Solving Multi-Objective Problems with Building Blocks Identification

Jiradej Ponsawat[†], Wandao Punyaporn[†], Nachol Chaiyaratana[*], and Prabhas Chongstitvatana[†]
[†]Department of Computer Engineering, Chulalongkorn University, Bangkok, 10330 Thailand
Tel: +66-2-2186985, Fax: +66-2-2186955
[*]Department of Electrical Engineering, King Mongkut's Institute of Technology North Bangkok, Bangkok, 10800 Thailand
E-mail: jiradej.p@student.chula.ac.th, wandao.p@gmail.com, nchl@kmitnb.ac.th, and prabhas@chula.ac.th

***Abstract*— Multiple-objective problems are a challenge for evolutionary algorithms. The requirement to improve the quality of the solution and at the same time maintain good candidates which may have different and conflicting objectives is a difficult one. This work proposes to apply the concept of Building Blocks to improve evolutionary algorithms to tackle such problems. Building Block Identification algorithm is used to guide the crossover operator in order to maintain good Building Blocks and mix them effectively. The proposed method is evaluated by using Building Block Identification guided crossover in a well-known genetic algorithm to solve multiple-objective problems. The result shows that the proposed method is effective. Moreover, it obtains a good spread of solutions even when the building blocks are loosely encoded.**

I. INTRODUCTION

Real world problems usually contain multiple possibly conflict objectives. Solving problems gives rise to a set of solutions which can be regarded as Pareto-optimal. This set of solutions is a trade-off between many objectives. Solving multiple-objective problems is an important challenge in optimisation. Genetic Algorithm has been applied with good success to solve this class of problems.

In [1], Goldberg gives description that Building Blocks (BBs) are short, low order and highly fit schemata and that these BBs play important role in action of GA because they are sampled, recombined, and resampled to form strings of potentially high fitness. The class of problems called *GA-deceptive* are designed to mislead a Simple Genetic Algorithm or any hill-climber algorithm. For this class of problems, Building Block Identification [2] is shown to be a good solver. Building Blocks are important for the success of Genetic Algorithm, not only for single objective problems but also for multiple-objective problems.

For multiple-objective deceptive problems, applying Genetic Algorithm with single point crossover has limited success due to the disruptive effect of the crossover operator on the building blocks. Building Block Identification algorithm as its name implied partitions bit-positions into groups. These groups are regarded as building blocks (BBs). The knowledge of BBs can be used to prevent disruption of highly fit solutions from crossover operators. When performing crossover, group of bits in the same BB should not be divided.

Evolutionary algorithms are popular methods to solve multiple-objective problems. There are two approaches to Building Blocks finding: either it is explicit method [3] or implicit. Messy genetic algorithm (mGA) and Linkage Learning GA (LLGA) [4] are some examples of works that are explicit BBs finding. In these algorithms each bit is tagged with the position numbers so that they can be moved around without losing the meaning. The messy GA is later developed to fast messy genetic algorithm (FMGA) [5] and gene messy genetic algorithm (GEMGA) [6]. The latter is an early work that can find the optimal solution for m×k trap function. Bayesian optimization algorithm (BOA) using Bayesian network to model a population proposed by Pelikan [7] is another one of explicit BBs finding. In a later version of BOA called hierarchical BOA (hBOA) [8], the *l*-vertex network is represented by *l* decision trees/graphs in order to avoid the exponentially growth of the number of conditional probabilities in the network. As the result, the latter models are more compact and applicable for problems having higher order of variable interaction.

In the context of multiple-objective problems, the Multiple-objective Bayesian Optimization Algorithm (mBOA) [9] is identical to BOA, except that the selection procedure which is replaced by the non-dominated sorting and selection mechanism of NSGA-II [10]. The NSGA-II is well known in Multiple-objective Evolutionary Algorithm (MOEA) group and there have been much interest in improving its quality, for example, in [11] and [12]. NSGA-II is considered to be a leader of MOEA. NSGA-II is an implicit BB builder rather than an explicit one.

This work proposed the application of Building Block Identification algorithm to solve multiple-objective problems. We focus on finding good building blocks in the context of multiple-objective problems. The multiple-objective problems are solved with evolutionary algorithms. The Building Block Identification algorithm is used to guide the crossover operator which will mix building blocks. The aim of this work is to find out whether the claim that Building Blocks are important to multiple-objective problems can be substantiated [3].

The approach taken in this work is to modify a standard MOEA to use Building Block Identification in place of its original crossover operator. NSGA-II is chosen to be the

1-4244-0977-2/07/$25.00 © 2007 IEEE

evolutionary algorithm in our work. For the test problems, we employ the m×k-trap function [13], which is a problem with clearly defined BBs. There exist many works that use the trap function as the test problem, e.g. [14] [15] [16] and [17].

The structure of this paper is as follows. The next section gives the background on NSGA-II algorithm and the test problems, the multiple-objective deceptive function. Section III explained the proposed algorithm in details. The description of Building Block Identification algorithm and its application to modify the crossover operator are given. Section IV gives the detail of the experiments and discusses the results. Finally, the conclusion is offered in section V.

II. BACKGROUND

A. NSGA-II

Non-dominated sorting genetic algorithm II (NSGA-II) first published in [10] is one of the most popular genetic algorithm in many recent years. NSGA-II is a second generation of a genetic algorithm designed especially for multiple-objective optimisation (NSGA [18]). It has the ability to find multiple Pareto-optimal solutions in one single run. In NSGA-II, the population is sorted according to the level of non-domination. The diversity among non-dominated solutions is maintained using a measure of density of solutions in the neighbourhood. NSGA-II is able to find much better widespread solutions and better convergence near the true Pareto-optimal front in most problems.

B. Multi-Objective Deceptive Function

To test the proposed method, Trap functions are used. They are difficult problems for Genetic Algorithms. Moreover they are problems which building blocks are obviously defined. The deceptive trap functions are modified to be multiple-objective style in [13]. To make it harder, a modified version called Shuffle trap function is created. This function creates non-compact building blocks (bit positions are not contiguous) which renders a simple crossover operator ineffective. These problems are defined in this section.

$m \times k$-trap function

This problem has 2 objectives: m × k deceptive trap, and m × k deceptive inverse trap. String positions are first divided into disjoint subsets or partitions of k bits each.

The k-bit trap and inverse trap are defined as follows:

$$\text{trap}_k(u) = \begin{cases} k & ; \text{if } u = k, \\ (k-d)\left[1 - \frac{u}{k-1}\right] & ; \text{otherwise} \end{cases} \quad (1)$$

$$\text{invtrap}_k(u) = \begin{cases} k & ; \text{if } u = 0, \\ (k-d)\left[\frac{u-1}{k-1}\right] & ; \text{otherwise} \end{cases} \quad (2)$$

Where u is the number of 1s in the input string of k bits, and d is the signal difference. Here, we use $k = 5$, and $d = 1$. The m×k-trap conflicts with the inverse trap by its objective. A solution that sets the bits in its partition either to 0s or 1s is Pareto optimal and there are a total of 2^m solutions in the Pareto-optimal front.

$m \times k$-trap function is defined as follows.

$$F_{m\times k} : B \rightarrow R \,;\, \text{B} \in \text{B}_0 \ldots \text{B}_{m-1}, \text{B}_i \in \{0,1\}^k, \quad \text{R} \in [0,\, m\times k] \quad (3)$$

$$F_{m\times k}(B_0 \ldots B_{m-1}) = \sum_{i=0}^{m-1} F_k(B_i) \quad (4)$$

Where F_k is $trap_k$ or $invtrap_k$ function. The m and k are varied to produce many test functions. These functions are often referred to as additively decomposable functions (ADFs). In this experiment, this function is modified to be multi-objective.

Shuffle $m \times k$-trap function

The shuffle trap function is constructed by separating the bit position of the same building blocks in order to deceive the algorithm. For instance, normal 4×5-trap function have building blocks as shown.

11111 ***** ***** *****

In shuffle trap, the modulo method is used to construct one building block. The same building block is repeated in every m bit.

1*** 1*** 1*** 1*** 1***

The function in the experiment uses 2-objective 10×5-trap function.

III. ALGORITHM

Building Blocks can be identified by computing Chi-square Matrices then perform partitioning of bit positions using Partition Algorithm proposed in [19]. Each element of Chi-square matrices represents the degree of relation between each bit of selected population. Partition Algorithm groups bits which are highly related into BBs. This knowledge of BBs is used in the design of crossover operator. When performing crossover, all bits in the same partition will be moved together.

A. Chi-square Matrix

To identify highly-related-group of bits, it is noted that their quantities are inversely related to randomness. The Chi-square Matrix [19] is chosen for measuring randomness because computing the matrix is simple and fast.

Let M = (m_{ij}) be an $l \times l$ symmetric matrix of numbers. Let S be a population or a set of l-bit binary strings. The Chi-square matrix is defined as follows.

$$m_{ij} = \begin{cases} ChiSquare(i,j) & ; \text{if } i \neq j \\ 0 & ; \text{otherwise} \end{cases} \quad (5)$$

The *ChiSquare(i,j)* is defined as follows.

$$\sum_{xy} \frac{(Count_S^{xy}(i,j) - n/4)^2}{n/4} \; ; xy \in \{00, 01, 10, 11\} \qquad (6)$$

Where the observe frequency $Count_S^{xy}$ *(i,j)* denotes the number of solutions in which bit *i* is identical to *x* and bit *j* is identical to *y*. The expected frequencies of observing "00,""01," "10," "11" are *n*/4 where *n* is the number of solutions. The common structures (or building-blocks) appear more often than the expected frequency. Consequently, the Chi-square of bit variables that are in the same BB is high. The time complexity of computing the matrix is $O(l^2 n)$.

B. Partitioning (PAR) Algorithm

Partitioning (PAR) Algorithm will partition each input bit into suitable blocks. When performing crossover, bits in the same partition must not be separated. The PAR input is an $l \times l$ matrix and its outputs the partition:

$$P = \{B_0, \ldots, B_{|P|-1}\} \quad ; \bigcup_{i=0}^{|P|-1} B_i = \{0, \ldots, l-1\},$$
$$B_i \cap B_j = \varnothing \quad \text{for all } i \neq j. \qquad (7)$$

The B_i is called partition subset. There are several definitions of the desired partition. Algorithm PAR must have some preconditions.

1. P is a partition.
   The members of P are disjoint set.
   The union of all members of P is $\{0, \ldots, l-1\}$.
2. $P \neq \{\{0, \ldots, l-1\}\}$.
3. For all $B \in P$ such that $|B| > 1$,
   For all $i \in B$, the largest $|B| - 1$ matrix elements in row i are founded in columns of $B \setminus \{i\}$.
4. For all $B \in P$ such that $|B| > 1$,
   $H_{max} - H_{min} < \alpha(H_{max} - L_{min})$ where $0 \leq \alpha \leq 1$,
   $H_{max} = max(\{m_{ij} \mid (i,j) \in B \times B, i \neq j\})$,
   $H_{min} = min(\{m_{ij} \mid (i,j) \in B \times B, i \neq j\})$, and
   $L_{min} = min(\{m_{ij} \mid i \in B, j \in \{0, \ldots. l-1\} \setminus B\})$.
5. 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, second, third and fourth conditions, $B \subset B_x$.

All partition subsets can be expanded until they are unsatisfied one of the preconditions. The motivation for the expansion is to put *i* and *j* into the same partition subset if m_{ij} is high. The time complexity of computing PAR is $O(l^4)$. The partition algorithm is shown as follow.

M= (m_{ij}) denotes $l \times l$ Chi-Square matrix. $0 \leq i, j \leq l-1$.
T_i and $R_{i,j}$ denote arrays of numbers indexed by $0 \leq i, j \leq l-1$.
A and *B* are partition subsets. *P* denotes a partition.

Algorithm PAR(M, α)
$P \leftarrow \varnothing$;
for $i = 0$ **to** $l - 1$ **do**
  if $i \notin B$ for all $B \in P$ **then**
    $T \leftarrow$ {matrix elements in row i sorted in descending order};
    for j = 0 to l − 1 do
      $R_{i,j} = x$ where $m_{ix} = T_j$
    endfor
    $A \leftarrow \{i\}$;
    $B \leftarrow \{i\}$;
    for $j = 0$ **to** $l - 3$ **do**
      $A \leftarrow A \cup \{R_{i,j}\}$;
      if A satisfies the third and the fourth conditions **then**
        $B \leftarrow A$;
      endif
    endfor
    $P \leftarrow P \cup \{B\}$;
  endif
endfor

return P;

C. Crossover Method

The crossover operator can exploit the knowledge of BBs by choosing appropriate cut points. The cut point should not separate bits in the same BB (see Fig.1). To achieve this, a crossover mask is created for each partition. When parents exchange bits to create offspring, all bits in the same partition will be moved together.

The mask bit is generated for each partition. All bits in each partition can be either exchange (with other individual) or remained the same. Flip coin method is used to choose whether a partition will be moved or remain unchanged. For instance, if the partition number 1 is assigned to be exchanged (mask value 0), all bit-positions in that partition will be assigned 0 in the mask. After all partitions have been assigned, the partitions which are assigned to 0 will be swapped with their mates. Otherwise, they remain the same. See the following example:

If the Chi-square matrix be partitioned like this one,
{ {1,5,9,13,17}, {2,6,10,14,18}, {3,7,11,15,19}, {4,8,12,16,20} }.

Partition <1234 1234 1234 1234 1234>
Mask Bits <0011 0011 0011 0011 0011>

x	x	x	x	x	x	x	x	…	x	x	x	x

Parent1

y	y	y	y	y	y	y	y	…	y	y	y	y

Parent2

After crossover, the two parents produce two children.

y	y	x	x	y	y	x	x	…	y	y	x	x

Child1

x	x	y	y	x	x	y	y	…	x	x	y	y

Child2

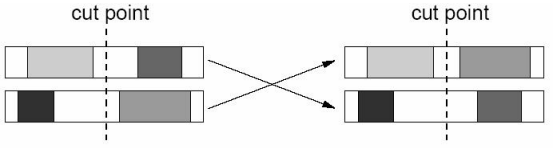

Fig. 1. Mixing and maintaining BBs

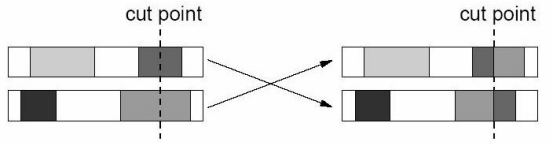

Fig. 2. Mixing and losing BBs

Fig. 1 and 2 illustrates the difference between the Building Block Identification crossover and a single point crossover employed in a Simple Genetic Algorithm. The crossover using BBI will choose the crossover point "between" BBs. The crossover in SGA is a single point crossover with random crossover point. The former will not break the partitions while the latter will random the cut point without considering the building block.

IV. Result and Discussion

The experiment is set to find out the effectiveness of the application of Building Block Identification to the multiple-objective trap problems. This class of functions usually leads the GA away from the global optimum. NSGA-II is used as the evolutionary algorithm. Its crossover operator is replaced by the crossover operator with Building Block Identification. The experiment is set to compare the result of the proposed method (named BB-NSGA-II) with the original NSGA-II. Any difference in the results should arise mainly from the use of Building Block Identification.

The function in the first experiment is 2-objective 10×5-trap function. To find solutions to the problems, the parameters are set as follows: Population size = 2000, Generation = 1000, Crossover rate = 0.9, Mutation rate is 0.1 and Threshold (α) of PAR is set to 0.95 where it is the appropriate value that generates high quality building blocks. The data in the experiments are the average from 30 independent runs. The evaluation criterion is the number of solutions found in the Pareto front. The number of function evaluations is the same for both methods.

Fig. 3 illustrates that at the first 320 generations, NSGA-II population appears in Pareto front more than BB-NSGA-II. This is because BB-NSGA-II must spend effort in identifying building blocks, hence the progress is slower. After generation 350, BB-NSGA-II population is found in Pareto front more than NSGA-II.

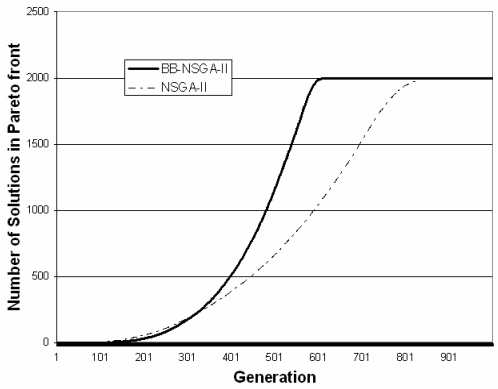

Fig. 3. The result for 2-objective 10×5-trap function.

The function in the second experiment is 2-objective shuffle 10×5 - trap function. The parameters setting are the same as in the former experiment. In Fig. 4, BB-NSGA-II population appears in Pareto front much greater than NSGA-II from the generation 200 until the end of run. The shuffle trap function is designed to create an impossible situation for an ordinary crossover. The crossover will always disrupt the solution. However, with Building Block Identification, the correct crossover will be obtained, hence produces a good progress. The performance of BB-NSGA-II does not differ much between the ordinary and shuffle version of problems, while the performance of NSGA-II drops dramatically in the shuffle problems.

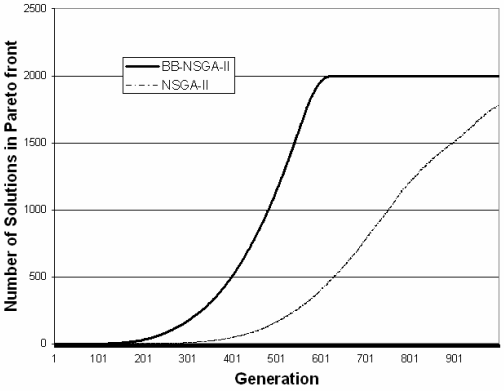

Fig. 4. The result for 2-objective shuffle 10×5-trap function

Pareto front of this problem have 1024 distinct solutions from all (2^{50} = 1125899906842624) possible solutions. When plot the Pareto-optimal set, there are 11 distinct points in objective space. Fig.5 shows number of candidate solutions of those points. Both left-most and right-most point has only one solution, whereas the middle zone has more than two-hundred candidate solutions. Fig. 6 illustrates spreading of solutions in Pareto front taken from the result of the experiments. Each row is a result from a run. All results are collected from 30 independent runs and they are displayed in 30 rows. There are 11 dots of white or black in each row. The black dot means the solution is found in Pareto front, while the white

dot means that solution is not found. This figure shows the "simplified view" of the spreading of solution in Pareto front. In Fig.7, the results are shown in more quantitative terms. The number of solutions in Pareto front is portrait by gray scale. The darker color means higher number of solutions.

Fig. 6 and 7 shows that in 2-objective 10×5-trap problems NSGA-II give better spread of solutions in Pareto front than BB-NSGA-II. In contrast, the spread of solutions given by NSGA-II is significantly reduced for 2-objective shuffle 10 × 5-trap problems. BB-NSGA-II is much better. In addition, Fig. 7 (c) shows that almost all solutions from NSGA-II are at the same point in Pareto front.

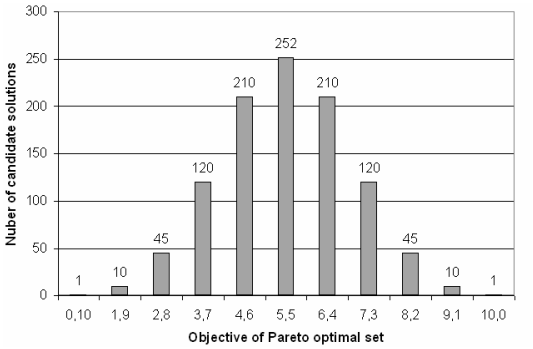

Fig. 5. Density of solution of each Pareto optimal

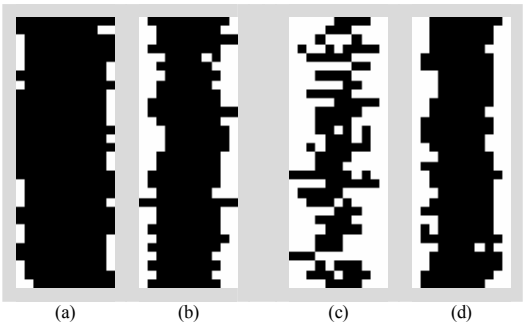

Fig. 6. Spread of solutions in Pareto front
(a) NSGA-II
(b) BB-NSGA-II
(c) NSGA-II (shuffle trap)
(d) BB-NSGA-II (shuffle trap)

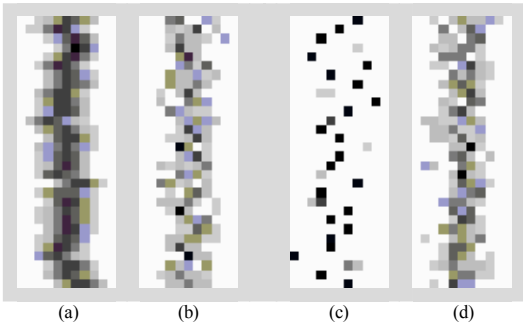

Fig. 7. Density of solutions in Pareto front
(a) NSGA-II
(b) BB-NSGA-II
(c) NSGA-II (shuffle trap)
(d) BB-NSGA-II (shuffle trap)

V. CONCLUSION

Building Block Identification method is effective in solving the Multi-Objective trap functions. From the experiment, the proposed method performs more effectively than NSGA-II, even though NSGA-II is designed for Multi-Objective problems. The experiment with the shuffle trap function demonstrates clearly that composing building blocks is highly effective.

Several interesting topics regarding the Building Block Identification require further exploration. We would apply our algorithm to other multi-objective problems. Some interesting problems such as multi-objective traveling salesperson problems [20] and multi-objective knapsack problems [21].

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989.

[2] C. Aporntewan, and P. Chongstitvatana, "Building-block identification by simultaneity matrix". Soft Computing. (2007) 11: 541-548.

[3] R. O. Day, and G. B. Lamont. An Effective Explicit Building Block MOEA, the MOMGA-IIa. *In 2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, vol 1, 17–24, Edinburgh, Scotland, September 2005.

[4] G. R. Harik, Learning Linkage, *Foundation of Genetic Algorithms 4*, Morgan Kaufmann, San Francisco, 1997, 247-262.

[5] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *In Proc of the 5th International Conference on Genetic Algorithm*, 56-64, 1993.

[6] H. Kargupta, The gene expression messy genetic algorithm. *In Proc of Congress on Evolutionary Computation*, 814-819, 1996

[7] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz The Bayesian optimization algorithm, *In Proc of Genetic Algorithm and Evolutionary Computation Conference,* 1999.

[8] M. Pelikan, and D. E. Goldgerg, Escaping hierarchical traps with competent genetic algorithms. *Technical Report 2001003*, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana Champaign.

[9] M. Laumanns and J. Ocenasek, "Bayesian Optimization Algorithms for Multi-objective Optimization", *In Proc of 7th Int. Conf. Parallel Problem Solving from Nature (PPSN VII)*, Granada, Spain, September 7-11, 2002.

[10] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation, 6*, 2 (April 2002), 182-197.

[11] F. G. Lobo, and C. F. Lima, Revisiting Evolutionary Algorithms with On-the-Fly Population Size Adjustment, *In Proc. of the Genetic and Evolutionary Computation (GECCO 2006)*, Seattle, Washington, USA, July 8–12, 2006, 1241-1248.

[12] K. D. Tran, Elitist non-dominated sorting GA-II (NSGA-II) as a parameter-less multi-objective genetic algorithm. *In Proc of the SIGCHI conference on Human factors*, IEEE,SounteastCon, Apritl 8-10, 2005, 359-367.

[13] K. Sastry, M. Pelikan, and D. E. Goldberg, (2005). Limits of scalability of multiobjective estimation of distribution algorithms. *In Proc of the Congress on Evolutionary Computation*, 3, 2217-2224.

[14] C. Aporntewan, and P. Chongstitvatana, A quantitative approach for validating the building block hypothesis, *IEEE Congress of Evolutionary Computation*, Edinburgh, September 2-5, 2005.

[15] C. Aporntewan, and P. Chongstitvatana, Simultaneity Matrix for Solving Hierarchically Decomposable Functions, *In Proc of the Genetic and Evolutionary Computation (GECCO 2004),* Lecture Notes in Computer Science, pp. 877-888, Seattle, WA, USA, June 26-30, 2004.

[16] M. Clergue, and P. Collard, GA-hard functions built by combination of Trap functions. *In Proc of the 2002 Congress on Evolutionary Computation*, May 12-17, 2002.

[17] S Nijssen, and T Back, An analysis of the behavior of simplified evolutionary algorithms on trap functions. *IEEE Transactions on Evolutionary Computation, 7(1),* February, 2003.

[18] N. Srinivas, and K. Deb, "Multiple objective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, pp. 221-248, 1994.

[19] C. Aporntewan, and P. Chongstitvatana, Chi-square matrix: An approach for building-block identification. *In Proc of 9th Asian Computing Science Conference*, December 8-10, 2004, 63–77.

[20] E. Zitzler, *Evolutionary Algorithms for Multiobjective Optimization: Methods Applications*. Ph.D. thesis, Shaker Verlag, Aachen, Germany, 1999.

[21] MCDM Numerical Instances Library, http://www.univ-valenciennes.fr/ROAD/MCDM/ListMOKP.html