Simultaneity Matrix for Solving Hierarchically Decomposable Functions

Chatchawit Aporntewan and Prabhas Chongstitvatana

Chulalongkorn University, Bangkok 10330, Thailand
Chatchawit.A@student.chula.ac.th Prabhas.C@chula.ac.th

Abstract. The simultaneity matrix is an $\ell \times \ell$ matrix of numbers. It is constructed according to a set of ℓ-bit solutions. The matrix element m_{ij} is the degree of linkage between bit positions i and j. To exploit the matrix, we partition $\{0, \ldots, \ell - 1\}$ by putting i and j in the same partition subset if m_{ij} is significantly high. The partition represents the bit positions of building blocks (BBs). The partition is used in solution recombination so that the bits governed by the same partition subset are passed together. It can be shown that by exploiting the simultaneity matrix the hierarchically decomposable functions can be solved in a polynomial relationship between the number of function evaluations required to reach the optimum and the problem size. A comparison to the hierarchical Bayesian optimization algorithm (hBOA) is made. The hBOA uses less number of function evaluations than that of our algorithm. However, computing the matrix is 10 times faster and uses 10 times less memory than constructing Bayesian network.

1 Introduction

For some conditions [6, Chapter 7–11], the success of genetic algorithms (GAs) can be explained by the schema theorem and the building-block hypothesis [4]. The schema theorem states that the number of solutions that match the above average, short defining-length, and low-order schemata grows exponentially. The optimal solution is hypothesized to be composed of the above average schemata or the building blocks (BBs). However, in simple GAs only short defining-length and low-order schemata are permitted to the exponential growth. The other schemata are more disrupted due to the single-point crossover. When the good BBs are more disrupted, it is said to be a GA-hard problem. Trap function [1] is an adversary function for studying BBs and linkage problems in GAs [7]. The general k-bit trap functions are defined as:

$$F_k(b_0 \ldots b_{k-1}) = \begin{cases} f_{\text{high}} & ; \text{ if } u = k \\ f_{\text{low}} - u\frac{f_{\text{low}}}{k-1} & ; \text{ otherwise,} \end{cases} \tag{1}$$

where $b_i \in \{0, 1\}$, $u = \sum_{i=0}^{k-1} b_i$, and $f_{\text{high}} > f_{\text{low}}$. Usually, f_{high} is set at k and f_{low} is set at $k - 1$. The additively decomposable functions (ADFs), denoted by $F_{m \times k}$, are defined as:

$$F_{m\times k}(B_0 \dots B_{m-1}) = \sum_{i=0}^{m-1} F_k(B_i),\ B_i \in \{0,1\}^k. \tag{2}$$

The m and k are varied to produce a number of test functions. The ADFs fool gradient-based optimizers to favor zeroes, but the optimal solution is composed of all ones. Trap function is a fundamental unit for designing test functions that resist hill-climbing algorithms. The test functions can be effectively solved by composing BBs. Several discussions of the test functions can be found in [9,21, 22].

The BBs are inferred from a population of highly-fit individuals [6, pp. 60–61]. A population of highly-fit individuals (5×3-trap function) is shown in Table 1. The dependency between variables b_i, b_{i+1}, b_{i+2} $(i = 0, 3, 6, 9, 12)$ can be detected by means of a statistical method. An inference might be that the highly-fit individuals are composed of triple zeroes and triple ones. It is said that the triple zeroes and triple ones are common traits or BBs. We aim to identify these BBs.

Table 1. A population of highly-fit individuals (5×3-trap function)

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	111	000	111	000	13.0
2	000	000	111	000	111	12.0
3	111	000	000	111	000	12.0
4	000	000	000	000	111	11.0
5	000	000	000	000	000	10.0

Thierens raised the scalability issue of simple GAs [20]. He used the uniform crossover so that the solutions are randomly mixed. The objective function is the $m\times 5$-trap functions. The analysis shows that either the computational time grows exponentially with the number of 5-bit trap functions or the population size must be exponentially increased. It is clear that scaling up the problem size requires information about the BBs so that the solutions are efficiently mixed. In addition, the performance of simple GAs relies on the ordering of solution bits. The ordering may not pack the dependent bits close together. Such an ordering results in poor mixing. Therefore the BBs need to be identified to improve the scalability issue.

Many strategies in the literature use the bit-reordering approach to pack the dependent bits close together, for example, inversion operator [4], messy GAs [5], and linkage learning [7]. The bit-reordering approach does not explicitly identify BBs, but it successfully delivers the optimal solution. Several works explicitly identify BBs. An approach is to find a partition of bit positions. For instance, Table 1 infers the partition:

$$\{\{0,1,2\},\{3,4,5\},\{6,7,8\},\{9,10,11\},\{12,13,14\}\}. \tag{3}$$

In the case of nonoverlapped BBs, partition is a clear representation [8,11,12,13]. Note that Kargupta [12] computes Walsh's coefficients which imply the partition. The bits governed by the same partition subset are passed together to prevent BB disruption.

Identifying BBs is somewhat related to building a distribution of solutions [8,14,15]. The basic concept of optimization by building a distribution is to start with a uniform distribution of solutions. Next, a number of solutions is drawn according to the distribution. Some good solutions (winners) are selected, and the distribution is adjusted toward the winners (the winners-like solutions will be drawn with higher probability in the next iteration). These steps are repeated until the optimal solution is found or reaching a termination condition. The works in this category are referred to as *probabilistic model-building genetic algorithms (PMBGAs)*. For a particular form of distribution used in the extended compact genetic algorithm (ECGA), building the distribution is identical to searching for a partition [8]. The Bayesian optimization algorithm (BOA) uses Bayesian network to represent a distribution [14]. Pelikan showed that if the problem is composed of k-bit trap functions, the network will be fully connected sets of k nodes [17, pp. 54]. In addition, the Bayesian network is able to represent joint distributions in the case of overlapping BBs. The hierarchical BOA (hBOA) is the BOA enhanced with decision tree/graph and a niching method called restricted tournament replacement [17]. The hBOA can solve the hierarchically decomposable functions (HDFs) in a scalable manner [17]. Successful applications for BB identification are financial applications [10], cluster optimization [19], maximum satisfiability of logic formulas (MAXSAT) and Ising spin glass systems [18].

The Bayesian network is able to identify common structures in a population. Nevertheless, building the network is time-consuming. This paper presents a BB identification algorithm that is simpler and faster than that of the hBOA. In addition, our algorithm uses less memory. The algorithm is named building-block identification by simultaneity matrix (BISM) [2]. The BISM input is a set of ℓ-bit solutions. The BISM output is a partition of $\{0, \ldots, \ell-1\}$. Algorithm BISM consists of two parts: simultaneity matrix construction (SMC) and partitioning (PAR) algorithms. The SMC constructs the matrix according to a set of solutions. Next, PAR searches for a partition for the matrix. The remainder of the paper is organized as follows. Section 2 defines the hierarchically decomposable functions. Section 3 describes the SMC algorithm. Section 4 describes the PAR algorithm. Section 5 presents the experimental results and discussions. Section 6 concludes the paper.

2 Hierarchically Decomposable Functions

To solve ADFs, the BBs need to be identified so that the solutions are efficiently mixed. The hierarchically decomposable functions (HDFs) are far more difficult than the ADFs. First, BBs in the lowest level need to be identified. The solution quality is improved by exploiting the identified BBs in solution recombination. Next, the improved population reveals larger BBs. Again the BBs in higher levels need to be identified. Identifying and exploiting BBs are repeated many times until reaching the optimal solution. Commonly used HDFs are hierarchically if-and-only-if (HIFF), hierarchical trap 1 (HTrap1), and hierarchical trap 2 (HTrap2) functions. Due to page limitations, the original definitions of HDFs can be found in [21,17].

3 Simultaneity Matrix Construction (SMC) Algorithm

The SMC input is a set of ℓ-bit binary string denoted by:

$$S = \{s_0, \ldots, s_{n-1}\}, \tag{4}$$

where s_i is the i^{th} string, $0 \leq i \leq n-1$. The $s_i[j]$ denotes the j^{th} bit of s_i, $0 \leq j \leq \ell - 1$. Algorithm SMC outputs an $\ell \times \ell$ symmetric matrix of numbers, denoted by $M = (m_{ij})$, $0 \leq i, j \leq \ell - 1$. A closed form of m_{ij} is shown in Equation 5.

$$m_{ij} = \begin{cases} 0 & ;\ \text{if } i = j \\ \text{Count}_S^{00}(i,j)\text{Count}_S^{11}(i,j) + \text{Count}_S^{01}(i,j)\text{Count}_S^{10}(i,j) & ;\ \text{otherwise,} \end{cases} \tag{5}$$

where $\text{Count}_S^{ab}(i,j) = |\{x \in \{0,\ldots,n-1\} : s_x[i] = a \text{ and } s_x[j] = b\}|$ for all $0 \leq i, j \leq \ell - 1$, $(a,b) \in \{0,1\}^2$.

Algorithm SMC is shown in Figure 1. Step 1 constructs only the upper triangle of the matrix by using Equation 5. Step 2 perturbs the matrix so that there are no identical elements. This matrix, in which all the elements are distinct, is greatly helpful in partitioning. The perturbation does not totally change the matrix because each element is incremented by a small real random number ranging between 0 and 1. The perturbation by adding an integer with a real number is practical for a random number generator with a sufficiently large period because it is hardly possible to produce identical random numbers. Step 3 copies the upper triangle $\{m_{ij} \mid i < j\}$ to the lower triangle $\{m_{ij} \mid i > j\}$. Step 4 returns the simultaneity matrix $M = (m_{ij})$. The time complexity of SMC is $O(\ell^2 n)$.

The matrix element m_{ij} is proportional to the probability that 2-bit BBs at bit positions i and j will be disrupted by the uniform crossover. All cases for mixing 2-bit BBs are enumerated. Mixing "00" with "11" results in "01" and "10." Mixing "01" with "10" results in "00" and "11." Only mixing in the two cases must be done carefully because the processing BBs will be lost. Mixing 2-bit BBs in the other cases gives the same BBs. Therefore Algorithm SMC counts a pair of 2-bit BBs that are complement to each other. To exploit the matrix, the bits at positions i and j are passed together every time performing crossover if the matrix element m_{ij} is significantly high. The 3-bit BBs are identified by inserting k to $\{i, j\}$. If the matrix elements m_{ij}, m_{jk}, and m_{ik} are significantly high, i, j, k should be in the same partition subset. Larger BBs can be identified in a similar fashion.

The trap functions embedded in the HDFs bias the population to two aligned chunks of zeroes and ones, that are complementary to each other. Certainly, the dependency between every pair of bits in a chunk is stored in the matrix. The matrix is not limited to the cases where the two aligned chunks are complementary

```
Algorithm SMC($S$)
1. for $i = 0$ to $\ell - 1$ do
       $m_{ii} \leftarrow 0$;
       for $j = i + 1$ to $\ell - 1$ do
           $m_{ij} \leftarrow \mathrm{Count}_S^{00}(i,j) \times \mathrm{Count}_S^{11}(i,j) + \mathrm{Count}_S^{01}(i,j) \times \mathrm{Count}_S^{10}(i,j)$;
2. for $i = 0$ to $\ell - 1$ do
       for $j = i + 1$ to $\ell - 1$ do
           $m_{ij} \leftarrow m_{ij} + Random(0,1)$;
3. for $i = 0$ to $\ell - 1$ do
       for $j = i + 1$ to $\ell - 1$ do
           $m_{ji} \leftarrow m_{ij}$;
4. return $M = (m_{ij})$;
```

Fig. 1. SMC algorithm

to each other. In the other cases, the matrix does not detect unnecessary dependency. For instance, the bits at positions of $\{0, 1, 2, 3, 4\}$ are mostly "b_0b_1000" and "b_0b_1111" where $b_i \in \{0, 1\}$. The dependency among five bits is obvious, but passing the bits governed by $\{2, 3, 4\}$ together is sufficient to guarantee that "b_0b_1000" and "b_0b_1111" will exist in the next generation with a high probability. In summary, the matrix records only dependency that is actually necessary for preserving BBs.

4 Partitioning (PAR) Algorithm

The PAR input is an $\ell \times \ell$ simultaneity matrix. The PAR outputs the partition:

$$P = \{B_0, \ldots, B_{|P|-1}\}, \ \bigcup_{i=0}^{|P|-1} B_i = \{0, \ldots, \ell-1\}, \ B_i \cap B_j = \emptyset \text{ for all } i \neq j. \quad (6)$$

The B_i is called partition subset. There are several definitions of the desired partition, for example, the definitions in the senses of nonmonotonicity [13], GEMGA [11], Walsh coefficients [12], and entropy measurement [8]. We develop a definition in the sense of simultaneity matrix. Algorithm PAR searches for a partition P such that

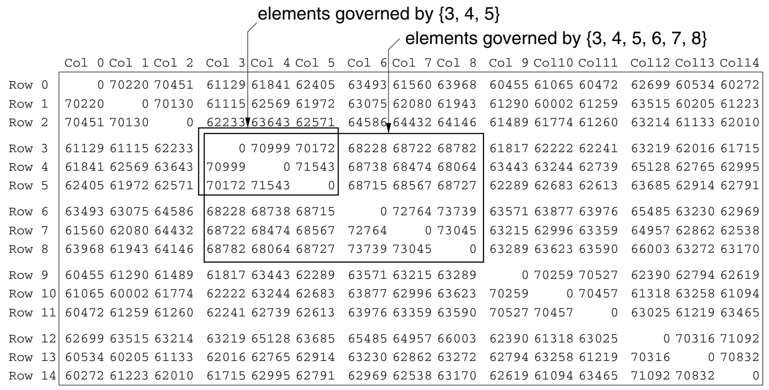
1. $P \neq \{\{0, \ldots, \ell - 1\}\}$.
2. For all $B \in P$ such that $1 < |B| < \ell$, for all $b \in B$, the largest $|B| - 1$ matrix elements in row b are founded in columns of $B \setminus \{b\}$.
3. For all $B \in P$ such that $1 < |B| < \ell$, $H_{max} - H_{min} < \alpha(H_{max} - L_{min})$ where $\alpha \in [0, 1]$,
   $H_{max} = max(m_{ij} \mid (i,j) \in B^2, \ i \neq j)$,
   $H_{min} = min(m_{ij} \mid (i,j) \in B^2, \ i \neq j)$,
   $L_{min} = min(m_{ij} \mid i \in B, \ j \in \{0, \ldots, \ell - 1\} \setminus B)$.

4. There are no partition P_x such that for some $B \in P$, for some $B_x \in P_x$, P and P_x satisfy the first, the second, and the third conditions, $B \subset B_x$.

An example of the simultaneity matrix is shown in Figure 2. The perturbation is omitted because the values of $\{m_{ij} \mid i < j\}$ are distinct. The first condition does not allow the coarsest partition because it is not useful in solution recombination. The second condition makes i and j, in which m_{ij} is significantly high, in the same partition subset. For instance, $P_1 = \{\{0,1,2\}, \{3,4,5\}, \{6,7,8\}, \{9,10,11\}, \{12,13,14\}\}$ satisfies the second condition because the largest two elements in row 0 are found in columns of $\{1,2\}$, the largest two elements in row 1 are found in columns of $\{0,2\}$, the largest two elements in row 2 are found in columns of $\{0,1\}$, and so on. However, there are many partitions that satisfy the second condition, for example, $P_2 = \{\{0,1,2\}, \{3,4,5,6,7,8\}, \{9,10,11\}, \{12,13,14\}\}$. There is a dilemma between choosing the fine partition (P_1) and the coarse partition (P_2). Choosing the fine partition prevents the emergence of large BBs, while the coarse partition results in poor mixing. To overcome the dilemma, the coarse partition will be acceptable if it satisfies the third condition. The fourth condition says choosing the coarsest partition that is consistent with the first, the second, and the third conditions.

By the third condition, the partition subset $\{3,4,5\}$ is acceptable because the values of matrix elements governed by $\{3,4,5\}$ are close together (see Figure 2). Being close together is defined by $H_{max} - H_{min}$ where H_{max} and H_{min} is the maximum and the minimum of the nondiagonal matrix elements governed by a partition subset. The $H_{max} - H_{min}$ is a degree of irregularities of the matrix. The main idea is to limit $H_{max} - H_{min}$ to a threshold. The threshold, $\alpha(H_{max} - L_{min})$, is defined relatively to the matrix elements because the threshold cannot be fixed for a problem instance. The partition subset $\{3,4,5\}$ gives $H_{max} = 71543$, $H_{min} = 70172$, and $L_{min} = 61115$. L_{min} is the minimum of the nondiagonal matrix elements in rows of $\{3,4,5\}$. The third condition limits $H_{max} - H_{min}$ to $100 \times \alpha$ percent of the difference between H_{max} and L_{min}. An empirical study showed that α should be set at 0.75 for both ADFs and HDFs. Choosing $\{3,4,5,6,7,8\}$ yields ($H_{max} = 73739, H_{min} = 68064, L_{min} = 61115$) which does not violate the third condition. The fourth condition prefers a coarse partition $\{\{3,4,5,6,7,8\},\ldots\}$ to a fine partition $\{\{3,4,5\},\ldots\}$ so that the partition subsets can be grown to compose larger BBs in higher levels.

Algorithm PAR is shown in Figure 3. A trace of the algorithm is shown in Table 2. The outer loop processes row 0 to $\ell - 1$. In the first step, the columns of the sorted values in row i are stored in array R. For $i = 0$, array $R[\ ] = \{2,1,8,6,12,5,4,7,3,10,13,11,9,14,0\}$. Next, the inner loop tries a number of partition subsets by enlarging B_1 ($B_1 \leftarrow B_1 \cup \{R[j]\}$). If B_1 satisfies the second and the third conditions, B_1 will be saved to B_2. Finally, P is the partition that satisfies the four conditions. Checking the second and the third conditions is the most time-consuming section. It can be done in $O(\ell^2)$. The checking is done at most ℓ^2 times. Therefore the time complexity of PAR is $O(\ell^4)$.


elements governed by {3, 4, 5}
elements governed by {3, 4, 5, 6, 7, 8}

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col10	Col11	Col12	Col13	Col14
Row 0	0	70220	70451	61129	61841	62405	63493	61560	63968	60455	61065	60472	62699	60534	60272
Row 1	70220	0	70130	61115	62569	61972	63075	62080	61943	61290	60002	61259	63515	60205	61223
Row 2	70451	70130	0	62233	63643	62571	64586	64432	64146	61489	61774	61260	63214	61133	62010
Row 3	61129	61115	62233	0	70999	70172	68228	68722	68782	61817	62222	62241	63219	62016	61715
Row 4	61841	62569	63643	70999	0	71543	68738	68474	68064	63443	63244	62739	65128	62765	62995
Row 5	62405	61972	62571	70172	71543	0	68715	68567	68727	62289	62683	62613	63685	62914	62791
Row 6	63493	63075	64586	68228	68738	68715	0	72764	73739	63571	63877	63976	65485	63230	62969
Row 7	61560	62080	64432	68722	68474	68567	72764	0	73045	63215	62996	63359	64957	62862	62538
Row 8	63968	61943	64146	68782	68064	68727	73739	73045	0	63289	63623	63590	66003	63272	63170
Row 9	60455	61290	61489	61817	63443	62289	63571	63215	63289	0	70259	70527	62390	62794	62619
Row 10	61065	60002	61774	62222	63244	62683	63877	62996	63623	70259	0	70457	61318	63258	61094
Row 11	60472	61259	61260	62241	62739	62613	63976	63359	63590	70527	70457	0	63025	61219	63465
Row 12	62699	63515	63214	63219	65128	63685	65485	64957	66003	62390	61318	63025	0	70316	71092
Row 13	60534	60205	61133	62016	62765	62914	63230	62862	63272	62794	63258	61219	70316	0	70832
Row 14	60272	61223	62010	61715	62995	62791	62969	62538	63170	62619	61094	63465	71092	70832	0

Fig. 2. Simultaneity matrix

Table 2. A trace of the PAR algorithm

i	j	B_1	2nd cond.	3rd cond.	B_2
0	0	{0, 2}	True	True	{0, 2}
0	1	{0, 2, 1}	True	True	{0, 1, 2}
0	2	{0, 2, 1, 8}	False	False	{0, 1, 2}
0	3	{0, 2, 1, 8, 6}	False	False	{0, 1, 2}
0	4	{0, 2, 1, 8, 6, 12}	False	False	{0, 1, 2}
0	5	{0, 2, 1, 8, 6, 12, 5}	False	False	{0, 1, 2}
0	6	{0, 2, 1, 8, 6, 12, 5, 4}	False	False	{0, 1, 2}
0	7	{0, 2, 1, 8, 6, 12, 5, 4, 7}	False	False	{0, 1, 2}
0	8	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3}	False	False	{0, 1, 2}
0	9	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10}	False	False	{0, 1, 2}
0	10	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10, 13}	False	False	{0, 1, 2}
0	11	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10, 13, 11}	False	False	{0, 1, 2}
0	12	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10, 13, 11, 9}	False	False	{0, 1, 2}

5 Experimental Results

5.1 Methodology

Most papers report the performance in terms of function evaluations required to reach the optimum. Such a performance measurement is affected by selection method, solution recombination, and the other factors. At present, research community does not provide a formal framework for measuring the effectiveness of a BB identification algorithm regardless of the other factors we have mentioned. Inevitably, we have to make a comparison in terms of function evaluations. We have presented the building-block identification by simultaneity matrix (BISM). An optimization algorithm that exploits the BISM is needed. We customize simple GAs as follows. Every generation, the simultaneity matrix is constructed.

```
Algorithm PAR($M$)
$P \leftarrow \emptyset$;
for $i = 0$ to $\ell - 1$ do
    if $i \notin B$ for all $B \in P$ then
        array $T$ = {matrix elements in row $i$ sorted in descending order};
        for $j = 0$ to $\ell - 1$ do
            $R[j] = x$ where $m_{ix} = T[j]$;
        endfor
        $B_1 \leftarrow \{i\}$;
        $B_2 \leftarrow \{i\}$;
        for $j = 0$ to $\ell - 3$ do
            $B_1 \leftarrow B_1 \cup \{R[j]\}$;
            if $\{B_1\}$ satisfies the second and the third conditions then
                $B_2 \leftarrow B_1$;
            endif
        endfor
        $P \leftarrow P \cup \{B_2\}$;
    endif
endfor
return $P$;
```

Fig. 3. PAR algorithm

The PAR algorithm is executed to find the partition. Two parents are chosen by the roulette-wheel method. The solutions are reproduced by a restricted uniform crossover – bits governed by the same partition subset must be passed together. The mutation is turned off. The diversity is maintained by the rank-space method [23, pp. 520–523]. The population size is determined empirically by the bisection method [17, pp. 64]. The bisection method performs binary search for the minimal population size. There might be 10% different between the population size used in the experiments and the minimal population size that ensures the optimal solution in all independent 10 runs.

5.2 A Visualization of the Simultaneity Matrix

To illustrate how the matrix changes over time, a matrix element is represented by a square. The square intensity is proportional to the value of matrix element (see Figure 4). In the early generation (A), the matrix elements are nearly identical because the initial population is generated at random. After that (B), the matrix elements become more distinct. The BBs in the lowest level are detected. The solution recombination is more speculative. Multiple bits are passed together, and therefore forming larger BBs. A few generations later (C), higher-level BBs are revealed. Finally (D), the population begins to lose diversity. The matrix elements are going to be identical. Note that the bits governed by the same BB do not need to be packed close together. It is done for the ease of presentation.

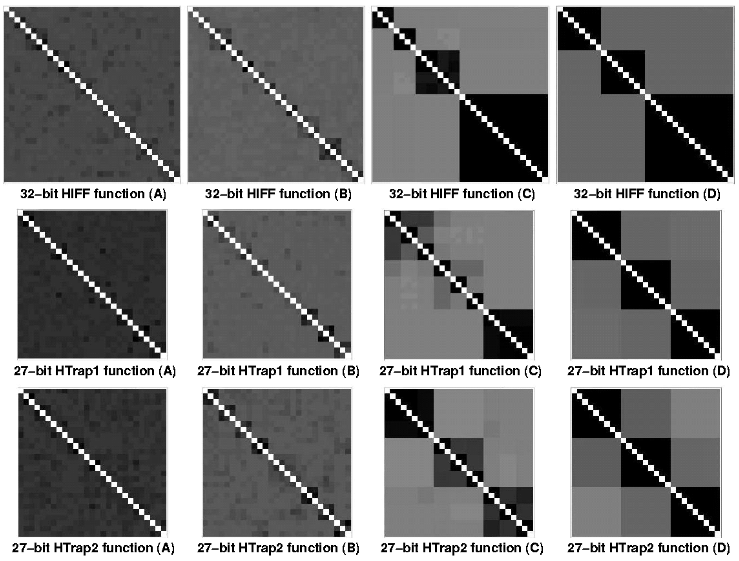


Fig. 4. Simultaneity matrices (HIFF, HTrap1, and HTrap2 functions)

5.3 A Comparison to the hBOA

Our algorithm is compared to the hBOA [17, pp. 164–165]. Figure 5 shows the number of function evaluations required to reach the optimum. The HTrap2 result is not shown because it is identical to that of the HTrap1. The linear regression in log scale indicates a polynomial relationship between the number of function evaluations and the problem size. The degree of polynomial can be approximated by the slope of linear regression. It can be seen that the hBOA and BISM can solve the HDFs in a polynomial time. The hBOA performs better than the BISM. However, the performance gap narrows as the problem becomes harder (HIFF, HTrap1, and HTrap2 functions respectively).

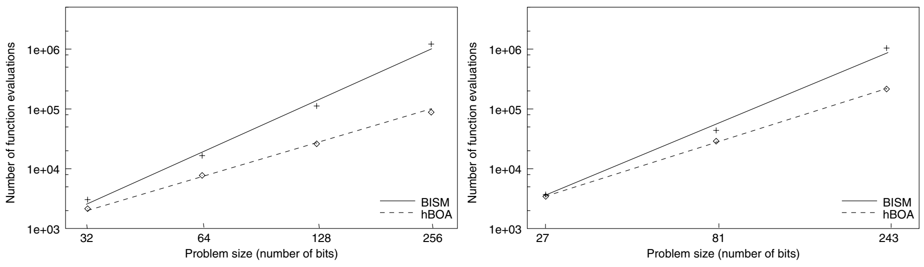


Fig. 5. Performance comparison: HIFF (left) and HTrap1 (right)

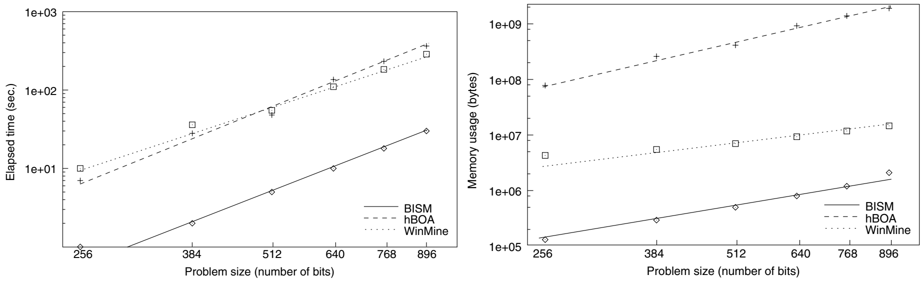


Fig. 6. Performance comparison: elapsed time (left) and memory usage (right)

We make another comparison in terms of elapsed time and memory usage. The elapsed time is an execution time of a call on `constructTheNetwork` subroutine [16]. The memory usage is the number of bytes dynamically allocated in the subroutine. The hardware platform is HP NetServer E800, 1GHz Pentium-III, 2GB RAM, and Windows XP. The memory usage in the hBOA is very large because of inefficient memory management in constructing Bayesian network. A fair implementation of the Bayesian network is the WinMine Toolkit [3]. The WinMine is a set of tools that allow you to build statistical models from data. It constructs Bayesian network with decision tree that is similar to that of the hBOA. The WinMine's elapsed time and memory usage are measured by an execution of `dnet.exe` – a part of the WinMine that constructs the network. All experiments are done with the same biased population that is composed of aligned chunks of zeroes and ones. The parameters of the hBOA and WinMine Toolkit are set at default. The population size is set at three times greater than the problem size.

The elapsed time and memory usage averaged from 10 independent runs are shown in Figure 6. The Bayesian network is a powerful tool that builds statistical models from data. However, constructing the network is time-consuming. This is because the network gathers all dependency between bit variables. In contrast, the matrix records only dependency between two bits that are likely to be disrupted in the uniform crossover. Therefore the matrix computation is much faster. The empirical results show that the hBOA outperforms the BISM in terms of function evaluations, but computing the matrix is 10 times faster and uses 10 times less memory than constructing Bayesian network.

6 Conclusions

The BB identification is indispensable to the scalability of GAs. We have presented a BB identification by simultaneity matrix. The matrix element m_{ij} is proportional to the probability that 2-bit BBs at positions i and j will be disrupted by the uniform crossover. The matrix does not detect all dependency between bit variables. We have shown that there might be dependency between bits at positions i and j that cannot be detected by the matrix. Such dependency is not necessary because the 2-bit BBs at positions i and j are very likely to survive in the next generation regardless of the solution recombination methods. Exploiting the matrix is simply passing the bits at positions i and j together if m_{ij} is significantly high. More formally, we search for a partition of bit positions. The bits governed by the same partition subset are passed together every time performing crossover. It can be shown that the BISM can solve the hierarchical problem in a polynomial relationship between the number of function evaluations and the problem size. More importantly, the matrix computation is simple, fast, and memory efficient. The partition may not fully take advantages of the matrix. The matrix could be exploited in another way rather than partitioning. Future work is to combine the strengths of Bayesian network and the simultaneity matrix.

References

1. Ackley, D. H. (1987). A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers, Boston, MA.
2. Aporntewan, C., and Chongstitvatana, P. (2003). Building-block identification by simultaneity matrix. In CantúPaz, E. et al., editors, Proceedings of the Genetic and Evolutionary Computation, page 1566–1567, Springer-Verlag, Heidelberg, Berlin.
3. Chickering, D. M. (2002). The WinMine Toolkit. Technical Report MSR-TR-2002-103, Microsoft Research, Redmond, WA.
4. Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley, Reading, MA.
5. Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. Complex Systems, Vol. 3, No. 5, page 493–530, Complex Systems Publications, Inc., Champaign, IL.
6. Goldberg, D. E. (2002). The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Boston, MA.
7. Harik, G. R. (1997). Learning linkage. In Belew, R. K., and Vose, M. D., editors, Foundation of Genetic Algorithms 4, page 247–262, Morgan Kaufmann, San Francisco, CA.
8. Harik, G. R. (1999). Linkage learning via probabilistic modeling in the ECGA. Technical Report 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Champaign, IL.
9. Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. Evolutionary Computation, Vol. 8, No. 4, page 373–391, MIT Press, Cambridge, MA.
10. Kargupta, H., and Buescher, K. (1995). The gene expression messy genetic algorithm for financial applications. In Proceedings of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, page 155–161, IEEE Press, Piscataway, NJ.
11. Kargupta, H. (1996). The gene expression messy genetic algorithm. In Proceedings of the IEEE International Conference on Evolutionary Computation, page 814–819, IEEE Press, Piscataway, NJ.
12. Kargupta, H., and Park, B. (2001). Gene expression and fast construction of distributed evolutionary representation. Evolutionary Computation, Vol. 9, No. 1, page 43–69, MIT Press, Cambridge, MA.

13. Munetomo, M., and Goldberg, D. E. (1999). Linkage identification by non-monotonicity detection for overlapping functions. Evolutionary Computation, Vol. 7, No. 4, page 377–398, MIT Press, Cambridge, MA.
14. Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W. et al., editors, Proceedings of Genetic and Evolutionary Computation Conference, Vol. 1, page 525–532, Morgan Kaufmann, San Francisco, CA.
15. Pelikan, M., Goldberg, D. E., and Lobo, F. (1999). A survey of optimization by building and using probabilistic models. Computational Optimization and Applications, Vol. 21, No. 1, page 5–20, Kluwer Academic Publishers.
16. Pelikan, M. (2000). A C++ implementation of the Bayesian optimization algorithm (BOA) with decision graph. Technical Report 2000025, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Champaign, IL.
17. Pelikan, M. (2002). Bayesian optimization algorithm: From single level to hierarchy. Doctoral dissertation, University of Illinois at Urbana-Champaign, Champaign, IL.
18. Pelikan, M., and Goldberg, D. E. (2003). Hierarchical BOA solves Ising Spin Glasses and MAXSAT. In Cant-úPaz, E. et al., editors, Proceedings of Genetic and Evolutionary Computation Conference, page 1271–1282, Springer-Verlag, Heidelberg, Berlin.
19. Sastry, K., and Xiao, G. (2001). Cluster optimization using extended compact genetic algorithm. Technical Report 2001016, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Champaign, IL.
20. Thierens, D. (1999). Scalability problems of simple genetic algorithms. Evolutionary Computation, Vol. 7, No. 4, page 331–352, MIT Press, Cambridge, MA.
21. Watson, R. A., and Pollack, J. B. (1999). Hierarchically consistent test problems for genetic algorithms. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, Proceedings of Congress on Evolutionary Computation, page 1406–1413, IEEE Press, Piscataway, NJ.
22. Whitley, D., Rana, S., Dzubera, J., and Mathias, K. E. (1996). Evaluating evolutionary algorithms. Artificial Intelligence, Vol. 85, No. 1–2, page 245–276, Elsevier.
23. Winston, P. H. (1992). Artificial Intelligence, third edition. Addison-Wesley, Reading, MA.