A Local Genetic Algorithm for Binary-Coded Problems*

Carlos García-Martínez[1], Manuel Lozano[2], and Daniel Molina[3]

[1] Dept. of Computing and Numerical Analysis, Univ. of Córdoba, 14071, Spain
in1gamac@uco.es
[2] Dept. of Computer Science and Artificial Intelligence, Univ. of Granada, 18071, Spain
lozano@decsai.ugr.es
[3] Dept. of Software Engineering, Univ. of Cádiz, 11002, Spain
daniel.molina@uca.es

Abstract. *Local Genetic Algorithms* are search procedures designed in order to provide an effective *local search*. Several Genetic Algorithm models have recently been presented with this aim. In this paper we present a new *Binary-coded Local Genetic Algorithm* based on a *Steady-State Genetic Algorithm* with a *crowding replacement method*. We have compared a *Multi-Start Local Search* based on the Binary-Coded Local Genetic Algorithm with other instances of this metaheuristic based on Local Search Procedures presented in the literature. The results show that, for a wide range of problems, our proposal consistently outperforms the other local search approaches.

1 Introduction

Local Search Procedures (LSPs) are optimisation methods that maintain a solution, known as *current solution*, and explore the search space by steps within its *neighbourhood*. The interest on LSPs comes from the fact that they may effectively and quickly explore the basin of attraction of optimal solutions, finding an optimum with a high degree of accuracy and within a small number of iterations. In fact, these methods are a key component of metaheuristics that are *state-of-the-art* of many optimisation problems, such as *Multi-start Local Search* ([3]), *Greedy Randomised Adaptive Search Procedures*, *Iterated Local Search*, *Variable Neighbourhood Search*, and *Memetic Algorithms* ([2]).

Genetic Algorithms (GAs) ([9,14]) have been seen as search procedures that can locate high performance regions of vast and complex search spaces, but they are not well suited for fine-tuning solutions ([17]). However, the components of the GAs may be *specifically designed* and their parameters *tuned*, in order to provide an *effective local search* as well. In fact, several GA models have recently been presented with this aim ([17,18]). These algorithms are called *Local Genetic Algorithms* (LGAs).

* This research was supported by the Spanish MEC project TIN2005-08386-C05-01.

T.P. Runarsson et al. (Eds.): PPSN IX, LNCS 4193, pp. 192–201, 2006.

LGAs present some advantages over classic LSPs. Most LSPs lack the ability to follow the proper path to the optimum on complex search landscapes. This difficulty becomes much more evident when the search space contains very narrow paths of arbitrary direction, also known as ridges. That is due to LSPs attempt successive steps along orthogonal directions that do not necessarily coincide with the direction of the ridge. However, it was observed that LGAs are capable of following ridges of arbitrary direction in the search space regardless of their direction, width, or even, discontinuities ([17]). Thus, the study of LGAs becomes a promising way to allow the design of more effective metaheuristics based on LSPs ([6,13,17,18,22]).

In this paper, we propose a *Binary-coded LGA* (BLGA) based on a *Steady-State Genetic Algorithm* (SSGA) with a *crowding replacement method*. It iteratively crosses a *leader solution* with individuals of the population belonging to the nearest niches. Then, the best solution between the leader one and the offspring becomes the new leader solution, and the other one is inserted in the population by means of the *Restricted Tournament Selection* ([11]). We have compared a Multi-start Local Search based on the new LGA with other instances of this metaheuristic based on LSPs proposed in the literature. The results show that, for a wide range of problems, this LGA consistently outperforms the other local search approaches.

The paper is organised as follows. In Section 2, we present the LGAs. In Section 3, we propose the BLGA. In Section 4, we compare the performance of the BLGA with LSPs presented in the literature. Finally, in Section 5, we provide some conclusions.

2 Local Genetic Algorithms

There are two primary factors in the search carried out by a GA ([23]):

- *Selection pressure*. In order to have an effective search there must be a search criterion (the fitness function) and a selection pressure that gives individuals with higher fitness a higher chance of being selected for reproduction, mutation, and survival. Without selection pressure, the search process becomes random and promising regions of the search space would not be favoured over non-promising regions.
- *Population diversity*. It is crucial to a GA's ability in order to continue the fruitful exploration of the search space.

Selection pressure and population diversity are inversely related: increasing selection pressure results in a faster loss of population diversity, while maintaining population diversity offsets the effect of increasing selection pressure.

Traditionally, GA practitioners have carefully designed GAs in order to obtain a balanced performance between selection pressure and population diversity. The main objective is to obtain their beneficial advantages simultaneously: to allow the most promising search space regions to be reached (*reliability*) and refined (*accuracy*).

Due to the flexibility of the GA architecture, it is possible to design GA models specifically aimed to provide *effective local search*. In this way, their unique objective is to obtain *accurate* solutions. These algorithms are named *Local Genetic Algorithms*. LGAs arise as an alternative choice to classical LSPs, in order to design metaheuristics based on LSPs. In fact, some LGAs were considered for this task ([6,13,17,18,22]).

3 Binary-Coded Local GA

In this section, we present a Binary-coded LGA (BLGA) that may be used to design metaheuristics based on LSPs. It is a *Steady-state GA* ([20,23]) that inserts one single new member into the population (P) in each iteration. It uses a *crowding replacement method* (*restricted tournament selection* (RTS) ([11])) in order to force a member of the current population to perish and to make room for the new offspring. It is important to know that RTS favours the formation of niches in P (groups of chromosomes with high quality located in different and scattered regions of the search space). In addition, the BLGA maintains an external chromosome, the *leader chromosome* (C^L), which is always selected as one of the parents for the crossover operation. The following sections indicate the main components of the BLGA.

3.1 General Scheme of the Binary-Coded LGA

Let's suppose that a particular metaheuristic applies the BLGA as LSP. When the metaheuristic calls the BLGA to refine a particular solution, the BLGA will consider this solution as C^L. Then, the following steps (Figure 1) are carried out during each iteration:

1. *Mate selection.* m chromosomes, Y^1, Y^2, ..., Y^m, are selected from the population applying the positive assortative mating m times (Section 3.2).
2. *Crossover.* C^L is crossed over with $Y^1, Y^2, ..., Y^m$ by applying the multiparent uniform crossover operator, generating an offspring Z (Section 3.3).
3. *Update of the leader solution and replacement.* If Z is better than C^L, then C^L is inserted into the population using the restricted tournament selection (Section 3.4) and Z becomes the new C^L. Otherwise, Z is inserted in the population using the same replacement scheme.

These steps are carried out until the stop condition described in Section 3.5 is achieved.

3.2 Positive Assortative Mating

Assortative mating is the natural occurrence of mating between individuals of similar phenotype more or less often than expected by chance. Mating between individuals with similar phenotype more often is called positive assortative mating and less often is called negative assortative mating. Fernandes et al. ([5])

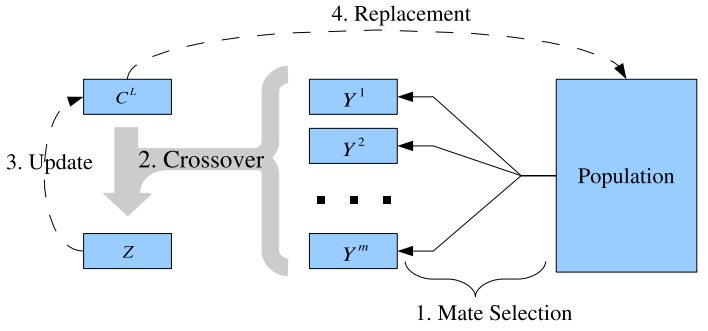

Fig. 1. Model of the BLGA

implement these ideas to design two mating selection mechanisms. A first parent is selected by the roulette wheel method and n_{ass} chromosomes are selected with the same method (in BLGA all the candidates are selected at random). Then, the similarity between each of these chromosomes and the first parent is computed (similarity between two binary-coded chromosomes is defined as the Hamming distance between them). If assortative mating is negative, then the one with less similarity is chosen. If it is positive, the genome more similar to the first parent is chosen to be the second parent. In the case of BLGA, the first parent is the leader chromosome and the method is repeated m times.

3.3 Multiparent Uniform Crossover Operator

The BLGA uses a *multiparent* version of the *Uniform crossover* (UX) ([20]) with a *short term memory* mechanism that avoids the generation of any offspring previously created. The pseudocode is shown in Figure 2, where $U(0,1)$ is a random number in $[0,1]$, $RI(1,m)$ is a random integer in $\{1,2,...,m\}$, and p_f is the probability of choosing genes from C^L (p_f is set to a high value in order to create offspring similar to C^L).

The short term memory remembers the genes of C^L that have been flipped at generating an offspring Z_k. Then, it avoids flipping those genes of C^L, in order to prevent the creation of Z_k once again. In order to do that, this mechanism maintains a mask, $M = (M_1, \ldots, M_n)$, where $M_i = 1$ indicates that the gene c_i^L can not be flipped in order to create an offspring. Initially, and when C^L is updated with a better solution, any gene can be flipped, so M_i is set to 0 for all $i \in \{1, \ldots, n\}$.

The multiparent UX with short term memory creates the offspring $Z = (z_1, \ldots, z_n)$ with:

- z_i is set to c_i^L for all $i = 1, \ldots, n$ with $M_i = 1$.
- If $M_i = 0$, then z_i is set to c_i^L with probability p_f. Otherwise, z_i is set to the ith gene of a randomly chosen parent Y^j. The mask is updated if z_i is different from c_i^L.

```
multiparent_UX(C^L, Y^1, ..., Y^m, m, p_f)
   For i = 1, ..., n
      If M_i = 1 OR U(0,1) < p_f //short term memory mechanism
         z_i ⟵ c_i^L;
      Else
         k ⟵ RI(1,m);
         z_i ⟵ Y_i^k;
         If z_i ≠ c_i^L
            M_i ⟵ 1; //update the mask
   If Z = C^L
      j ⟵ RI(1,n) such as M_j = 0;
      M_j ⟵ 1; //update the mask
      z_j ⟵ 1 − z_j;
   Return Z;
```

Fig. 2. Pseudocode of the multiparent UX with short term memory

– If Z is equal to C^L, then a gene chosen at random, i with $M_i = 0$, is flipped and the mask is updated.

The short term memory mechanism shares ideas with the one of the *Tabu Search* (TS) ([7]). Both of them help the sampling operator to efficiently explore the neighbourhood of the current solution C^L. Both of them avoid sampling previous solution more than once. The main difference is that the mechanism of the BLGA is entirely reset every time an offspring Z becomes better than C^L, whereas the elements in the one of the TS are eliminated, one by one, when their *tabu tenure* expires (usually, a fix number of algorithm iterations).

3.4 Restricted Tournament Selection

BLGA considers the *Restricted Tournament Selection* (RTS) ([11]) as *crowding method*. Its main idea is to replace the closest chromosome R to the one being inserted in the population, I, from a set of n_T randomly selected ones, if I is better than R.

The application of RTS together with the use of high population size may favour the creation of groups of chromosomes with high quality in P, which become located in different and scattered regions of the search space (*niches*).

3.5 Stop Condition

It is important to notice that, when every bit of the mask of the short term memory is set to 1 (Section 3.3), then, C^L will not be further improved, because the crossover operator will create new solutions exactly equal to C^L. Thus, this condition will be used as stop condition for the BLGA.

4 Experiments: Comparison with Other LSPs

The aim of this section is to compare the BLGA with other LSPs for binary-coded problems presented in the literature:

- the *First LSP* ([2]) that changes a random component of the current solution, which improves its fitness value,
- the *Best LSP* ([2]), which changes the bit that makes the best improvement, and,
- the *RandK LSP* ([16,19]) that examines a *k-variable* neighbourhood (it looks for solutions changing k components).

We have implemented four instances of the simplest LSP based metaheuristic, the Multi-start Local Search ([3]), each one with a different LSP. Multi-start Local Search iteratively creates a random solution and apply a LSP on it, until a stop condition is reached. At last, Multi-start Local Search returns the best solution obtained so far.

The four Multi-start Local Search instances will be called as follows:

- MS-First-LS: Multi-start with the First LSP.
- MS-Best-LS: Multi-start with the Best LSP.
- MS-RandK-LS: Multi-start with the RandK LSP.
- MS-BLGA: Multi-start with the BLGA.

We have chosen the Multistart Local Search metaheuristic in order to avoid possible synergies between the metaheuristic and the LSP. In this way, comparisons among the LSPs are fairer. All the algorithms were executed 50 times, each one performing 100,000 evaluations.

The BLGA uses 500 individuals as the population size, $p_f = 0.95$ and $m = 10$ mates for the crossover operator, $n_{ass} = 5$ for the Positive Assortative Mating, and $n_T = 15$ for the Restricted Tournament Selection. The population of the BLGA does not undergoes initialisation after the iterations of the Multistart Local Search, i.e. the initial population of the BLGA at the jth iteration of the MS-BLGA is the last population of the $(j-1)$th iteration. On the other hand, the leader chromosome is randomly generated at the beginning of the iterations of this metaheuristic.

4.1 Test Suite

Table 1 shows the test function used, their dimension, optimisation criterion (to maximise/minimise), optimum value and reference. Some comments are needed:

- Trap(4) consists on applying Trap(1) to a chromosome with 4 groups of 36 genes. Each group is evaluated with Trap(1), and the overall fitness of the chromosomes is the sum of the fitness of each group.

Table 1. Used test problems

Name	Dim	Criterion	f^*	Ref
Onemax(400)	400	min	0	
Deceptive(13)	39	min	0	[8]
Deceptive(134)	402	min	0	[8]
Trap(1)	36	max	220	[21]
Trap(4)	144	max	880	[21]
Maxcut(G11)	800	max	Not known	[15]
Maxcut(G12)	800	max	Not known	[15]
Maxcut(G17)	800	max	Not known	[15]
Maxcut(G18)	800	max	Not known	[15]
Maxcut(G43)	1000	max	Not known	[15]
M-Sat(100,1200,3)	100	max	1[1]	[4]
M-Sat(100,2400,3)	100	max	1[1]	[4]
NkLand(48,4)	48	max	1[1]	[4]
NkLand(48,12)	48	max	1[1]	[4]
BQP('gka')	50	max	3414[2]	[1,10]
BQP(50)	50	max	2098[2]	[1,10]
BQP(100)	100	max	7970[2]	[1,10]
BQP(250)	250	max	45607[2]	[1,10]
BQP(500)	500	max	116586[2]	[1,10]

- We have used 5 instances of the Max-cut problem (G11, G12, G17, G18, G43) from [12].
- We have used two set of instances of the Max-Sat problem with 100 variables (n), 3 variables by clause (l), and 1200 and 2400 clauses (m) respectively ([4]). They are denoted as M-Sat(n, m, l).
- We have used two set of instances of the NK-Landscape problem: one with $N = 48$ and $K = 4$, and another with $N = 48$ and $K = 12$ ([4]). They are denoted as NKLand(N, K).
- We have used 5 instances of the *Binary Quadratic Problem* (BQP) with different dimensions (n). They have been taken from the OR-Library. They are the first instances of the files 'bqpgka', 'bqp50', 'bqp100', 'bqp250', 'bqp500'. They are called BQP('gka'), BQP(50), BQP(100), BQP(250), and BQP(500), respectively.

4.2 Results

The results for all the algorithms are included in Table 2. It shows the average and the standard deviation of the best fitness function found over 50 executions. We have added, in parenthesis, the times the MS-BLGA is slower than the average of the other algorithms (the time consumed by the MS-BLGA divided by the average of the time consumed by the remainder, which were extremely similar). In addition, a two-sided *t-test* at 0.05 level of significance was applied in order to ascertain if the differences in the performance of MS-BLGA are significant when compared against the ones for the other algorithms. We denote the direction of any significant differences as follows:

[1] 1 is the maximum possible fitness value, however it may not exist any optimal solution with that fitness value, depending on the current problem instance.

[2] Best known values presented in [1].

Table 2. Comparison of the MS-BLGA with other Multistart LSP instances

		MS-First-LS	MS-Best-LS	MS-RandK-LS	MS-BLGA	+ ∼ -
	average	0	0	0	0	
Onemax(400)	sd	0 ∼	0 ∼	0 ∼	0 (753.12)	0 3 0
	average	8.68	3.36	14.32	8.68	
Deceptive(13)	sd	1.11 ∼	1.24 −	0.94 +	1.43 (1162.51)	1 1 1
	average	177.6	128.4	201.6	185.84	
Deceptive(134)	sd	5.03 −	10.5 −	7.51 +	9.56 (742.11)	1 0 2
	average	213.12	219.1	201.86	218.38	
Trap(1)	sd	2.54 +	1.94 ∼	2.41 +	2.39 (873.49)	2 1 0
	average	790.08	828.92	781.78	869.3	
Trap(4)	sd	7.17 +	8.09 +	7.88 +	6.97 (562.19)	3 0 0
	average	437.36	349.6	441	506.64	
Maxcut(G11)	sd	7.37 +	17.11 +	10.78 +	6.92 (52.47)	3 0 0
	average	425.6	335.16	431.32	497.36	
Maxcut(G12)	sd	7.23 +	15.65 +	12.17 +	6.97 (52.44)	3 0 0
	average	2920.82	2824.66	2946.58	2975.7	
Maxcut(G17)	sd	5.97 +	15.59 +	11.06 +	8.15 (51.16)	3 0 0
	average	849.86	628.32	873.82	898.08	
Maxcut(G18)	sd	11.30 +	22.15 +	18.68 +	15.98 (51.37)	3 0 0
	average	6427.44	5735.84	6463.1	6463.18	
Maxcut(G43)	sd	16.27 +	40.74 +	26.20 ∼	24.86 (49.3)	2 1 0
	average	0.9551	0.9526	0.9563	0.9566	
M-Sat(100,1200,3)	sd	3.7e-3 +	3.9e-3 +	3.3e-3 ∼	3.2e-3 (21.59)	2 1 0
	average	0.9332	0.9314	0.9335	0.9338	
M-Sat(100,2400,3)	sd	2.0e-3 ∼	2.5e-3 +	2.2e-3 ∼	1.9e-3 (11.25)	1 2 0
	average	0.7660	0.7647	0.7694	0.7750	
NkLand(48,4)	sd	1.4e-2 +	1.3e-2 +	1.4e-2 +	1.4e-2 (13.48)	3 0 0
	average	0.7456	0.7442	0.7493	0.7468	
NkLand(48,12)	sd	8.3e-3 ∼	7.7e-3 ∼	1.0e-2 ∼	9.5e-3 (9.39)	0 3 0
	average	3414	3414	3414	3414	
BQP('gka')	sd	0 ∼	0 ∼	0 ∼	0 (143.8)	0 3 0
	average	2098	2094.08	2096.72	2098	
BQP(50)	sd	0 ∼	15.68 ∼	9.05 ∼	0 (146.11)	0 3 0
	average	7890.56	7831.7	7881.52	7927.56	
BQP(100)	sd	33.79 +	57.75 +	38.01 +	43.15 (96,4)	3 0 0
	average	45557.16	45171.38	45504.22	45510.96	
BQP(250)	sd	33.68 ∼	295.46 +	99.28 +	128.92 (62.92)	2 1 0
	average	115176.88	108588.26	115335.34	115256.3	
BQP(500)	sd	494.89 ∼	2210.02 +	527.97 ∼	814.44 (50.14)	1 2 0
+/ ∼ /−		10 / 8 / 1	12 / 5 / 2	11 / 8 / 0		

- A plus sign (+): the average of MS-BLGA is better than the one of the corresponding algorithm.
- A minus sign (−): the algorithm improves the average of MS-BLGA.
- An approximate sign (∼): non significant differences.

We have added the last three columns and the last three rows that count the number of improvements, non-differences and reductions according to the t-test by functions and by algorithms, respectively.

The last three rows indicate that the BLGA arises as a promising algorithm to deal with binary-coded optimisation problems because it achieves many improvements and very few reductions versus the other approaches.

On the other hand, two remarks are worth being mentioned from the last three columns:

- MS-BLGA is one of the best algorithms for almost the 90% of the test functions. Concretely, MS-BLGA achieves better or equivalent results than

the ones of the other algorithms for all the functions, except on the two Deceptive ones.
- MS-BLGA returns the best results for 4 from up to 5 Max-cut problems.

It can be seen that these good results do not come for free. MS-BLGA invest runtime in order to obtain better results than the ones obtained by the other LSPs, performing the same number of fitness evaluations. However, it is interesting to notice that the differences become smaller when the dimension of the problem increases. The design of less time consuming LGAs, including parallel GAs, arises as an important idea from this study.

To sum up, we may conclude that the BLGA, working within the Multistart Local Search metaheuristic, is very competitive with classic LSPs, because it obtains better or equivalent results for almost all the test problems considered in this study.

5 Conclusions

In this paper, we have presented the BLGA, a LGA instance that incorporates specific mate selection mechanism, crossover operator, and replacement strategy to direct the local search towards promising search regions represented in the proper BLGA population.

An experimental study, including 19 binary coded test problems, has shown that when we incorporate the BLGA into a Multistart Local Search metaheuristic, this metaheuristic may improve their results with regards to the use of other LSP instances that are frequently used to implement it.

Several ideas for future developments arise from this study:

- Analyse the behaviour of the BLGA when it is used by different metaheuristics based on LSPs ([3,2]).
- Extend our investigation to different test-suites (other coding schemes) and real-world problems.
- Study adaptive mechanisms that control the parameters of the algorithm according to the current state of the search process.

References

1. J.E. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem. Technical Report, Management School, Imperial College, UK, 1998.
2. C. Blum, A. Roli. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys (CSUR) 35:2, 2003, pp. 268-308.
3. K.D. Boese and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. Operations Research Letters 16, 1994, pp. 101-113.
4. K. De Jong, M.A. Potter, W.M. Spears. Using problem generators to explore the effects of epistasis. Proc. of the Seventh International Conference on Genetic Algorithms, 1997, pp. 338-345.

5. C. Fernandes, A. Rosa. A study on non-random mating and varying population size in genetic algorithms using a royal road function. Proc. of the 2001 Congress on Evolutionary Computation, IEEE Press, Piscataway, New Jersey, 2001, pp. 60-66.
6. C. García-Martínez, M. Lozano, F. Herrera, D. Molina, A.M. Sánchez. Global and local real-coded genetic algorithms based on parent-centric crossover operators. European Journal of Operational Research, 2006. In press.
7. F. Glover, M. Laguna. Tabu search. Operational Research Society Journal 50:1, 1999, pp. 106-107.
8. D.E. Goldberg, B. Korb, K. Deb. Messy genetic algorithms: motivation, analysis, and first results. Complex Systems 3, 1989, pp. 493-530.
9. D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA, 1989.
10. V.P. Gulati, S.K. Gupta, A.K. Mittal. Unconstrained quadratic bivalent programming problem. European Journal of Operational Research 15, 1984, pp. 121-125.
11. G. Harik. Finding multimodal solutions using restricted tournament selection. Proc. of the 6th International Conference on Genetic Algorithms, L.J. Eshelman, editor, Morgan Kaufmann, San Mateo, California, 1995, pp. 24-31.
12. C. Helmberg, F. Rendl. A spectral bundle method for semidefinite programming. Siam Journal of Optimization 10:3, 2000, pp. 673-696.
13. F. Herrera, M. Lozano. Gradual distributed real-coded genetic algorithms. IEEE Transactions on Evolutionary Computation 4:1, 2000, pp. 43-63.
14. J.H. Holland. Adaptation in natural and artificial systems. The University of Michigan Press (The MIT Press, London, 1992).
15. R.M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85-103.
16. K. Katayama, H. Narihisa. A variant *k-opt* local search heuristic for binary quadratic programming. Trans. IEICE (A) J84-A:3, 2001, pp. 430-435.
17. S.A. Kazarlis, S.E. Papadakis, J.B. Theocharis, V. Petridis. Microgenetic algorithms as generalized hill-climbing operators for GA optimization. IEEE Transactions on Evolutionary Computation 5:3, 2001, pp. 204-217.
18. M. Lozano, F. Herrera, N. Krasnogor, D. Molina. Real-coded memetic algorithms with crossover hill-climbing. Evolutionary Computation Journal 12:3, 2004, pp. 273-302.
19. P. Merz, K. Katayama. Memetic algorithms for the unconstrained binary quadratic programming problem. Bio Systems 79:1-3, 2004, pp. 99-118.
20. G. Sywerda. Uniform crossover in genetic algorithms. Proc. of the third international conference on Genetic algorithms, 1989, pp. 2-9.
21. D. Thierens. Population-based iterated local search: restricting neighborhood search by crossover. Proc. of the Genetic and Evolutionary Computation Conference, LNCS 3103, 2004, pp. 234-245.
22. S. Tsutsui, A. Ghosh, D. Corne, Y. Fujimoto. A real coded genetic algorithm with an explorer and an exploiter population. Proc. of the Seventh International Conference on Genetic Algorithms, T. Bäck, editor, Morgan Kaufmann Publishers, San Francisco, 1997, pp. 238-245.
23. D. Whitley. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. Proc. of the Third International Conference on Genetic Algorithms, J. David Schaffer, editor, Morgan Kaufmann, San Mateo, 1989, pp. 116-121.