Evolving Crossover Operators for Function Optimization

Laura Dioşan and Mihai Oltean

Department of Computer Science,
Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Cluj-Napoca, Romania
{lauras, moltean}@cs.ubbcluj.ro

Abstract. A new model for evolving crossover operators for evolutionary function optimization is proposed in this paper. The model is a hybrid technique that combines a Genetic Programming (GP) algorithm and a Genetic Algorithm (GA). Each GP chromosome is a tree encoding a crossover operator used for function optimization. The evolved crossover is embedded into a standard Genetic Algorithm which is used for solving a particular problem. Several crossover operators for function optimization are evolved using the considered model. The evolved crossover operators are compared to the human-designed convex crossover. Numerical experiments show that the evolved crossover operators perform similarly or sometimes even better than standard approaches for several well-known benchmarking problems.

1 Introduction

Evolutionary algorithms are relatively robust over many different types of search spaces. This is why they are often chosen for use where there is little domain knowledge.

However, for particular problem domains, their performance can often be improved by tuning their parameters (such as type of operators, probabilities of applying the genetic operators, population size etc). One possible way to obtain good parameters is to let them to be adjusted along with the population of solutions. Another possibility is to evolve a population of parameters (or operators) which are applied for solving a particular problem. This is usually referred in the literature as Meta EA (or Meta GP) and it has been successfully applied for evolving complex structures (such as computer programs) [4], [8], [10], [12], [13].

Usually the genetic operators are fixed by the programmer and are not modified during the search process. Moreover, the same particular operators are used for a wide range of problems. This could lead to non-optimal behavior of the considered algorithms for some particular problems.

Our purpose is to find (by using evolutionary techniques) genetic operators which are suitable for solving particular problems. Roughly speaking, we will let the problem find by itself the genetic operators that correspond to its structure.

P. Collet et al. (Eds.): EuroGP 2006, LNCS 3905, pp. 97–108, 2006.

A new model for evolving crossover operators is proposed in this paper[1]. The model is a hybrid technique that combines Genetic Programming (GP) [6] and Genetic Algorithms (GAs) [5] within a two-level model. Each GP chromosome encodes a crossover operator which contains standard symbols (mathematical operators, constants and some variables). The evolved crossover is embedded into a standard Genetic Algorithm which is used for solving a particular problem (function optimization in our case).

The evolved crossover operators are compared to the human-designed convex crossover. For numerical experiments we have used ten artificially constructed functions and one real-world problem. Results show that the evolved crossover operators perform similarly or sometimes even better than standard approaches for several well-known benchmarking problems.

This research was motivated by the need of answering several important questions concerning genetic operators. The most important question is: *Can genetic operators be automatically synthesized by using only the information about the problem being solved?* And, if yes, which are the symbols that have to be used within a genetic operator (for a given problem)? We better let the evolution find the answer for us.

The paper is structured as follows: section 2 discusses work related to the evolution of evolutionary structures (such as genetic operators or evolutionary algorithms). The proposed model is described in section 3. Several numerical experiments are performed in section 4. Conclusions and further work directions are discussed in section 5.

2 Related Work

Several attempts for evolving variation operators for different techniques were made in the past.

Teller [13] describes a procedure for automatic design and the use of new genetic operators for GP. These SMART operators are co-evolved with the main population of programs and they learn to recombine the new population better than random genetic recombination. The SMART operators are programs that learn to do a graph crossover better than standard GP crossover.

Meta-Genetic Programming (MGP) [4] encodes the genetic operators as trees. These operators "act" on other tree structures to produce the next generation of individuals. In his paper on Meta-Genetic Programming [4], Edmonds used two populations: a standard GP population and a co-evolved population of operators that act on the main population. This technique introduces extra computational cost, which must be weighed against any advantage gained. Also the technique turns out to be very sensitive to biases in the syntax, from which the operators are generated, therefore it is less robust.

Peter Angeline [1] investigated the possibility of a "self-adaptive" crossover operator. In this work, the basic operator action is fixed (as a crossover) but

[1] The source code for evolving crossover operators and all the evolved operators will be available on www.eea.cs.ubbcluj.ro

probabilistic guidance is used to help the operator to choose the crossover nodes so that the operation is more productive.

Note that all these approaches are focused on evolving a crossover operator for GP technique. Their purpose was to obtain a better GP crossover. Our approach is quite different: we use a standard GP technique (with standard GP crossover) for evolving a crossover operator for evolutionary function optimization. We have trained our GP algorithm to find the expression of a crossover operator using one test function, and then we test this operator for other 10 functions.

3 Proposed Model

3.1 Representation

Consider the standard convex crossover operator [2], [5], [9]:

$$\text{Offspring} = x * \alpha + (1 - \alpha) * y.$$

The right-hand expression may be easily represented as a GP individual depicted in Figure 1.

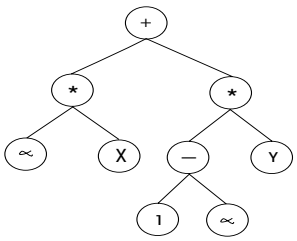

Fig. 1. Convex crossover. Two parents x and y are recombined in order to obtain an offspring. α is a real value randomly generated between 0 and 1. If the function to be optimized has multiple dimensions, the convex crossover operator will be applied for each of them.

Our purpose is to evolve a crossover operator (for function optimization) based on the information taken from a function to be optimized. The evolved crossover will be represented as a GP tree. The set of GP function symbols will consist in mathematical operators that can appear into a crossover operator:

$$F = \{+, -, *, sin, cos, exp\}$$

Our aim is to design a crossover operator which is able to optimize functions defined over any real domain. Not all genetic operators can do that. For instance, a genetic operator defined as $sin(x) + sin(y)$ will considerably reduce the search space to the interval [-2, 2]. If the optimal solution is not in that interval, our algorithm, which uses only that variation operator, will not be able to find it.

An idea is to have a crossover operator whose inputs (the parents and any other values placed in the leaves of the tree) are real values between 0 and 1. The output of that chromosome should also be a real number between 0 and 1. When we apply this operator for a particular problem, first we need to scale the parents to the [0, 1] interval and then we need to scale the output of the crossover to the definition domain of the function to be optimized. For instance, if the domain of the function to be optimized is [-5, 5], we need to scale down the parents to the interval [0, 1] and then we apply crossover and then we need to scale up the [0, 1] result to the interval [-5, 5].

Because we are dealing with real-valued definition domains (e.g. [0,1]) we have to find a way of protecting against overflowing these limits. For instance if we employ a standard addition of two numbers greater than 0.5 we would get a result greater than 1 (domain upper bound). Thus, each operator has been redefined in order to output result in the standard interval [0,1]. The redefinitions are given in Table 1.

Table 1. The redefinitions of the operators so that the output should always be between 0 and 1 if the inputs are between 0 and 1

Operator	Definition
$\widehat{+}$	(x + y)/2
$\widehat{-}$	$\|x - y\|$
$\widehat{*}$	**None** (If you multiply two numbers between 0 and 1 you will always get a number between 0 and 1.)
$\widehat{sin}$	sin(x)/sin(1)
$\widehat{cos}$	cos(x)/cos(1)
$\widehat{exp}$	exp(x)/exp(1)

The terminal set is composed by several, uniformly distributed, constants between 0 and 1. Our purpose is to design a genetic operator that recombines two parents. Thus, the terminal set should also contain two symbols reserved for parents (x and y).

$$T = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, x, y, R\},$$

where R is actually a function with no arguments that always outputs a random value between 0 and 1. This terminal symbol was added in order to simulate the α parameter from the convex crossover. Note that if the function has multiple dimensions the evolved crossover operator will be applied for each of them.

3.2 The Model

The proposed approach is a hybrid technique divided in two levels: a macro level and a micro level. The macro level is a GP algorithm that evolves crossover operators for function optimization. The micro level is a GA used for computing the quality of a GP individual.

When we compute the quality of a GP chromosome we actually have to compute the quality of the crossover operator encoded in that GP tree. For assessing the performance of a crossover operator we have to embed that operator within an evolutionary algorithm and we have to run the obtained algorithm for a particular problem. Since our problem is a function optimization we embed the evolved crossover within as standard Genetic Algorithm as described in [2], [9].

The fitness of a GP individual is equal to the fitness of the best solution generated by the Genetic Algorithm which uses the GP tree as the crossover operator. But, since the GA uses pseudo-random numbers, it is very likely that successive runs of the same algorithm will generate completely different solutions. This problem can be fixed in a standard manner: the GA embedding the GP individual is run more times (200 runs in fact) and the fitness of the GP chromosome is averaged over all runs.

3.3 The Algorithms

The algorithms used for evolving a crossover operator are described in this section. As we said before we are dealing with a hybrid technique which has a macro level and a micro level.

The Macro-level Algorithm. The macro level algorithm is standard GP algorithm [6] used for evolving crossover operators for function optimization.

We use steady-state evolutionary model as underlying mechanism for our GP implementation. The GP algorithm starts by creating a random population of individuals (trees). The following steps are repeated until a given number of generations is reached: Two parents are selected using a standard selection procedure. The parents are recombined in order to obtain two offspring. The offspring are considered for mutation. The best offspring O replaces the worst individual W in the current population if O is better than W.

The Micro-level Algorithm. The micro level algorithm is a Genetic Algorithm [5] used for computing the fitness of each GP individual from the macro level. The GA starts by creating a random population of individuals. Each individual is a real-valued array whose number of dimensions is equal to the number of dimensions of the function to be optimized. The entire process is run along a fixed number of generations. The best individual in the current population is automatically copied to the next generation. The following steps are repeated until the new population is filled: two parents are selected randomly and are recombined in order to obtain one offspring which will be added to the new population.

We have removed the mutation operator and we have performed random selections. In this way, the performance of the algorithm will mainly be guided by the crossover operator only.

The recombination operator is evolved by the macro level algorithm. During the training stage, the micro-level algorithm is run multiple times and the results are averaged (see section 3.2).

4 Numerical Experiments

In this section, several numerical experiments for evolving crossover operators for function optimization are performed. After evolving it, the crossover operator is embedded into a Genetic Algorithm and used to solve eleven difficult benchmarking problems. Ten functions are artificially constructed and one test problem (the Portfolio Selection Problem) is an important real-world problem (Table 2). Several numerical experiments, with a standard Genetic Algorithm [5] that use a convex crossover for function optimization, are also performed. Finally the results are compared.

The Portfolio Selection Problem. Modern computational finance has its historical roots in the pioneering portfolio theory of Markowitz [7]. This theory is based on the assumption that investors have an intrinsic desire to maximize return and minimize risk on investment. Mean or expected return is employed as a measure of return, and variance or standard deviation of return is employed as a measure of risk. This framework captures the risk-return tradeoff between a single linear return measure and a single convex nonlinear risk measure.

The solution typically proceeds as a two-objective optimization problem where the return is maximized while the risk is constrained to be below a certain threshold. The well-known risk-return efficient frontier is obtained by varying the risk target and maximizing on the return measure.

The Markowitz mean-variance model [7] gives a multi-objective optimization problem, with two output dimensions. A portfolio p consisting of N assets with specific volumes for each asset given by weights w_i is to be found, which minimizes the variance of the portfolio:

$$\sigma_p = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij} \tag{1}$$

maximizes the return of the portfolio:

$$\mu_p = \sum_{i=1}^{N} w_i \mu_i \tag{2}$$

subject to: $\sum_{i=1}^{N} w_i = 1, 0 \leq w_i \leq 1$, where $i = 1...N$ is the index of the asset, N represents the number of assets available, μ_i the estimated return of asset i and σ_{ij} the estimated covariance between two assets. Usually, μ_i and σ_{ij} are to be estimated from historic data. While the optimization problem given in (1) and (2) is a quadratic optimization problem for which computationally effective algorithms exist, this is not the case if real world constraints are added. In this paper we treat only the cardinality constraints problem [11].

Cardinality constraints restrict the maximal number of assets used in the portfolio

$$\sum_{i=1}^{N} z_i = K, \tag{3}$$

where $z_i = sign(w_i)$. Let K be the desired number of assets in the portfolio, ϵ_i be the minimum proportion that must be held of asset i, $(i = 1, ..., N)$ if any of asset i is held, δ_i be the maximum proportion that can be held of asset i, $(i = 1, ..., N)$ if any of asset i is held, where we must have $0 \leq \epsilon_i \leq \delta_i \leq 1 (i = 1, ..., N)$. In practice, ϵ_i represents a "min-buy" of "minimum transaction level" for asset i and δ_i limits the exposure of the portfolio to asset i.

$$\epsilon_i z_i \leq w_i \leq \delta_i z_i, i = 1...N \tag{4}$$

$$w_i \in [0, 1], i = 1...N. \tag{5}$$

Eq. (3) ensures that exactly K assets are held. Eq. (4) ensures that if any of asset i is held $(z_i = 1)$ its proportion w_i must lie between ϵ_i and δ_i, whilst if none of asset is held $(z_i = 0)$ its proportion w_i is zero. Eq. (5) is the integrality constraint. The objective function (Eq. (1)), involving as it does the covariance matrix, is positive semi-definite and hence we are minimizing a convex function. The chromosome - within the GA heuristic - supposes (conform to [3]) a set Q of K distinct assets and K real numbers s_i, $(0 \leq s_i \leq 1)$, $i \in Q$.

Now, given a set Q of K assets, a fraction $\sum_{j \in Q} \epsilon_j$ of the total portfolio is already accounted for and so we interpret s_i as relating to the share of the *free* portfolio proportion $(1 - \sum_{j \in Q} \epsilon_j)$ associated with asset $i \in Q$.

Thus, our GA chromosome will encode real numbers s_i and the proportion of asset i from Q in portfolio will be:

$$w_i = \epsilon_i + \frac{s_i}{\sum_{j \in Q} s_j} (1 - \sum_{j \in Q} \epsilon_j) \tag{6}$$

For this experiment we have used the daily rate of exchange for a set of assets quoted to Euronext Stock [16] during June to December, 2002.

4.1 Experimental Results

We evolve a crossover operator (used by Genetic Algorithm for function optimization) and then we assess its performance by comparing it with the standard convex crossover.

Experiment 1. A crossover operator is evolved in this experiment. For evolving this kind of genetic operator we use a modified version of function f_1 as the training problem. We need this modification to function f_1 because its optimal solution is $x^* = (0, 0, ..., 0)$. This means that a crossover (obtained by evolution) which always outputs value 0 will be able to solve this problem in one generation (in fact after the first crossover operation). An example of this kind of crossover is $x - x$ or $0.3 - 0.3$ or other similar structures. The same issue could appear for all training problems whose optimal solution is an array containing the same constant (e.g. $x^* = (1.56, 1.56... 1.56)$). In all these cases the macro level algorithm could evolve a tree (crossover operator) whose output is always a constant value.

Table 2. Test functions used in our experimental study. The parameter n is the space dimension ($n = 5$ in our numerical experiments) and f_{min} is the minimum value of the function. All functions should be minimized.

Test function	Domain	f_{min}
$f_1(x) = \sum_{i=1}^{n} (i \cdot x_i^2)$.	[-10, 10]^n	0
$f_2(x) = \sum_{i=1}^{n} x_i^2$.	[-100, 100]^n	0
$f_3(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$.	[-10, 10]^n	0
$f_4(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$.	[-100, 100]^n	0
$f_5(x) = \max_i \{x_i, 1 \leq i \leq n\}$.	[-100, 100]^n	0
$f_6(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2$.	[-30, 30]^n	0
$f_7(x) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$	[-5, 5]^n	0
$f_8(x) = -a \cdot e^{-b\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\sum \cos(c \cdot x_i)}{n}} + a + e$.	[-32, 32]^n $a = 20$, $b = 0.2$, $c = 2\pi$.	0
$f_9(x) = \frac{1}{4000} \cdot \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$.	[-500, 500]^n	0
$f_{10}(x) = \sum_{i=1}^{n} (-x_i \cdot \sin(\sqrt{\lvert x_i \rvert}))$	[-500, 500]^n	-n* 418.98
f_{11} = The Portfolio Selection Problem	[0, 1]^n	0

In order to avoid this problem we have to modify the training function. We do this by adding some randomly generated constants to each variable x_i. We obtain the function: $f_1(x) = \sum_{i=1}^{n} (i \cdot (x_i - r_i)^2)$, where r_i are some randomly generated constants between -10 and 10. In this case the optimal solution is $x^* = (r_1, r_2 ... r_n)$.

We have two possibilities for generating the constants: we could keep them fixed during training or we could generate new constants each time the function is called. The second strategy seems to be more general. However, we have tested both strategies, but, for our simple case, both provided similar results.

Note that the modified function is used in the training stage only. During testing stage we use the unmodified (see Table 2) version of the function.

The parameters of the GP algorithm (macro level) are given in Table 3. For GA we use a population with 200 individuals,each of them with 5 dimensions. During 50 generations we apply random selection and recombination (using the evolved crossover) with probability 0.8.

We performed 30 independent runs for evolving operators. In all runs we obtained a very good crossover operator able to compete with the standard

Table 3. The parameters of the GP algorithm (the macro level algorithm) used for evolving genetic operators

Parameter	**Value**
Population size	50
Number of generations	100
Maximum size (depth) for a tree	10
Initialization	Ramped half and half
Crossover probability	0.8
Mutation probability	0.1
Function set	$F = \{\widehat{+}, \widehat{-}, \widehat{*}, \widehat{sin}, \widehat{cos}, \widehat{exp}\}$
Terminal set	$T = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, x, y, R\}$

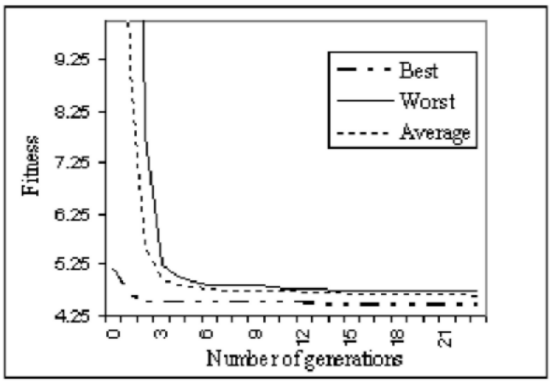

Fig. 2. The evolution of the fitness of the best/worst GP individual, and the average fitness (of all GP individuals in the population) in a particular run. We have depicted a window of 25 generations for a better visualization of the results.

convex crossover. The results obtained in one of the runs (randomly selected from the set of 30 runs) are presented in Figure 2.

Crossover operators of different complexities have been evolved. The simplest operators contain 5 nodes, whereas the most complex evolved operator has 17 nodes. One of the simplest evolved operators is depicted (as GP tree) in Figure 3. We can see that the complexity of the evolved operator is similar to the complexity of the standard convex crossover.

The crossover operator (given in Figure 3) will be used in the numerical experiments performed in the next section.

Experiment 2. This experiment serves our purpose to compare the evolved crossover operator with a convex crossover operator [5]. A Genetic Algorithm [5] is used for testing the quality of the crossover operators. This algorithm has the same structure as the one used in the previous experiment (the micro level algorithm), the only difference being the employed recombination operators. First

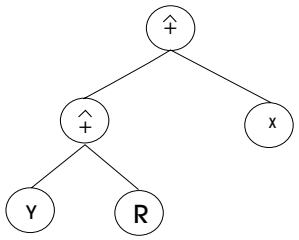

Fig. 3. One of the evolved crossover operators represented as a GP tree. The function symbols have been defined in Table 1.

Table 4. The results obtained by applying the evolved operator and the standard convex crossover to the considered test functions. $Best/Worst$ stands for the fitness of the best individual in the best/worst run. The results are averaged over 500 runs.

Functions	Evolved crossover				Convex crossover			
	Best	Worst	Mean	StdDev	Best	Worst	Mean	StdDev
f_1	0.036	1.157	0.513	0.215	0.093	16.320	2.945	2.490
f_2	3.630	194.718	78.036	32.348	12.796	2884.690	438.121	343.631
f_3	0.415	2.590	1.561	0.342	0.644	9.586	3.589	1.514
f_4	5.263	202.777	76.756	34.022	14.588	4512.660	496.383	524.641
f_5	1.325	9.496	6.030	1.355	1.720	37.528	13.243	5.286
f_6	58.786	4936.8	1198.430	833.183	102	1.65E+06	57400	1.33E+05
f_7	1.387	16.745	8.881	2.600	2.007	34.621	16.877	5.973
f_8	2.681	8.272	5.986	0.895	3.497	17.300	9.719	72.452
f_9	0.557	2.223	1.426	0.250	0.619	19.568	3.576	2.188
f_{10}	-1436.21	-417.259	-849.782	189.805	-1470.00	-454.968	-884.159	198.569
f_{11}	1.49E-04	1.98E-04	1.64E-04	8.75E-06	1.47E-04	3.32E-04	1.87E-04	3.12E-05

Table 5. The results of F-test and t-test

Functions	F-test	t-test	Functions	F-test	t-test
f_1	7.48E-03	2.32E-15	f_7	1.89E-01	5.16E-42
f_2	8.86E-03	4.91E-13	f_8	1.33E-01	1.77E-43
f_3	5.10E-02	5.41E-33	f_9	1.31E-02	1.46E-12
f_4	4.21E-03	2.99E-08	f_{10}	9.14E-01	1.45E-01
f_5	6.57E-02	1.42E-31	f_{11}	3.71E-51	1.96E-143
f_6	3.95E-05	1.77E-03			

we run the GA employing the evolved crossover and later we run the same GA, with the same parameters, using the convex crossover this time. The results of this experiment are presented in Table 4.

Taking into account the average values presented in Table 4 we can conclude that the evolved operator performs significantly better than the classical recombination in 10 out of 11 cases. Taking into account the best values we can see

that the evolved crossover performs better than the convex crossover in 10 cases (out of 11).

In order to determine whether the differences between the evolved crossover and the convex crossover are statistically significant, we use a t-test with a 0.05 level of significance. Before applying the t-test, an F-test is used for determining whether the compared data have the same variance. The P-values of a two-tailed t-test with 499 degrees of freedom are given in Table 5. Table 5 shows that the differences between the results obtained by standard convex crossover and by the evolved crossover are statistically significant ($P < 0.05$) in 9 cases.

5 Conclusions and Further Work

A new hybrid technique for evolving crossover operators has been proposed in this paper. The model has been used for evolving crossover operators for function optimization. Numerical experiments have shown that the evolved crossover performs better than the standard convex crossover for most of the test problems. However, taking into account the No Free Lunch theorems for Search and Optimization [15] we cannot make any assumption about the generalization ability of the evolved crossover operators. Further numerical experiments are required in order to assess the power of the evolved operators. Further work will be focused on: evolving better crossover operators for real encoding, evolving more complex genetic operators which can act as both crossover and mutation, evolving genetic operators for other problems.

References

1. P. J. Angeline, "Two self-adaptive crossover operators for genetic programming", *Advances in Genetic Programming II*, pp. 89-110, MIT Press, 1996.
2. H. J. Bremermann, "Optimization through evolution and recombination", M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, editors, *Self-Organizing Systems 1962*, Proceedings of the Conference on Self-Organizing Systems, Chicago, Illinois, 22.- 24.5.1962, pp. 93-106, 1962.
3. T. -J. Chang, N. Meade, J. E. Beasley, and Y. M. Sharaiha, "Heuristics for cardinality constrained portfolio optimisation" Comp. & Opns. Res. 27, pp. 1271-1302, 2000.
4. B. Edmonds, "Meta-genetic programming: coevolving the operators of variation", *Electrik on AI*, Vol. 9, pp. 13-29, 2001.
5. D. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.
6. J. R. Koza, *Genetic programming, On the programming of computers by means of natural selection*, MIT Press, Cambridge, MA, 1992.
7. H. Markowitz, "Portfolio Selection", *Journal of Finance*, 7, pp. 77-91, 1952.
8. M. Oltean and C. Grosan, "Evolving EAs using Multi Expression Programming", *European Conference on Artificial Life*, pp. 651-658, Springer-Verlag, 2003.
9. H.-P. Schwefel, *Numerical optimization of computer models*, John Wiley & Sons, New York, 1981.

10. L. Spector and A. Robinson, A., "Genetic Programming and Autoconstructive Evolution with the Push Programming Language", *Genetic Programming and Evolvable Machines*, Issue 1, pp. 7-40, Kluwer, 2002.
11. F. Streichert, H. Ulmer, and A. Zell, "Comparing Discrete and Continuous Genotypes on the Constrained Portfolio Selection Problem", *Genetic and Evolutionary Computation Conference - GECCO 2004, Proceedings*, Part II., pp. 1239-1250, Springer-Verlag, 2004.
12. J. Tavares, P. Machado,A. Cardoso, F.-B. Pereira and E. Costa, "On the evolution of evolutionary algorithms", in Keijzer, M. (et al.) editors, *European Conference on Genetic Programming*, pp. 389-398, Springer-Verlag, Berlin, 2004.
13. A. Teller, "Evolving programmers: the co-evolution of intelligent recombination operators", *Advances in Genetic Programming II*, pp. 45-68, MIT Press, USA, 1996.
14. X. Yao, Y. Liu and G. Lin, "Evolutionary programming made faster", *IEEE Transaction on Evolutionary Computation*, pp. 82-102, IEEE Press, 1999.
15. D. H. Wolpert and W. G. McReady, "No Free Lunch Theorems for Search", *Technical Report SFI-TR-05-010*, Santa Fe Institute, USA, 1995.
16. http://www.euronext.com