent competition between the structures of good and deceptive solutions. In the success case (Figure 14(a)), the average size of different BBs develops progressively because of the deceptive bias.

For both the Royal Road and the Trap-5 problems, in the success case, the required size of the subpopulation in the  $n$ -Best selection was approximately 10% of the population size. This small number results in fast convergence from the restricted diversity. Furthermore, the desired solution can be achieved from the selected chromosomes. This evidence supports the belief that among the good chromosomes there are good substructures. If the good structures can be identified correctly, they can be used to produce good results. Finding a large common BB is more difficult than finding a small one. Therefore, the BBs in the Royal Road problem (8-bit BB) are harder to identify than the BBs in the Trap-5 function (5-bit BB), and thus the Royal Road problem requires a larger population size than the Trap-5 prob-

lem. The collation of all pairs in the BBI process provides a high degree of variation, and the recombination operation in the BBC process generates a great number of possible patterns for the chromosome structure.

The number of BBs in each position in the archive varies (see Figure 7). Normally, the first position has more members because the BBI process proceeds from left to right. When the BB size is bigger than a few bits, the next positions have fewer members because the next BBs have to be identified in sequence. This occurs repeatedly throughout the positions of the chromosome because the pattern of identification is from left to right.

### Benchmark and performance

In this section, several problems were used to test the performance of the BBIC algorithm. Most of them are BB validation problems. The results obtained using the BBIC algorithm are compared to the results obtained using several competing algorithms: the sGA, sGA-FC, the CSM, the BOA, and the hBOA. The benchmark problems are the One-Max, the Royal Road, the Trap-5, the hierarchical-if-and-only-if (HIFF) (Yu & Goldberg, 2006), and the hierarchical Trap-1 (hTrap-1) function (Yu, Golberg, Sasty, Lima, & Pelikan, 2009). The details of these problems can be found

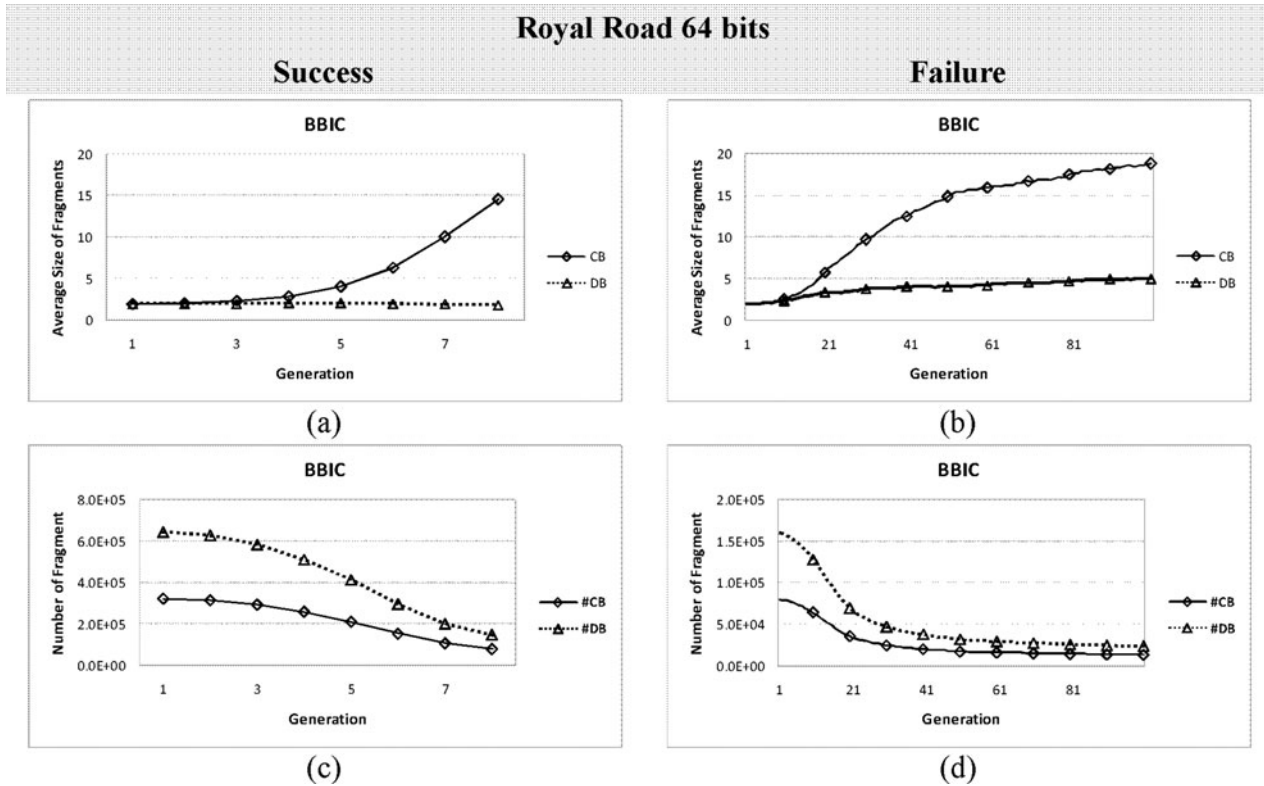


Figure 13 The BBIC algorithm results for the Royal Road 64-bit problem: (a) and (b) the average size of the BBs in each generation, and (c) and (d) the number of BBs in each generation.

Note: CB denotes the average common Building Block size, DB denotes the average different Building Block size, #CB denotes the number of common BBs, and #DB denotes the number of different BBs.

in the references, and their characteristics are classified (Figure 15). Only the OneMax problem is not a BB validation problem and is used as a performance reference for general cases. Both Trap-5 and hTrap-1 are deceptive problems, but Trap-5 has a straightforward BB structure, whereas hTrap-1 has a hierarchical BB structure. The HIFF problem is the only one that has a hierarchical BB structure and a multi-modal fitness landscape, which has two optimal solution structures. The HIFF and the hTrap-1 problems are hierarchical decomposable function (HDF) problems, which are harder to solve than ADF problems.

All of the benchmark problems were performed with 30 independent runs, and they were required to find the optimal solution in all of the runs. There are two versions of the BBIC algorithm: without mutation (BBIC-1) and with mutation, rate = 0.02, (BBIC-2). We want to study how mutations affect the BBs and how they act as a source of genetically diverse material in each problem. The minimum population size used to achieve the optimum in 30 runs is shown in Table 2. The average number of function evaluations is compared to the sGA, sGA-FC, CSM, BOA, and hBOA. The results are shown in Table 3. The results show that the BBIC-2 algorithm outperforms the BOA, hBOA,

and CSM in all of the problems, and the BBIC-1 algorithm outperforms all competitors in the Royal Road (except the 64-bit problem), the Trap-5, the HIFF, and the hTrap-1 problems. The performance of the BBIC-1 algorithm (without mutation) on the tested problems, ordered from high to low, is as follows: OneMax, Royal Road, HIFF, Trap-5, and hTrap-1. In the same way, the performance of the BBIC-2 algorithm (with mutation) is ranked in the following order: OneMax, Royal Road, HIFF, hTrap-1, and Trap-5.

There are two parameter settings in the BBIC algorithm as shown in Table 2. The population size indicates the level of diversity required to explore various solutions until the optimal solution is attained. The subpopulation denotes the level of selection pressure required to assure the quality of the result. The selection pressure is calculated as the size of the subpopulation (#sub) divided by the size of the population (#pop). A larger problem size or a harder problem requires that higher solution quality be obtained using a higher selection pressure. The diversity extension from the mutation process can compensate for the restricted variation of the population. The BBIC-2 algorithm requires a smaller population size and a smaller subpopulation size because of the mutation.



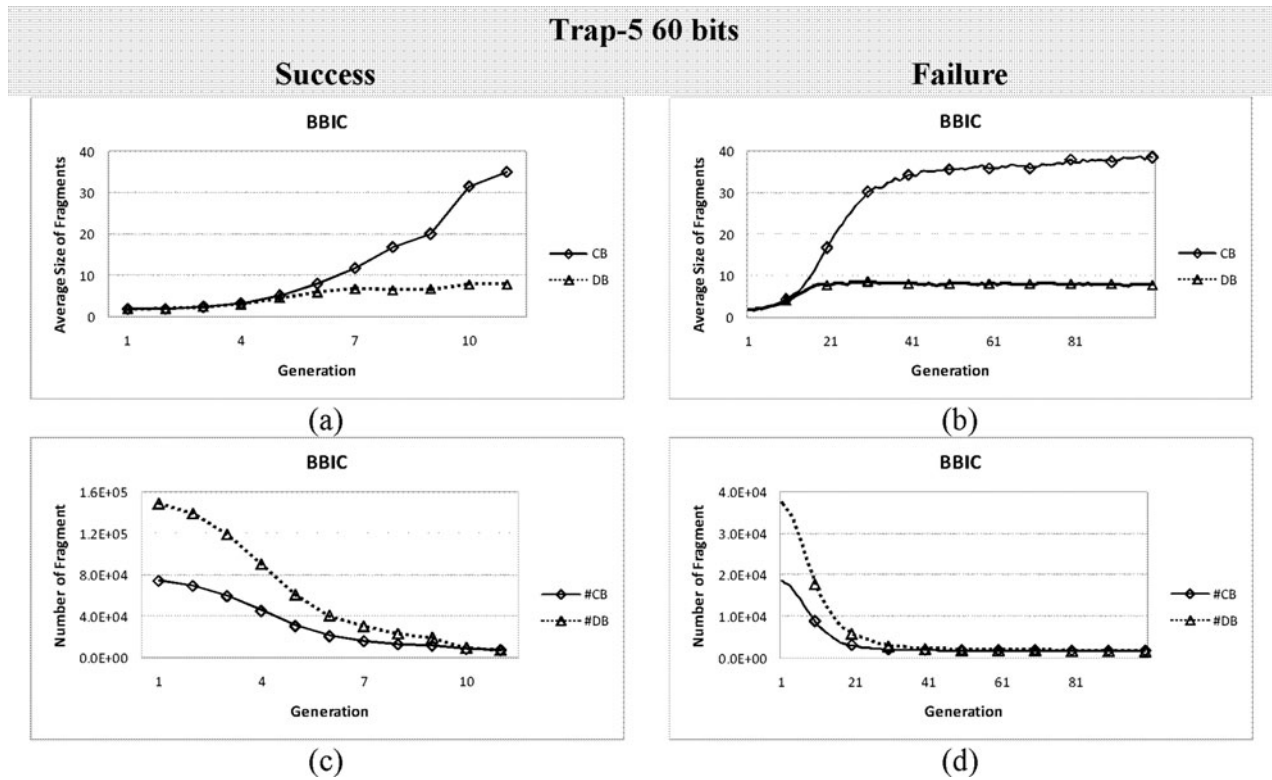


Figure 14 The BBIC algorithm results for the Trap-5 60-bit problem: (a) and (b) the average size of the BBs in each generation, and (c) and (d) the number of BBs in each generation.

Note: CB denotes the average common Building Block size, DB denotes the average different Building Block size, #CB denotes the number of common BBs, and #DB denotes the number of different BBs.

The results in Table 3 convince us that the proposed algorithm is suitable for ADF and HDF problems that are composed of tightly grouped BBs.

### Discussion

The main component of the BBIC algorithm is the BB, which will be summarised in this section. There are two

types of BBs: common and different. Many chromosomes in the selection process will be paired. The common BBs are regarded to have the same bias between two collated chromosomes. The identification procedure prevents the plausible unknown BBs from being disrupted. If the rate of construction of the BBs is higher than the rate of disruption of the BBs, the quality of the solutions will be improved.

Test Problems	Non-Deceptive		All Optimal bits 0/1	Straightforward	Hierarchical	Linear Fitness		Non-linear Fitness	Unimodal	Multimodal	BBs Test	Tightly Grouped BBs	Hierarchical BBs
	Non-Deceptive	Deceptive				Linear Fitness	Non-linear Fitness						
OneMax	•		•	•		•			•				
Royal Road	•		•	•		•			•		•	•	
Trap-5		•	•	•		•			•		•	•	
HIFF	•		•		•		•			•	•	•	•
hTrap-1		•	•		•		•		•		•	•	•

Figure 15. Characteristics of the benchmark problems.

Table 2. Benchmark parameter settings.

Problems	Problem size (bit)	BBIC-1 without mutation			BBIC-2 with mutation		
		#pop	#sub	(#sub/#pop)	#pop	#sub	(#sub/#pop)
OneMax	100	600	100	0.1666	<b>100</b>	<b>10</b>	<b>0.1000</b>
	150	1000	150	0.1500	<b>100</b>	<b>10</b>	<b>0.1000</b>
	200	2000	200	0.1000	<b>100</b>	<b>10</b>	<b>0.1000</b>
	250	3000	200	0.0666	<b>100</b>	<b>10</b>	<b>0.1000</b>
Royal Road	<b>64</b>	2000	<b>200</b>	<b>0.1000</b>	<b>1500</b>	<b>200</b>	0.1333
	128	3500	<b>200</b>	<b>0.0571</b>	<b>2500</b>	<b>200</b>	0.0800
	256	5500	<b>300</b>	<b>0.0545</b>	<b>4000</b>	<b>300</b>	0.0750
Trap-5	100	2000	<b>200</b>	<b>0.1000</b>	<b>1100</b>	<b>200</b>	0.1818
	150	2500	<b>200</b>	<b>0.0800</b>	<b>2000</b>	<b>200</b>	0.1000
	200	4500	<b>200</b>	<b>0.0444</b>	<b>3500</b>	<b>200</b>	0.0571
	250	8000	<b>250</b>	<b>0.0312</b>	<b>6700</b>	<b>200</b>	0.0299
HIFF	32	<b>200</b>	100	0.5000	<b>200</b>	<b>50</b>	<b>0.2500</b>
	<b>64</b>	<b>500</b>	<b>100</b>	0.3000	<b>500</b>	<b>100</b>	<b>0.2000</b>
	128	2000	<b>200</b>	<b>0.1000</b>	<b>1000</b>	<b>200</b>	0.2000
	256	6500	400	0.0615	<b>6000</b>	<b>300</b>	<b>0.0500</b>
hTrap-1	<b>27</b>	300	<b>100</b>	<b>0.3333</b>	<b>200</b>	<b>100</b>	0.5000
	81	<b>1000</b>	<b>100</b>	0.2000	<b>1000</b>	<b>100</b>	<b>0.1000</b>
	243	11,500	<b>300</b>	<b>0.0260</b>	<b>10,000</b>	<b>300</b>	0.0300

Note: #pop means population and #sub means subpopulation or selected chromosomes from the  $n$ -Best selection method. The bold face indicates the minimum value.

Table 3. Benchmark results.

Problems	Problem size (bit)	Number of function evaluations (#FEs)						
		BBIC-1	BBIC-2	sGA	sGA-FC	CSM	BOA	hBOA
OneMax	100	7500	<b>1700</b>	22,200	3800	14,000	5100	N/A
	150	15,000	<b>2600</b>	64,300	6600	32,500	8300	N/A
	200	32,000	<b>3400</b>	128,300	9800	60,000	12,500	N/A
	250	48,500	<b>4600</b>	271,200	11,500	80,000	15,700	N/A
Royal Road	64	13,400	15,100	<b>11,900</b>	13,500	N/A	N/A	N/A
	128	<b>28,700</b>	31,800	49,500	29,900	N/A	N/A	N/A
	256	<b>73,300</b>	89,300	304,400	75,200	N/A	N/A	N/A
Trap-5	100	<b>21,200</b>	35,000	83,250	47,900	65,000	99,000	N/A
	150	<b>31,300</b>	73,000	305,500	114,000	165,000	220,000	N/A
	200	<b>56,400</b>	124,000	784,900	215,600	310,000	320,000	N/A
	250	<b>102,900</b>	219,000	N/A	375,900	750,000	490,000	N/A
HIFF	32	<b>1100</b>	1200	4800	2600	3300	N/A	2100
	64	<b>4700</b>	4900	38,800	11,800	14,500	N/A	7800
	128	<b>17,500</b>	21,000	584,000	45,900	51,000	N/A	27,000
	256	<b>71,300</b>	118,000	N/A	222,700	370,000	N/A	90,000
hTrap-1	27	<b>2100</b>	2800	11,400	8000	3000	N/A	3400
	81	<b>8900</b>	9500	N/A	N/A	35,000	N/A	30,000
	243	<b>115,000</b>	178,000	N/A	N/A	310,000	N/A	225,000

Note: N/A denotes that the data were not available because the optimal solution cannot be found under the limited number of function evaluations (#FEs  $\leq$  1,000,000). The bold face indicates the best value.

This is the main mechanism of the evolutionary process. The results of sGA-FCs and EDAs in several published papers confirm that various types of BBI processes are useful (Larrañaga & Lozano, 2001; Pelikan, Goldberg, & Lobo, 2002; Pelikan et al., 2006). In regard to the use of explicit BBs in GAs, the mGA and its variants (Goldberg et al., 1993; Goldberg et al., 1989) are directly related to our work. The method of mGAs is to improve the performance by increasingly build longer, highly fit strings from shorter BBs. This is similar to how we compose BBs. However, our method is different in the identification of BBs and how BBs are stored and used.

The size of the BBs gives indirect information about the level of knowledge or diversity. If the size of the common BBs is longer, the size of the different BBs is shorter. This means that the evolution process has learned something about the models. The different BBs preserve the unexploited structures or diversity of the search space. In the early generations, the average size of the BBs is small because the diversity is high. Although the common BBs in the early stage are not reliable, the number of different BBs is twice the number of common BBs. The different BBs act as the choices for search space exploration. In the middle generations, the common BBs are more stable and more reliable. The different BBs act as the more distinct and more limited alternatives. In the later generations, the different BBs work as rare mutations because they are too short, and thus they have a very low chance to be selected.

## Conclusion

The contiguous substructures of a chromosome can be regarded as the BBs. They are identified from the mutual data between two chromosomes. The proposed BBs are a simple form of the explicit BBs because they are short, low order and come from the highly fit chromosomes. The BBIC algorithm uses the centralised knowledge, similar to EDAs, that all of the BBs are retained in an archive to create new offspring. The BBC process is simple and direct: it proceeds from left to right (first bit to last bit) to form a new chromosome using random selection from the archive.

The experimental results of the BBIC algorithm confirm that the identification of BBs is an important process that guides the recombination procedure to improve the solutions. The execution time of the BBI and BBC processes is  $O((n)^2 l)$  and  $O(n)$ , respectively, where  $l$  is the chromosome length and  $n$  is the number of selected chromosomes. This is significantly less than the execution time of the CSM, BOA, and hBOA. The proposed method is simple to implement and easy to tune. In addition, this method efficiently solves difficult problems of both the ADF and HDF classes.

## Notes on contributors



**Chalermsub Sangkavichitr** earned BEng degree in electrical engineering from Kasetsart University, Thailand in 2000 and MSc degree in computer science from Chulalongkorn University, Thailand in 2003. Presently, he is a PhD candidate in the Department of Computer Engineering, Chulalongkorn University.



**Prabhas Chongstitvatana** earned BEng degree in electrical engineering from Kasetsart University, Thailand in 1980 and PhD degree from the Department of Artificial Intelligence, Edinburgh University, UK in 1992. Presently, he is a professor in the Department of Computer Engineering, Chulalongkorn University. His research included robotics, evolutionary computation,

and computer architecture. The current work involves bioinformatics and grid computing. He actively promotes the collaboration to create Thai national grid for scientific computing. He is the member of Thailand Engineering Institute, Thai Academy of Science and Technology, Thai Robotics Society, Thai Embedded System Association, ECTI Association of Thailand, and IEEE Robotics and Automation Society.

## References

- Alabsi, F., & Naoum, R. (2012). Fitness function for genetic algorithm used in intrusion detection system. *International journal of Applied Science and Technology*, 2, 129–134.
- Aporntewan, C., & Chongstitvatana, P. (2004). *Chi-square matrix: An approach for building-block identification* (pp. 63–77). Proceedings of Asian Computing Science Conference, Chiang Mai, Thailand.
- Aporntewan, C., & Chongstitvatana, P. (2005). *A quantitative approach for validating the building-block hypothesis* (pp. 1403–1409). Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), Edinburgh, Scotland.
- Beaudooin, W., Verel, S., Collard, P., & Escaput, C. (2006). *Deceptiveness and neutrality the ND family of fitness landscapes* (pp. 507–514). Proceedings of Genetic and Evolutionary Computation Conference 2006 (GECCO-2006), Seattle, WA.
- Chen, Y.-P. (2006). *Extending the scalability of linkage learning genetic algorithms*. New York, NY: Springer.
- Chen, Y.-P., & Goldberg, D.E. (2002). *Introducing start expression genes to the linkage learning genetic algorithm* (pp. 351–360). Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN VII), Granada, Spain.
- Chen, Y.P., & Lim, M.-H. (2008). *Linkage in evolutionary computation*. New York, NY: Springer.
- Chen, Y.-P., Yu, T.-L., Sastry, K., & Goldberg, D.E. (2007). *A survey of linkage learning techniques in genetic and evolutionary algorithms* (Tech. Rep. TR-2007014). Urbana: University of Illinois.
- Cheung, T., Cheung, N., Tobar, C.M.T., Caram, R., & Garcia, A. (2011). Application of a genetic algorithm to optimize purification in the zone refining process. *Materials and Manufacturing Processes*, 26, 493–500.
- Coffin, D.J., & Smith, R.E. (2008). Linkage learning in estimation of distribution algorithms. In Y.P. Chen & M.-H. Lim

- (Eds.), *Linkage in evolutionary computation* (pp. 141–156). New York, NY: Springer.
- Collard, P., Gaspar, A., Clergue, M., & Esczut, C. (1998). *Fitness distance correlation as statistical measurement of genetic algorithms difficulty, revisited* (pp. 650–654). Proceedings of the European Conference on Artificial Intelligence, Brighton.
- Eiben, A.E. (2000). Multiparent recombination. In T. Baeck, D.B. Fogel, & Z. Michalewicz (Eds.), *Evolutionary computation 1: Basic algorithms and operators* (pp. 289–307). New York, NY: Taylor & Francis.
- Eiben, A.E. (2002). Multiparent recombination in evolutionary computing. In A. Ghosh & S. Tsutsui (Eds.), *Advances in evolutionary computing* (pp. 175–192). New York, NY: Springer.
- Eiben, A.E., Raué, P.-E., & Ruttkay, Zs. (1994). *Genetic algorithms with multi-parent recombination* (pp. 78–87). Proceedings of the International Conference on Evolutionary Computation, Jerusalem, Israel.
- Eiben, A.E., & Smit, S.K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1, 19–31.
- Finger, M., Stutzle, T., & Lourenco, H. (2002). *Exploiting fitness distance correlation of set covering problems* (pp. 61–67). Proceedings of the Application of Evolutionary Computing on EvoWorkshops, Kinsale, Ireland.
- Goldberg, D.E. (1989). *Genetic algorithms in search, optimization & machine learning*. Boston, MA: Addison-Wesley Longman Publishing.
- Goldberg, D.E., Deb, K., Kargupta, H., & Harik, G. (1993). *Rapid, accurate optimization of difficult problems using fast messy genetic algorithms* (pp. 56–64). Proceedings of the International Conference on Genetic Algorithms, Urbana-Champaign, IL.
- Goldberg, D.E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3, 493–530.
- Goldberg, D.E., & Sastry, K. (2010). *Genetic algorithms: The design of innovation*. Berlin: Springer.
- Kargupta, H. (1996). *The gene expression messy genetic algorithm* (pp. 814–819). Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan.
- Howard, B., & Sheppard, J.W. (2004). *The Royal Road not taken: A re-examination of the reasons for ga failure on RI* (pp. 1208–1219). Proceeding of Genetic and Evolutionary Computation Conference (GECCO-2004), Seattle, WA.
- Larrañaga, P., & Lozano, J.A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston, MA: Kluwer Academic Publishers.
- Levenick, J.R. (1995). *Metabits: Generic endogenous crossover control* (pp. 88–95). Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95), Pittsburgh, PA.
- Lobo, F.G., Lima, C.F., & Michalewicz, Z. (2007). *Parameter setting in evolutionary algorithms*. Berlin: Springer.
- Ozcan, T., & Esnaf, S. (2013). A discrete constrained optimization using genetic algorithms for a bookstore layout. *International Journal of Computational Intelligence Systems*, 6, 261–278.
- Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Berlin: Springer.
- Pelikan, M., Goldberg, D.E., & Lobo, F.G. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21, 5–20.
- Pelikan, M., Sastry, K., & Cantú-Paz, E. (2006). *Scalable optimization via probabilistic modeling*. Berlin: Springer.
- Sangkavichitra, C., & Chongstitvatana, P. (2010). Fragment as a small evidence of the building blocks existence. In Y.P. Chen (Ed.), *Exploitation of linkage learning in evolutionary algorithms* (pp. 25–44). New York, NY: Springer.
- Schaefer, R. (2007). *Foundations of global genetic optimization*. New York, NY: Springer.
- Sivanandam, S.N., & Deepa, S.N. (2008). *Introduction to genetic algorithms*. New York, NY: Springer.
- Smit, S.K., & Eiben, A.E. (2010). Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In C. Di Chio A. Brabazon, G. Di Caro, M. Ebner, M. Farooq, A. Fink, ... N. Urquhart (Eds.), *Applications of evolutionary computation* (pp. 542–551). New York, NY: Springer.
- Smith, J., & Fogarty, T.C. (1995). *An adaptive poly-parental recombination strategy* (pp. 48–61). Proceedings of AISB-95 Workshop on Evolutionary computing, Sheffield.
- Syswerda, G. (1993). Simulated crossover in genetic algorithms. In L.D. Whitley (Ed.), *Foundations of genetic algorithms 2* (pp. 239–255). San Francisco, CA: Morgan Kaufmann.
- Ting, C.K., & Chen, C.C. (2007). *The effects of supermajority on multi-parent crossover* (pp. 4524–4530). Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), Singapore.
- Toussaint, M. (2003). *The structure of evolutionary exploration: On crossover, building blocks, and estimation-of-distribution algorithms* (pp. 1444–1456). Proceeding of Genetic and Evolutionary Computation Conference (GECCO-2003), Chicago, IL.
- Wang, K. (2009). Application of genetic algorithms to robot kinematics calibration. *International Journal of Systems Science*, 40, 147–153.
- Yu, T.-L., & Goldberg, D.E. (2006). *Conquering hierarchical difficulty by explicit chunking: Substructural chromosome compression* (pp. 1385–1392). Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006), Seattle, WA.
- Yu, T.-L., Goldberg, D.E., Sastry, K., Lima, C.F., & Pelikan, M. (2009). Dependency structure matrix, genetic algorithms, and effective recombination. *Evolutionary Computation*, 17, 595–626.
- Yu, X., & Gen, M. (2010). *Introduction to evolutionary algorithms*. Berlin: Springer.