A Selecto-recombinative Genetic Algorithm with Continuous Chromosome Reconfiguration

Jiří Kubalík, Petr Pošík, and Jan Herold

Department of Cybernetics, CTU Prague,
Technicka 2, 166 27 Prague 6, Czech Republic
{kubalik,posik}@labe.felk.cvut.cz, jan_herold@volny.cz

Abstract. A good performance of traditional genetic algorithm is determined by its ability to identify building blocks and grow them to larger ones. To attain this objective a properly arranged chromosome is needed to ensure that building blocks will survive the application of recombination operators. The proposed algorithm periodically rearranges the order of genes in the chromosome while the actual information about the inter-gene dependencies is calculated on-line through the run. Standard 2-point crossover, operating on the adapted chromosomal structure, is used to generate new solutions. Experimental results show that this algorithm is able to solve separable problems with strong intra building block dependencies among genes as well as the hierarchical problems.

1 Introduction

Standard selecto-recombinative genetic algorithms (GAs) work with fixed-length chromosomes with the gene order fixed during the whole evolution. A population of such chromosomes is evolved—they are recombined in order to generate the new ones. Through the evolutionary process, various high quality partial solutions called building blocks are generated that are needed to obtain the optimum solution. GA has to be able to identify essential building blocks and grow them to larger ones. Tightly linked building blocks have relatively high probability of surviving through successive generation, while building blocks with weak linkage are very likely to be disrupted by crossover operations. Thus, it is crucial that the genes belonging to the same building block are spatially grouped close to each other in order to maximize the efficiency of mixing the building blocks by recombination operators.

Many approaches have been proposed to deal with the linkage problem and to ensure appropriate building block mixing. They can be classified into the following categories:

- Linkage identification/learning techniques such as the messy GAs [1], linkage identification by non-monotonicity detection [6], linkage learning GA [2], dependency structure matrix driven GA [12], adaptive linkage model [5].
- Estimation of distribution algorithms also called probabilistic model building techniques include the bivariate marginal distribution algorithm [7], extended compact GA [3], Bayesian optimization algorithm [8].

A selecto-recombinative GA with continuous chromosome reconfiguration (CoCR), described in this paper, is based on an idea that given a chromosome structure that enables tight linkage of building blocks of the given problem, standard recombination operators can be used to efficiently grow the building blocks when recombining parental individuals. In order to attain this, CoCR updates pair-wise gene dependencies periodically during the evolutionary process and rearranges the chromosomal structure every generation according to the actual linkage information. The chromosome structure is generated by a heuristic procedure so that it reflects the most significant gene dependencies. Standard 2-point crossover, operating on the adapted chromosomal structure, is used to generate new solutions.

The rest of this paper is organized as follows. The next section describes the proposed algorithm. Section 3 and 4 describe the test problems and experimental setup used for our experiments. Then the results of experiments are presented and discussed. The work is summarized and possible future extensions are given in the last section.

2 CoCR Algorithm

An outline of CoCR algorithm is in Fig. 1. CoCR starts with a randomly generated order of genes in the chromosome. Then, the algorithm iterates in the main loop (1) generating subsequent populations of candidate solutions using a diversity preservation variant of selecto-recombinative GA while (2) continuously adapting the chromosome structure on the fly.

The pair-wise gene statistics are accumulated in *stats* every generation. All chromosomes of the processed population contribute to *stats*, no selection of high quality solutions is applied. The *stats* are accumulated for one epoch, consisting of *recalculation_step* number of generations, and then the gene dependencies are recalculated and stored in table *links_table*. *links_table* is a table of size $[n \times m]$, where for each gene $i \in \{0, n-1\}$ there is an ordered list of up to m genes that have the strongest epistasis with gene i. Each lists of the most epistatic genes is ordered in a descendant manner with respect to the strength of the epistasis. After the *links_table* has been updated the *stats* are erased.

The chromosome structure is reconfigured every generation using the actual gene dependencies in *links_table*. The gene order is calculated using the same gene dependencies information for each generation within one epoch. However, the actual linkage of genes in the chromosome might change slightly generation by generation since the heuristic procedure for the chromosome structure construction involves a stochastic component. This is an important aspect that makes the approach resistant against imperfectly identified epistasis among genes.

2.1 Identification of Pair-wise Gene Dependencies

CoCR uses the Pearson's chi-square test for discovering the pair-wise gene dependencies in the same way as it was used in the BMDA [7]. It estimates the dependencies from the collected set of N chromosomes. For each gene i we define the

```
init(population)
init(chromosome_structure)
generations = 0
erase(stats)
repeat
    population ← selecto-recombinative_GA(population)
    generations = generations + 1
    accumulate_pairwise_stats(population, stats)
    if(generations mod recalculation_step = 0)
        links_table ← recalculate_links_table(stats)
        erase(stats)
    chromosome_structure ← construct_chromosome(links_table)
until(CoCR termination condition is met)
```

Fig. 1. An outline of CoCR algorithm.

univariate marginal frequency $p_i(x_i)$ as the frequency of chromosomes that have ith gene set to x_i, where $x_i \in \{0, 1\}$. For any two positions $i \neq j \in \{0, \ldots, n-1\}$ and any possible values $x_i, x_j \in \{0, 1\}$ we define the bivariate marginal frequency $p_{i,j}(x_i, x_j)$ as the frequency of chromosomes that have genes i and j set to x_i and x_j, respectively. Then, for each pair of genes i and j, the Pearson's chi-square statistics can be defined by

$$X^2_{i,j} = \sum_{x_i, x_j} \frac{(Np_{i,j}(x_i, x_j) - Np_i(x_i)p_j(x_j))^2}{Np_i(x_i)p_j(x_j)},$$

where $Np_i(x_i)p_j(x_j)$ and $Np_{i,j}(x_i, x_j)$ is the expected and the observed frequency of the pair of values (x_i, x_j) on the positions i and j, respectively. If positions i and j are not independent for 95%, i.e. $X^2_{i,j} \geq 3.84$, then the value $X^2_{i,j}$ is recorded for both genes i and j and the strongest gene dependencies are retained in *links_table* for each gene. Note, that there may be some genes in *links_table* that have the list of the most related genes shorter than m if there were less than m dependencies identified for these genes.

2.2 Generation of the Chromosome Structure

The chromosome structure is constructed by means of a simple greedy algorithm (see Fig. 2) utilizing the information about the most significant epistases among genes (stored in *links_table*). The algorithm works in two steps - first the linkage groups are constructed, then the linkage groups are merged together to form the whole chromosome. It starts with a set of elementary linkage groups, where each gene represents one single group. Then it grows the linkage groups in the main loop (lines 3-11 in Fig. 2) using in turn the strongest links (i.e. the first column of *links_table*), then the second strongest links, etc., until all columns of *links_table*

```
init(linkage_groups)
$i = 0$
do
    unused ← {0,1,...,$n-1$}
    do
        gene = select_random(unused)
        linkage_groups ← link(gene,links_table[gene, i])
        remove gene from unused
    while(unused $\neq$ {})
    $i = i + 1$
while($i$ < $m$) and (card(linkage_groups)>1)
chromosome_structure ← merge(linkage_groups)
return(chromosome_structure)
```

Fig. 2. An outline of construct_chromosome procedure.

have been used or all the genes have been grouped in a single linkage group. In each i-th iteration of the main loop all links from the i-th column of *links_table* are used to grow the respective linkage groups. Note, that when processing ith column of *links_table* the links are picked from the list of unused links in a random order. Thus, each time the procedure is called different linkage groups can be generated, see Table 3.

A link between genes i and j is incorporated into the linkage groups so that the two linkage groups already assigned to i and j, respectively, are merged together. For example, let us assume that a link between genes 3 and 7 is to be added to the linkage groups, given the actual linkage groups assigned to gene 3 and 7 are LG_3 =(2–4–**7**–1) and LG_7 =(5–**3**–9), respectively. Merging LG_3 and LG_7 will yield a new linkage group $LG_3 = LG_7$ =(2–4–**7**–1–5–**3**–9).

After the linkage groups have been established they are merged together, again in a random order.

2.3 Genetic Algorithm with Allelic Diversity Preservation

It is crucial for proper functioning of the algorithm of identification of pair-wise dependencies to supply it with a diverse collection of chromosomes. Since the collection consists of the population contents collected over multiple generations the selecto-recombinative GA must be designed so that it can preserve the allelic diversity of the population at every generation.

In this work the genetic algorithm with limited convergence (GALCO) introduced by Kubalik et al. [4] is used to attain this goal. GALCO is based on the idea that the population is explicitly prevented from becoming homogenous by simply imposing limits on its convergence. This is done by specifying the maximum difference between the frequency of ones and zeros at any position of the chromosome calculated over the whole population. A steady-state evolutionary model and a special replacement operator are used to keep the desired distribution of ones and zeros during the whole run. An important point is that GALCO is a type of a simple selecto-recombinative GA that uses the standard recombination operators and no explicit mutation operator.

3 Test Problems

There are four test problems, each of them composed of different fundamental building blocks of variable size.

DF3. This is a representative of deceptive problems, i.e. problems that are intentionally designed to make a GA converge towards local deceptive optimum. The problem is composed of 25 copies of a 4-bit fully deceptive function DF3 taken from [11]. DF3 has a global optimum in the string 1111 with fitness 30 and a deceptive attractor 0000 with low fitness 10, which is surrounded, in the search space, by four strings of just one 1 with fitness values 28, 27, 26, and 25. The whole 100-bit long chromosome has the global optimum of value 750.

DF3-intrl. This problem is an extension of the previous problem such that the whole chromosome is split into 12-bit blocks, where each of the blocks contributes to the fitness by the value $\sum DF3(b_j, b_{j+1}, b_{(j+2)mod12}, b_{(j+3)mod12})$, where $j \in \{0, 2, 4, 6, 8, 10\}$. In the optimal case, when all bits of the block are set to 1, the contribution of the block is $6 \times 30 = 180$. Here, a problem composed of eight blocks was used with the global optimum of value 1440.

H-IFF. A hierarchical-if-and-only-if function proposed in [10] is the representative of hierarchically decomposable problems. The hierarchical block structure of the function is a balanced binary tree. Leaf nodes, corresponding to single genes, contribute to the fitness by 1. Each inner node is interpreted as 1 if and only if its children are both 1's, and as 0 iff they are both 0's - in such cases the inner node contributes to the fitness by a positive value $2^{height(x)}$, where $height(x)$ is the distance from the node x to its antecedent leaves. Otherwise the node is interpreted as null and its contribution is 0. The function has two global optima - one consists of all 1's and the other one has all 0's. We have used 128-bit problem with global optima of value 1024.

H-TRAP. The structure of this problem is similar to the structure of H-IFF with the difference that each inner node has three children and the contribution of each building block at every level is given as $3^{height(x)} \times f_{trap}(u)$, see [9]. $f_{trap}(u)$ is a trap function returning 1.0 if all its three children nodes interpret as 1, f_{min} if the three children nodes interpret as 0, $f_{min}/2$ if one out of the three children interpret as 1, and 0.0 if two children interpret as 1. We set $f_{min} = 1.01$ at lower levels, and $f_{min} = 0.9$ at the top most level. Leaf nodes (0-th level nodes) do not contribute any value. The problem with 81 bits with the global optimum of the value 324.0 was used.

4 Experimental Setup

For each problem, 20 runs have been executed, from which the following statistics were calculated:

- *MeanBest*. Mean *best-of-run* value calculated over the 20 independent runs.
- *StDev*. Standard deviation of the *best-of-run* values.
- *#Success*. A number of runs, in which the optimum solution was found.
- *WhenFound*. The average number of fitness evaluations needed to get the optimum solution.

Graphs showing (1) the evolution of the building block compactness and (2) the evolution of the best-so-far solution are generated to demonstrate the co-evolution of the chromosome structure and the quality of solution through the evolutionary process. Both the best-so-far fitness and the building block compactness show median values out of the 20 values in each generation.

The building block compactness is calculated as the average defining length of fundamental building blocks under the given chromosome structure with respect to the ringed representation implied by the 2-point crossover. For DF3 and DF3-intrl problem it shows an average defining length of 25 4-bit and 8 12-bit deceptive building blocks, respectively. In case of H-IFF problem, building blocks of 16 adjacent genes are considered. In case of H-TRAP problem, building blocks of 27 adjacent genes are considered. Four algorithms were compared:

- CoCR-GALCO. GALCO is used as the selecto-recombinative GA. The maximal diversity of the evolved population is forced by setting the convergence limit to 1. This means that the number of ones is within the interval $(PopSize/2-1, PopSize/2+1)$ at every position in the chromosome in any stage of the run.
- CoCR-SGA. CoCR using a standard genetic algorithm with bit flipping mutation applied to 1 bit per chromosome.
- GALCO-tight and GALCO-loose. GALCO operating on the chromosome of ideally organized genes (the most compact building blocks) and randomly chosen gene order, respectively. The gene order is static through the whole evolutionary process.

All algorithms used the same configuration: population size 500, 2-point crossover applied with a probability 1.0, tournament selection (n=4). Both variants of CoCR algorithm used *recalculation_step* 10 generations.

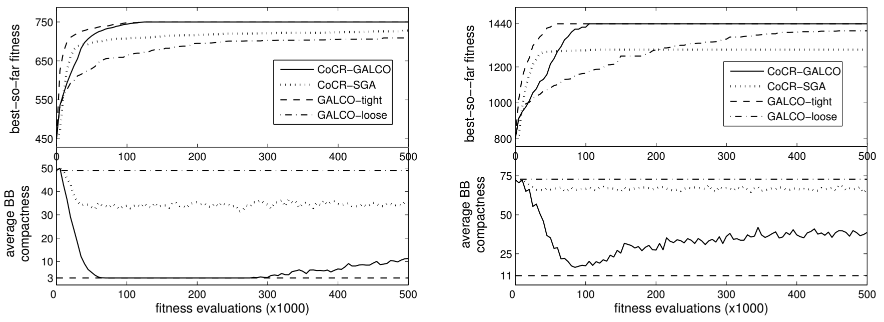
5 Experimental Results

Table 1 compares the observed performance characteristics of the algorithms. Results of CoCR algorithms are achieved with the number of columns of *links_table* set to 2. This means, that for each gene i a list of up to two genes with the strongest epistasis with gene i (i.e. *links_table* with $m = 2$) is considered in the process of building the chromosome structure. The results show that the performance of CoCR-GALCO lies between GALCO-loose and GALCO-tight and outperforms CoCR-SGA on all test problems. This is in agreement with our expectations. An interesting observation is that CoCR-GALCO perfectly solves DF3-intrl problem composed of high-order building blocks and scores even on the hierarchical problems.

Table 1. Comparisons on DF3, DF3-intrl, H-IFF, and H-TRAP.

		CoCR-GALCO	CoCR-SGA	GALCO-tight	GALCO-loose
DF3	*#Success*	20	0	20	0
	WhenFound	122658	-	115444	-
	MeanBest	750.0	728.8	750.0	708.9
	StDev	0.0	4.6	0	4.7
DF3-intrl	*#Success*	20	0	20	1
	WhenFound	108754	-	53881	498698
	MeanBest	1440.0	1305.0	1440.0	1392.4
	StDev	0.0	13.7	0.0	28.6
H-IFF	*#Success*	3	0	20	0
	WhenFound	115211	-	42564	-
	MeanBest	876.8	625.8	1024.0	626.8
	StDev	71.5	54.6	0.0	58.1
H-TRAP	*#Success*	10	0	20	0
	WhenFound	85152	-	26968	-
	MeanBest	293.6	197.6	324.0	199.6
	StDev	31.2	19.3	0.0	11.7

Fig. 3 also illustrates the inability of CoCR-SGA to evolve the proper chromosome structure with tight linkage. This is because SGA, using simple bit flipping mutation, is not able to maintain sufficiently high population diversity in comparison to GALCO. The less diverse the population is, i.e. the fewer unique chromosomes contribute to *stats*, the less information CoCR has for identification of the pair-wise gene dependencies used to derive the proper chromosome structure. This is also the reason why the BB compactness eventually increases



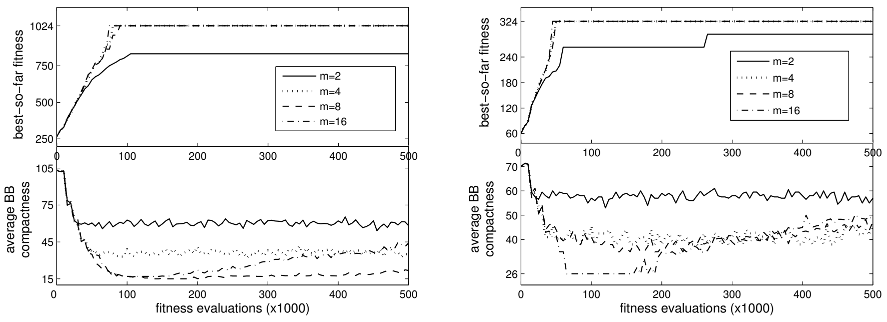
(a) DF3 (b) DF3-intrl

Fig. 3. Mean convergence characteristics observed for different algorithms

Table 2. Results achieved for different sizes of *links_table*.

	links_buffer	1	2	4	8
DF3	*#Success*	20	20	20	20
	WhenFound	141531	122658	122667	118522
	MeanBest	750.0	750.0	750.0	750.0
	StDev	0.0	0.0	0.0	0.0
DF3-intrl	*#Success*	9	20	20	20
	WhenFound	292447	108754	82501	85452
	MeanBest	1427.5	1440.0	1440.0	1440.0
	StDev	12.9	0.0	0.0	0.0
H-IFF	*#Success*	0	3	15	19
	WhenFound	-	115211	83714	61420
	MeanBest	763.2	876.8	992.0	1017.6
	StDev	52.9	71.5	56.9	28.6
H-TRAP	*#Success*	7	12	20	20
	WhenFound	105779	82303	42879	40234
	MeanBest	299.8	312.4	324.0	324.0
	StDev	21.3	18.7	0.0	0.0

for CoCR-GALCO, as observed in Fig. 3 and Fig. 4. This happens when the optimal solution has already been found. As GALCO still tries to keep the population maximally diverse, it gets saturated with half-to-half the copies of the optimum and copies of the string that is a binary complement to the optimum. From such a two-valued set of strings any valid information about gene dependencies can not be obtained and the generated chromosome structure becomes partially randomized.



(a) H-IFF (b) H-TRAP

Fig. 4. Mean convergence characteristics observed for different sizes of *links_table*

Table 3. Chromosome structure evolution. An excerpt of one run when solving 32-bit H-IFF problem. Population size was 200. The optimal solution was found after 3264 fitness evaluations. G stands for the generation, BBC is the average BB compactness of given chromosome structure.

G	BBC	chromosome structure
1	26.0	10 21 7 25 15 4 24 27 30 16 12 18 20 26 22 9 2 19 23 17 11 8 31 29 0 1 3 6 13 5 28 14
2	27.0	1 20 14 25 17 21 7 0 22 3 5 6 28 9 27 13 12 31 4 26 11 15 10 8 29 16 19 24 2 18 23 30
3	26.5	1 14 16 25 7 4 26 20 22 30 5 21 24 12 27 31 19 10 2 15 11 8 17 29 0 9 3 6 18 28 23 13
4	28.0	21 1 7 23 2 30 3 27 25 4 12 31 22 29 13 28 8 10 15 24 9 17 11 26 5 20 19 6 16 18 0 14
5	25.0	13 7 23 3 4 8 10 0 25 12 27 11 17 29 9 28 30 6 1 24 31 22 20 26 19 21 5 18 16 15 2 14
16	9.0	26 27 25 24 21 20 23 22 3 2 0 1 4 5 6 7 8 9 10 11 13 12 14 15 16 18 17 19 29 28 30 31
17	7.0	$\underline{4\ 5\ 6\ 7\ 3\ 2\ 0\ 1}$ $\overline{19\ 18\ 17\ 16\ 22\ 23\ 20\ 21\ 26\ 27\ 25\ 24\ 31\ 30\ 29\ 28}$ $\underline{11\ 10\ 8\ 9\ 12\ 13\ 14\ 15}$
18	9.0	3 2 0 1 28 29 30 31 7 6 5 4 11 10 8 9 13 12 14 15 19 17 16 18 21 20 22 23 26 27 25 24
19	7.0	$\underline{23\ 22\ 20\ 21\ 25\ 24\ 26\ 27\ 31\ 30\ 28\ 29}$ $\overline{10\ 11\ 9\ 8\ 15\ 14\ 12\ 13\ 4\ 5\ 6\ 7\ 1\ 0\ 2\ 3}$ $\underline{19\ 18\ 17\ 16}$
20	9.0	28 29 30 31 12 13 15 14 7 6 5 4 0 1 2 3 10 11 8 9 18 19 17 16 23 22 21 20 26 27 25 24

Results in Table 2 and graphs in Fig. 4 show how the size m of *links_table* influences the performance of CoCR-GALCO. The general observation is that as the size m increases the better results in shorter time are achieved so that even the hierarchical problems are almost perfectly solved with $m = 8$. In other words, the more of the linkage information the construction algorithm can use the better results can be expected.

Table 3 illustrates the evolution of the chromosome structure for the 32-bit H-IFF problem. It shows that despite the last 5 generations were generated using the same linkage information stored in *links_table* they differ each other. This is due to the stochastic nature of the algorithm for generating the chromosome structure. Intuitively, such a variability may make the algorithm more robust when solving problems with complicated epistatic structure where the genes can not be arranged in a linear chromosome so that tight linkage of all strongly dependent genes are satisfied. In such cases, different chromosomes that neglect different dependencies can be generated each time.

6 Conclusions

The proposed CoCR algorithm identifies the pair-wise gene dependencies that are used to generate the proper chromosome structure on which standard recombination operators work. The chromosome structure changes every generation and the actual information about the inter-gene dependencies is calculated online through the run. The algorithm for construction of the chromosome structure uses a list of the most related genes to each gene. Thus, the generated chromosome can capture a structure of the problem with higher-order fundamental building blocks that can be efficiently processed then. Moreover, the chromosome reconfiguration that takes place every generation makes the algorithm more resistant against partial errors in the gene order. This was documented on problems with 12-bit building blocks as well as on hierarchical problems.

The results of the experiments suggest that this approach could be used for solving problems, where the epistatic structure is too complicated so that it is hard to find just one optimal arrangement of genes in the chromosome. The experiments presented in this paper illustrated the performance of the proposed algorithm on problems with uniformly scaled subfunctions. Future research should investigate applicability of this approach to problems with exponentially scaled subfunctions as well.

Acknowledgments. This research has been supported by the research program Information Society No. 1ET 101 210 513 "Relational machine learning for analysis of biomedical data", sponsored by Czech Academy of Sciences.

References

1. Goldberg, D.E., Korb, B. and Deb, K.: Messy genetic algorithms: Motivation, analysis and first results. Complex Systems 3(5), pp. 493–530, 1989.
2. Harik, G. and Goldberg, D. E.: Learning linkage. In *Foundations of Genetic Algorithms IV*, pp. 270–85. San Mateo: Morgan Kaufmann, 1997.
3. Harik, G.: Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report N. 99010, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.
4. Kubalik, J., Rothkrantz, L.J.M., Lazansky, J.: Genetic Algorithm with Limited Convergence. In Grana, M., Duro, R., d'Anjou, A., and Wang, P.P., (Eds.), Information Processing with Evolutionary Algorithms: From Industrial Applications to Academic Speculations. Springer-Verlag, ISBN: 1-85233-866-0, pp. 216-235. 2005.
5. Kwon, Y. K., Hong, S.D., Moon, B. R.: A Genetic Hybrid For Critical Heat Flux Function Approximation. In W. B. Langdon et al. (Eds.), Proceedings of GECCO 2002, Morgan Kaufmann Publishers, pp. 1119-1125, 2002.
6. Munetomo, M. and Goldberg, D. E.: Linkage identification by non-monotonicity detection for overlapping functions, Evolutionary Computation, Vol. 7, No. 4, pp. 377-398, 1999.
7. Pelikan, M., Muehlenbein, H.: The Bivariate Marginal Distribution Algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, (Eds.), Advances in Soft Computing - Engineering Design and Manufacturing, pp. 521-535, Springer-Verlag. 1999.
8. Pelikan, M., Goldberg, D. E., Cantu-Paz, E.: Linkage learning, estimation distribution, and Bayesian networks. Evolutionary Computation, 8(3), 314-341, 2000.
9. Pelikan, M., Goldberg, D. E.: Escaping hierarchical traps with competent genetic algorithms. In L. Spector et al. (Eds.), Proceedings of the GECCO–2001, pp. 511–518. Morgan Kaufmann, 2001.
10. Watson, R. A., Hornby, G. S., and Pollack, J. B.: Modeling Building-Block Interdependency. In *proceedings of Fifth International Conference PPSN V*, pp. 97–106. Springer, 1998.
11. Whitley, D.: Fundamental Principles of Deception in Genetic Search. In: *Foundations of Genetic Algorithms*. G. Rawlins (ed.), 221–241, Morgan Kaufmann, 1991.
12. Yu, T.-L., Goldberg, D. E.: Dependency Structure Matrix Analysis: Offline Utility of the Dependency Structure Matrix Genetic Algorithm. In K. Deb et al. (Eds.), Proceedings of GECCO 2004, pp. 355-366, Springer Berlin/Heidelberg, 2004.