International Journal on Artificial Intelligence Tools

EVOLVING THE UPDATE STRATEGY OF THE PARTICLE SWARM OPTIMISATION ALGORITHMS

LAURA DIOŞAN *
Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania,
lauras@cs.ubbcluj.ro

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

Received 23 January 2006
Revised June 23, 2006
Accepted .....

A complex model for evolving the update strategy of a Particle Swarm Optimisation (PSO) algorithm is described in this paper. The model is a hybrid technique that combines a Genetic Algorithm (GA) and a PSO algorithm. Each GA chromosome is an array encoding a meaning for updating the particles of the PSO algorithm. The Evolved PSO algorithm is compared to several human-designed PSO algorithms by using ten artificially constructed functions and one real-world problem. Numerical experiments show that the Evolved PSO algorithm performs similarly and sometimes even better than the Standard approaches for the considered problems. The Evolved PSO is highly scalable (regarding the size of the problem's input), being able to solve problems having different dimensions.

Keywords: Swarm Intelligence; Evolutionary Computation; Particle Swarm Optimisation.

1. Introduction

Evolutionary Algorithms (EAs) are very powerful tools used for solving optimisation problems. The major advantage of EAs is given by the possibility of using them for searching in various spaces without performing deep changes in the algorithms structure. They can be easily adapted (by the human or by itself) to the particularities of the problem being solved. Apart from these evolution-motivated computation techniques, a relatively new evolutionary paradigm, called Particle Swarm Optimi-

*Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogalniceanu, 1, Cluj-Napoca, 400 084, Romania

sation (PSO) has been discovered through simplified social model simulation (aimed to provide a better simulation for social behavior).

PSO was developed by Kennedy and Eberhart in 1995[27,28,30,31]. Standard PSO algorithm randomly initializes a group of particles (solutions) and then searches for optima by updating all particles along a number of generations. In each iteration, each particle is updated by following few simple rules[24,40].

The particle swarm algorithm imitates human social behaviour. Individuals interact each other while learning from their own experience, and gradually the population members move into better regions of the problem space. The algorithm is extremely simple - it can be described in one straightforward formula - but it is able to surmount many of the obstacles that optimisation problems commonly present.

Researchers have investigated numerous ways to manipulate the search trajectories of the particles and some of these ways have resulted in improvements and insights. Standard model implies that particles are updated synchronously[24,26]. This means that the current position and speed for a particle are computed taking into account only information from the previous generation of particles.

A more general model allows updating a particle anytime. This basically means three things:

- When a particle is updated the current state of the swarm is taken into account. The best global and local values are computed for each particle which is about to be updated, because the previous modifications could affect these two values. This is different from the Standard PSO algorithm where the particles are updated taking into account only the information from the previous generation. Modifications performed so far (by a Standard PSO) in the current generation has no influence over the modifications performed further in the current generation.
- Some particles may be updated more often than other particles. For instance, in some cases, it is more important to update the best particles several times per generation than to update the worst particles.
- We will work with only one swarm. Any updated particle will replace its parent. Note that two populations/swarms are used in the Standard PSO and the current swarm is filled with information from the previous generation.

The main purpose is to evolve the update strategy of a PSO algorithm. This means that we want to find which particles should be updated and which is the order in which these particles are updated. In this respect, we described a complex technique, which is used for evolving the update strategy of a PSO algorithm. We evolve arrays of integers which provide a meaning for updating the particles within a PSO algorithm during iteration.

Our approach is a hybrid technique that works at two levels: the first (macro) level consists on a steady-state GA[20,44] whose chromosomes encode the update

strategy of PSO algorithms. In order to compute the quality of a GA chromosome we have to run the PSO encoded into that chromosome. Thus, the second (micro) level consists on a modified PSO algorithm that provides the quality of a GA chromosome.

First, the update strategy of a PSO algorithm is evolved and the obtained algorithm is used later for solving eleven difficult function optimisation problems. The Evolved PSO algorithm is compared with two human-designed PSO algorithms (standard and ordered) by using ten artificially constructed functions and one real-world problem. Numerical experiments show that the Evolved PSO algorithm performs similarly and 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 PSO algorithms. The most important question is:

Can a PSO algorithm be automatically synthesized by using only the information about the problem being solved?

And if *yes*, which is the optimal update strategy of a PSO algorithm (for a given problem)? We better let the evolution find the answer for us.

The paper is structured as follows: section 2 provides a brief review of the work on PSO parameters optimisation and on evolving Evolutionary Algorithms(EAs). Section 3 describes, in detail, the model for evolving the PSO update strategy. The test functions are given in section 4.1 and the numerical experiments are performed in section 4.2. Section 5 presents several advantages and weaknesses of the described model. Conclusions and further work directions are suggested in section 6.

2. Related work

Many improvements to the basic form of PSO have been proposed and tested in the literature[11,24,25,29,42]. Also, several analysis of the PSO algorithm behaviour were made[10,32,38,2]. Many of these works are focused on the behaviour convergence. Ozcan and Mohan[38] analyzed the trajectory of a particle in an "original" PSO algorithm (without an inertia weight or a constrict coefficient). Later, Clerc and Kennedy[10] proposed the model based on the constrict coefficient.

Langdon et al.[32] evolved kernel functions which describe the average behaviour of a swarm of particles as if it was responding as a single point moving on a landscape transformed by the kernel. The evolved functions (obtained with genetic programming technique) give another landscape which is "perceived" by a simple hill climber. The goal for the genetic programming is to evolve a kernel which causes the hill climber to move so as to resemble movement of the whole PSO swarm.

Srinivasan[42] proposed a hybrid evolutionary algorithm that combines the concepts of EA and Particle Swarm theory. The aim of the model is to extend PSO algorithm to effectively search in multi-constrained solution spaces, due to the constraints rigidly imposed by the PSO equations. To overcome the constraints, this algorithm replaces the PSO equations completely with a Self-Updating Mechanism,

which emulates the workings of the equations.

Unlike Parsopoulos' work[39] which uses Differential Evolution algorithm (suggested by Storn and Price[43]) for "on the fly" adaptation of the PSO parameters, our work attempts for designing some new rules for the PSO algorithm by taking into account the information from the problem being solved.

Several attempts for evolving Evolutionary Algorithms (EAs) using similar techniques were made in the past. A non-generational EA using the Multi Expression Programming technique[37] was evolved. A generational EA was evolved[36] using the Linear Genetic Programming technique[3,4,5].

3. The model for evolving the update strategy of the PSO algorithm

3.1. *Representation*

Standard PSO algorithm works with a group of particles (solutions) and then searches for optima by updating them during each generation. During iteration each particle is updated by following two "best" values. The first one is the location of the best solution that a particle has achieved so far. This value is called *pBest*. Another "best" value is the location of the best solution that any neighbour of a particle has achieved so far. This best value is a neighbourhood best and called *nBest*.

In the Standard PSO algorithm, all particles are updated once during the course of iteration.

In real-world swarm (such as flock of birds), not all particles are updated in the same time. Some of them are updated more often and others are updated later or not at all. Our purpose is to simulate this, more complex, behaviour. In this case we are interested to discover (evolve) a model which can tell us which particles must be updated and which is the optimal order for updating them.

We will use a GA[20] for evolving this strategy. Each GA individual is a fixed-length string of genes. Each gene is an integer number, in the interval $[0, SwarmSize - 1]$. These values represent indexes of the particles that will be updated during PSO iteration. Some particles could be updated more often and some of them are not updated at all. Therefore, a GA chromosome must be transformed so that it has to contain only the integer values from 0 to Max, where Max represents the number of different genes within the current array.

Suppose that we want to evolve the update strategy of a PSO algorithm with 8 particles. This means that the $SwarmSize = 8$ and all chromosomes of our macro level algorithm will have 8 genes whose values are integer numbers in the [0, 7] range. A GA chromosome with 8 genes can be:

$$C_1 = (2, 0, 4, 1, 7, 5, 6, 3).$$

For computing the fitness of this chromosome, we will use a swarm with 8 individuals and we will perform, during one generation, the following updates:

$$update(Swarm[2]),$$
$$update(Swarm[0]),$$
$$update(Swarm[4]),$$
$$update(Swarm[1]),$$
$$update(Swarm[7]),$$
$$update(Swarm[5]),$$
$$update(Swarm[6]),$$
$$update(Swarm[3]).$$

In this example, all 8 particles have been updated once per generation.

Let us consider another example which consists of a chromosome C_2 with 8 genes that contain only 5 different values:

$$C_2 = (6, 2, 1, 4, 7, 1, 6, 2)$$

In this case, particles 1, 2 and 6 are updated two times each and particles 0, 3 and 5 are not updated at all.

Because of that it is necessary to remove the useless particles from the chromosome and to scale each gene of the GA individual to an integer value from [0, 4] range. The obtained chromosome is:

$$C_2' = (3, 1, 0, 2, 4, 0, 3, 1).$$

The quality for this chromosome will be computed using a 5-size swarm (5 individuals), performing the following 8 updates:

$$update(Swarm[3]),$$
$$update(Swarm[1]),$$
$$update(Swarm[0]),$$
$$update(Swarm[2]),$$
$$update(Swarm[4]),$$
$$update(Swarm[0]),$$
$$update(Swarm[3]),$$
$$update(Swarm[1]).$$

Performing this transformation, we can obtain another swarm, which has a new size. The described model evolves only the update strategy for a PSO algorithm and it can find the optimal swarm size in the same time, even if this parameter is not directly evolved.

We evolve an array of indexes based on the information taken from a function to be optimised. With other words, we evolve an array that contains the update order for the "birds" embedded into the PSO algorithm. This algorithm is used for finding the optimal value(s) of a function. The quality of the update strategy is given by the performance of the PSO algorithm.

Note that the mentioned mechanism should not be based only on the index of the particles in the *Swarm* array. This means that we should not be interested in

updating a particular position since that position can contain (in one run) a very good individual and the same position could hold a very poor individual (during another run). For instance it is easy to see that all GA chromosomes, encoding permutations, perform similarly when averaged over (let's say) 1000 runs.

In order to avoid the problem we sort (after each generation) the $Swarm$ array ascending based on the fitness value. The first position will always hold the best particle at the beginning of a generation. The last particle in this array will always hold the worst particle found at the beginning of a generation. In this way we will know that $update(Swarm[0])$ will mean that the respective particles is not updated, but the best particle at the beginning of the current generation will.

3.2. *Fitness assignment*

The model described in this paper is divided in two levels: a macro level and a micro level. The macro-level is a GA algorithm that evolves the update strategy of a PSO algorithm. For this purpose, we use a particular function as training problem. The micro level is a PSO algorithm, which is used for computing the quality of a GA chromosome from the macro level.

The array of integers encoded into a GA chromosome represents the order of update for particles used by a PSO algorithm that solves a particular problem. We embed the evolved order within a modified Particle Swarm Optimisation algorithm as described in sections 3.1 and 3.3.

Roughly speaking the fitness of a GA individual is equal to the fitness of the best solution generated by the PSO algorithm encoded into that GA chromosome. However, since the PSO algorithm uses pseudo-random numbers, it is very likely that successive runs of the same algorithm will generate completely different solutions. This problem can be handled in a standard manner: the PSO algorithm encoded by the GA individual is run multiple times (50 runs in fact) and the fitness of the GA chromosome is averaged over all runs.

3.3. *The algorithms*

The algorithms used for evolving the PSO update strategy are described in this section. Because we use a hybrid technique that combines a GA and a PSO algorithm within a two-level model, we describe two algorithms: one for macro-level (GA) and another for micro-level (PSO algorithm).

3.3.1. *The macro-level algorithm*

The macro level algorithm is a standard GA[20] used for evolving particles order of update. We use steady-state evolutionary model[44] as underlying mechanism for our GA implementation. The GA algorithm starts by creating a random population of individuals. Each individual is a fixed-length array of integer numbers. The following steps are repeated until a given number of generations is reached: two parents are

selected using a standard selection procedure[34,35]. For performing the selection step of the algorithm we run for two times a "tournament"[35] between two individuals randomly chosen from the population and select the winner (the one with the best fitness). The parents are recombined (using one-cutting point crossover[18,22]) in order to obtain two offsprings O_1 and O_2 - a crossover point is selected in each parent. The genes after the cutting point are swapped between the parents. The offsprings are then considered for weak mutation[23] (The values of one or more genes of each chromosomes are replaced with other randomly generated numbers from [0, $Swarm\_Size$ - 1] range), obtaining $O_1^{'}$ and $O_2^{'}$. The best offspring O^* (out of $O_1^{'}$ and $O_2^{'}$) replaces the worst individual W in the current population if O is better than W[44]. This fact is similar to what happens in nature for longer-lived species where offsprings and parents are alive concurrently and have to compete.

3.3.2. *The micro-level algorithm*

The Micro-level algorithm is a modified Particle Swarm Optimisation algorithm[24] used for computing the fitness of a GA individual from the macro level.

Algorithm 1 One stage PSO Algorithm

```
Initialize the swarm of particles randomly
while not stop_condition do
  for each gene of the GA chromosome do
    Compute fitness of the particle specified by the current gene of the GA chromosome
    if the current fitness value is better than pBest then
      Update pBest
    end if
    Determine nBest for the current particle: choose the particle with the best fitness value of all the neighbours as the nBest
    Update particle's velocity according to Equation (1)
    Update particle's position according to Equation (2)
  end for
  Sort particles based on fitness.
end while
```

$$v_{id} = w * v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * rand() * (p_{nd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

where

- v_{id} represents the velocity of particle i on dimension d;

- w is the inertia weight;
- c_1 and c_2 are two learning factors, which control the influence of *pbest* and *nbest* on the search process. They represent the weighting of the stochastic acceleration terms that pull each particle toward *pbest* and *nbest* positions;
- x_{id} represents the current position for the i^{th} particle on d dimension;
- p_{id} represents the *pBest* value for the i^{th} particle on d dimension;
- p_{nd} represents the *nBest* value on d dimension;
- $rand()$ is a function which generates a random real value between 0 and 1.

The above algorithm is quite different from the Standard PSO algorithm[24].

Standard PSO algorithm works on two stages: one stage that establishes the fitness, *pBest* and *nBest* values for each particle and another stage that determines the velocity (according to Equation (1)) and makes update according to Equation (2) for each particle. Standard PSO usually works with two populations/swarms. Individuals are updated by computing the *pBest* and *nBest* values using the information from the previous population. The newly obtained individuals are added to the current population.

Our algorithm performs all operations in one stage only: determines the fitness, *pBest*, *nBest* and velocity values only when a particle is about to be updated. In this manner, the update of the current particle takes into account the previous updates in the current generation. Our PSO algorithm uses only one population/swarm. Each updated particle will automatically replace its parent.

4. Experiments

Numerical experiments for evolving a PSO algorithm for function optimisation are performed in this section. The obtained PSO algorithm is tested against eleven difficult problems. Several numerical experiments with the Standard PSO algorithm[24] and with the Ordered PSO algorithm are also performed. Finally, the results are compared. We evolve the update strategy of a PSO algorithm and then we assess its performance by comparing it with the Ordered PSO and with the Standard PSO algorithm.

4.1. *Test functions*

Ten artificially test problems[13,14,46] and one real-world problem are used in order to asses the performance of the evolved EA. Functions $f_1 - f_6$ are unimodal test functions. Functions $f_7 - f_{10}$ are highly multi modal (the number of the local minimum increases exponentially with problem's dimension[48]). Function f_{11} corresponds to the constrained portfolio optimisation problem described in that follows. All these functions are given in Table 1.

The Portfolio Selection Problem. Each of the larger fund management companies are responsible for the investment of several billion dollars/euros. This money is invested on behalf of pension funds, unit trusts (mutual funds) and other

institutions. The selection of an appropriate portfolio of assets in which to invest is an essential component of fund management. Although a large proportion of portfolio selection decisions are taken on a qualitative basis, quantitative approaches to selection are becoming more widely adopted.

Briefly, Portfolio Selection Problem (PSP) concerns the selection of the assets weights in a portfolio. The selection process is constrained by the anticipation of investor regarding the level of expected gain (return) and regarding the assumed level of risk.

Modern computational finance has its historical roots in the pioneering portfolio theory of Markowitz[33]. 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 trade-off between a single linear return measure and a single convex nonlinear risk measure.

The solution typically proceeds as a two-objective optimisation 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 the return measure.

The Markowitz mean-variance model[33] gives a multi-objective optimisation problem (a solution for this approach can be found in[15]) 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:

$$\sigma_p = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \sigma_{ij} \tag{3}$$

and maximizes the return of the portfolio:

$$\mu_p = \sum_{i=1}^{N} w_i \mu_i, \text{ subject to: } \sum_{i=1}^{N} w_i = 1, 0 \leq w_i \leq 1, \tag{4}$$

where:

- $i = 1...N$ is the index of the asset;
- N represents the number of available assets;
- μ_i the estimated return of asset i;
- σ_{ij} the estimated covariance between two assets.

Usually, μ_i and σ_{ij} are to be estimated from historic data. While the optimisation problem given in Equations (3) and (4) is a quadratic optimisation problem for which computationally effective algorithms exist[8], this is not the case if real world where the constraints are added. In this paper we treat only the cardinality constraints problem.

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

$$w_i = \epsilon_i + \frac{s_i}{\sum_{j \in Q} s_j}(1 - \sum_{j \in Q} \epsilon_j) \quad (5)$$

For this experiment we have used the daily rate of exchange for a set of assets quoted to Euronext Stock[1] during June to December 2002.

Table 1. Test functions used in our experimental study. The parameter n is the space dimension 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
$f_8(x) = -a \cdot e^{-b\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\sum \cos(c \cdot x_i)}{n}} + a + e.$, $a = 20$, $b = 0.2$, $c = 2\pi$	[-32, 32]^n	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

4.2. *Experimental results*

4.2.1. *Experiment 1*

The update strategy of a PSO algorithm is evolved in this experiment. We use function f_1 as training problem. The parameters of the GA (macro level) are given in Table 2.

The parameters of the PSO algorithm (micro level) are given in Table 3. The *SwarmSize* is not included in this table because different PSOs may have different number of particles. However, the number of function evaluations/generation is equal to 10 for all Evolved PSO.

Table 2. The parameters of the GA algorithm (the macro level algorithm) used for evolving the update strategy of the PSO algorithm

Parameter	Value
Population size	50
Number of generations	50
Number of genes	10
Selection	Binary Tournament
Recombination	One cutting point
Mutation	Weak mutation
Crossover probability	0.8
Mutation probability	0.1

Our algorithm uses a randomized inertia weight selected in the spirit of Clerc's constriction factor[9,16,17] (many reports use a linearly decreasing inertia weight which starts at 0.9 and ends at 0.4, but we want not to restrict our inertia to a fixed model: decreasing or increasing function). Learning factors are not identical. Initially we have used the same values for these parameters ($c_1 = c_2 = 2$), but recent works[6,7] report that it might be even better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with $c_1 + c_2 < 4$.

Table 3. The parameters of the PSO algorithm (the micro level algorithm) used for computing the fitness of a GA chromosome.

Parameter	Value
Number of generations	50
Number of function evaluations/generation	10
Number of dimensions of the function to be optimised	5
Learning factor c_1	2
Learning factor c_2	1.8
Inertia weight	0.5 + rand() / 2

We performed 50 independent runs for evolving the order for particles. The results obtained in one of the runs (randomly selected from the set of 50 runs) are presented in Figure 1.

Different orders of particles have been evolved. The chromosomes represent two of these orders:

$$C_1 = (1, 1, 1, 2, 0, 2, 0, 1, 1, 3)$$

and

$$C_2 = (1, 5, 0, 2, 0, 3, 3, 0, 4, 3)$$

.

The second chromosome (C_2) will be used in the numerical experiments performed in the next section.

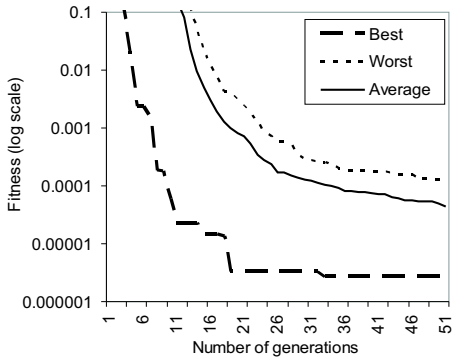

Fig. 1. The evolution of the fitness of the best/worst GA individual, and the average fitness (of all GA individuals in the population) in a particular run.

4.2.2. *Experiment 2*

For assessing the performance of the Evolved PSO, we will compare it with a Standard PSO algorithm using the test functions given in Table 1.

A comparison based on the average over all runs of the best particle from each generation is presented in Figures 2 and 3 on each test function. Figure 2 presents the evolution of the mean-best values during 50 generations for the first six problems that are unimodal test functions. Results for the multi modal test functions are presented in Figure 3.

Table 4. The results obtained by the Evolved PSO algorithm and the Standard PSO algorithm for 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 PSO			Standard PSO		
	Best	Mean	StdDev	Best	Mean	StdDev
f_1	0.01	0.52	0.72	0.28	2.47	1.51
f_2	0.07	0.75	0.71	0.07	1.49	0.99
f_3	0.34	1.38	0.63	0.39	0.84	0.34
f_4	5.54	151.43	134.16	2.36	81.11	127.01
f_5	0.45	0.76	0.41	0.45	1.27	1.26
f_6	11.58	281.54	407.17	4.25	88.37	230.07
f_7	6.79	12.76	4.88	5.74	15.61	5.08
f_8	0.16	1.60	1.24	0.46	1.96	0.68
f_9	8.74	11.91	1.25	3.23	18.04	9.37
f_{10}	- 910.36	- 853.20	30.99	- 1563.89	- 880.47	207.07
f_{11}	0.01	0.78	1.16	0.18	1.28	0.80

In order to make a fair comparison we have to perform the same number of function evaluations in the Evolved PSO and Standard PSO. The Evolved PSO has six particles, but, because some of them are updated more often, it will perform 10

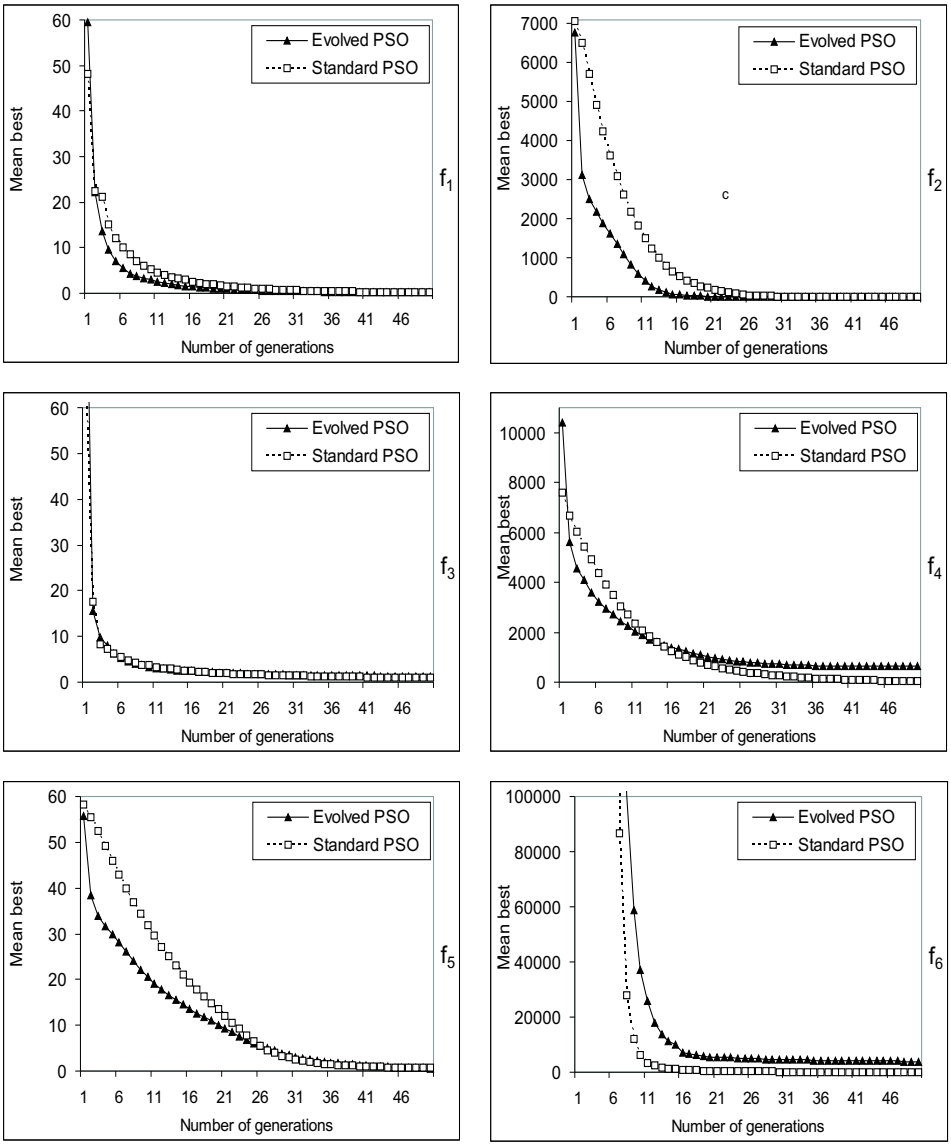

Fig. 2. The evolution of the fitness of the best/worst PSO particle, and the average fitness (of all particles in the swarm) in a particular run.

function evaluations per generation. This means that in each generation 10 updates will be processed, but some particles will be updated more times. Standard PSO[24] has 10 particles and thus it will perform 10 function evaluations/generation (each particle will be updated one time, 10 updates in total). The other parameters of the PSO algorithms are similar to those used by the Evolved PSO and are given in Table 3.

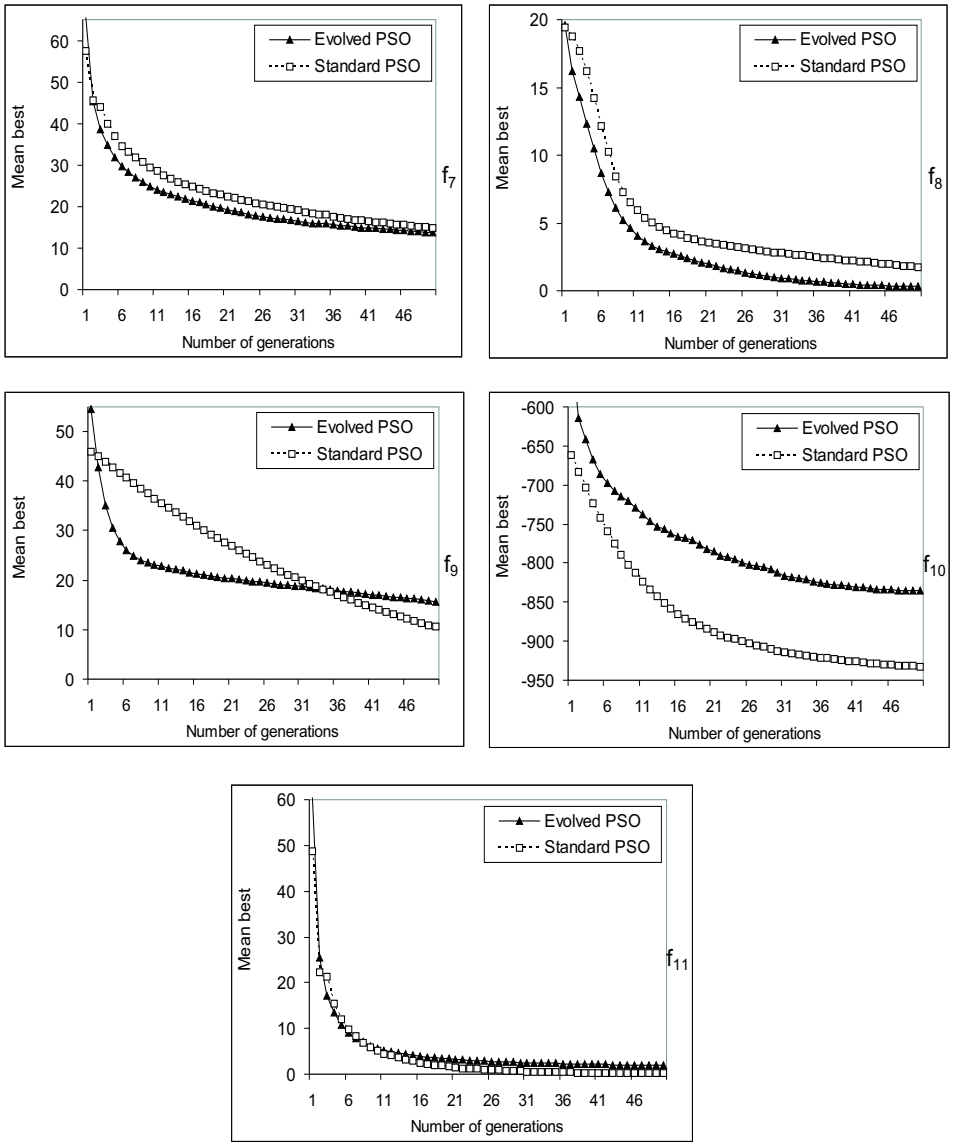

Fig. 3. The evolution of the fitness of the best/worst PSO particle, and the average fitness (of all particles in the swarm) in a particular run.

Taking into account the averaged values we can see in Figures 2, 3 and Table 4 that the Evolved PSO algorithm performs better than the Standard PSO algorithm in 7 cases (out of 11). When taking into account the solution obtained in the best run the Evolved PSO algorithm performs better than the Standard PSO algorithm in 5 cases (out of 11) and tied in 1 case.

In order to determine whether the differences between the Evolved PSO algo-

rithm and the Standard PSO algorithm are statistically significant, we use a t-test with a 0.05 level of significance. Before applying the t-test, an F-test[41] has been used for determining whether the compared data have the same variance. The P-values of the F-test and of the two-tailed t-test with 499 degrees of freedom are given in Table 5.

Table 5. The P-values associated to F-test and to t-test for Experiment 2.

Functions	F-test	t-test
f_1	7.40E-07	5.13E-13
f_2	2.43E-02	2.41E-05
f_3	3.14E-05	3.63E-07
f_4	7.03E-01	4.18E-03
f_5	5.76E-13	3.90E-03
f_6	1.06E-04	2.17E-03
f_7	7.80E-01	2.58E-03
f_8	4.06E-05	3.87E-02
f_9	4.13E-30	6.64E-06
f_{10}	8.55E-28	1.80E-01
f_{11}	9.57E-03	7.12E-03

Table 5 shows that the differences between mean of results obtained by Standard PSO and by the Evolved PSO are statistically significant ($P < 0.05$) in 10 cases (out of 11) and the difference between standard deviations of two result sets are statistically significant in 9 case (out of 11) .

4.2.3. *Experiment 3*

We have also compared the Evolved PSO to another PSO algorithm that updates all particles one by one (i.e. the order of update is 0, 1, ... $SwarmSize - 1$ and the swarm particles are sorted based on their fitness). This algorithm will be called "Ordered PSO".

For this comparison we use the test functions given in Table 1.

The parameters of the Ordered PSO are similar to those used by the Evolved PSO (given in Table 3). Results are presented in Table 6. In 9 cases the Evolved PSO performed better (on average) than the other algorithm.

An F-test[41] is used to verify if the standard deviations of the resulted series are equal. We use a two-tailed test (the two-tailed version tests against the alternative that the standard deviations are not equal). For comparing the means of two results groups (the Evolved PSO algorithm and the Ordered PSO algorithm), we use a t-test with a 0.05 level of significant. The P-values of a F-test and of a two-tailed t-test with 499 degrees of freedom are given in Table 7.

Since for problems f_1, f_2, f_3, f_4, f_5, f_8, f_9, f_{10} the P-values associated to F-test are less than 0.05, we can conclude that there are differences between the two

Table 6. The results obtained by the Evolved PSO algorithm and the Ordered PSO algorithm for 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 PSO			Ordered PSO		
	Best	Mean	StdDev	Best	Mean	StdDev
f_1	0.01	0.52	0.72	0.01	0.48	0.31
f_2	0.07	0.75	0.71	0.29	4.50	9.84
f_3	0.34	1.38	0.63	0.17	0.56	0.27
f_4	5.54	151.43	134.16	0.93	136.16	293.59
f_5	0.45	0.76	0.41	0.49	2.51	3.09
f_6	11.58	281.54	407.17	3.19	134.84	321.91
f_7	6.79	12.76	4.88	2.65	1.69	5.73
f_8	0.16	1.60	1.24	0.30	1.18	0.62
f_9	8.74	11.91	1.25	4.28	16.71	6.95
f_{10}	- 910.30	- 853.20	30.99	- 1324.10	- 848.20	202.31
f_{11}	0.01	0.78	1.16	0.20	2.34	1.31

Table 7. The P-values associated to F-test and to t-test for Experiment 3.

Functions	F-test	t-test
f_1	4.29E-08	3.49E-01
f_2	9.71E-43	4.29E-03
f_3	1.41E-08	1.96E-13
f_4	1.90E-07	3.69E-01
f_5	2.75E-30	7.40E-05
f_6	1.03E-01	2.42E-02
f_7	2.62E-01	4.72E-01
f_8	3.01E-06	1.94E-02
f_9	4.73E-24	2.75E-06
f_{10}	2.54E-27	4.33E-01
f_{11}	4.15E-01	4.83E-09

standard deviations associated to these problem. Since for problem f_6, f_7, f_{11} the P-values are not less than 0.05 we can say that is no significance difference between the two standard deviations associated to these problems.

All P-values for t-test are positive: the first mean (associated to Evolved PSO) is larger than the second (associated to Ordered PSO) for each test problem.

4.2.4. *Experiment 4*

To gain an impression of the average performance of the algorithms on more general data, we analyze their performance under the different values of n (problem size).

We performed more tests taking into account the following values: $n = 10$, $n = 15$, $n = 20$, $n = 25$ and $n = 30$, respectively. The other parameters of the PSO algorithms are similar to those used in previous experiments. The results are presented in Tables 8 and 9.

Table 8. The best solutions founded by the Evolved PSO algorithm and the Standard PSO algorithm for the considered test functions - different values of n were used. The results are averaged over 500 runs.

Functions	$n = 10$		$n = 15$		$n = 20$	
	EvoPSO	StdPSO	EvoPSO	StdPSO	EvoPSO	StdPSO
f_1	15.44	40.55	145.67	33.39	16.53	151.66
f_2	7.43	12.33	13.01	103.93	60.08	271.56
f_3	1.62	0.05	85.01	2.39	1705.30	6.35
f_4	3996.43	1681.00	12614.28	6395.16	30024.95	13118.09
f_5	7.40	6.94	10.29	12.04	14.06	13.57
f_6	1680.11	2317.24	5637.94	490.43	89611.33	2660.63
f_7	21.99	76.62	49.32	143.25	125.55	158.84
f_8	1.14	5.17	4.59	6.48	4.85	5.16
f_9	33.78	36.33	65.02	67.40	93.96	99.40
f_{10}	-1242.45	-1351.49	-1528.96	-1643.70	-1762.75	-1926.68
f_{11}	37.33	32.55	1.43	134.98	393.32	60.32

Table 9. The best solutions founded by the Evolved PSO algorithm and the Standard PSO algorithm for the considered test functions - different values of n were used. The results are averaged over 500 runs.

Functions	$n = 25$		$n = 30$	
	EvoPSO	StdPSO	EvoPSO	StdPSO
f_1	330.52	127.74	863.47	1206.65
f_2	388.02	595.58	352.69	980.55
f_3	40.98	10.30	77.16	12.60
f_4	53806.47	25777.80	81771.98	41989.57
f_5	18.94	15.83	20.02	18.29
f_6	40936.29	30106.47	98885.27	60450.71
f_7	194.04	213.24	209.86	317.95
f_8	5.19	6.19	8.45	7.20
f_9	121.88	133.65	151.39	170.63
f_{10}	-2033.83	-2130.84	-2197.84	-2389.00
f_{11}	60.29	555.29	1379.15	1302.47

The average solution founded by the Evolved PSO algorithm is better than that obtained with Standard PSO algorithm in: 6 cases for $n = 10$ and $n = 15$, 5 cases for $n = 20$ and $n = 25$ and 4 cases for $n = 30$.

For $n = 30$ case, the Boxplots[45] of the obtained results are shown in Figure 4. The median for each dataset is indicated by the center line, and the first and third quartiles are the edges of the box area, which is known as the inter-quartile range. The extreme values (within 1.5 times the inter-quartile range from the upper or lower quartile) are the ends of the lines extending from the inter-quartile range. Points at a greater distance from the median than 1.5 times the inter-quartile range are plotted individually as circles. These points represent potential outliers.

The graphical overview of the data from the Figure 4 indicates that the Evolved

PSO algorithm does not consistently dominate the other, but for several functions it performs better (for functions f_1, f_7, f_9, f_{11} the median value of he data are smaller) or similarly (for functions f_3, f_4, f_6 the results obtained with these algorithms have nearly identical median values).

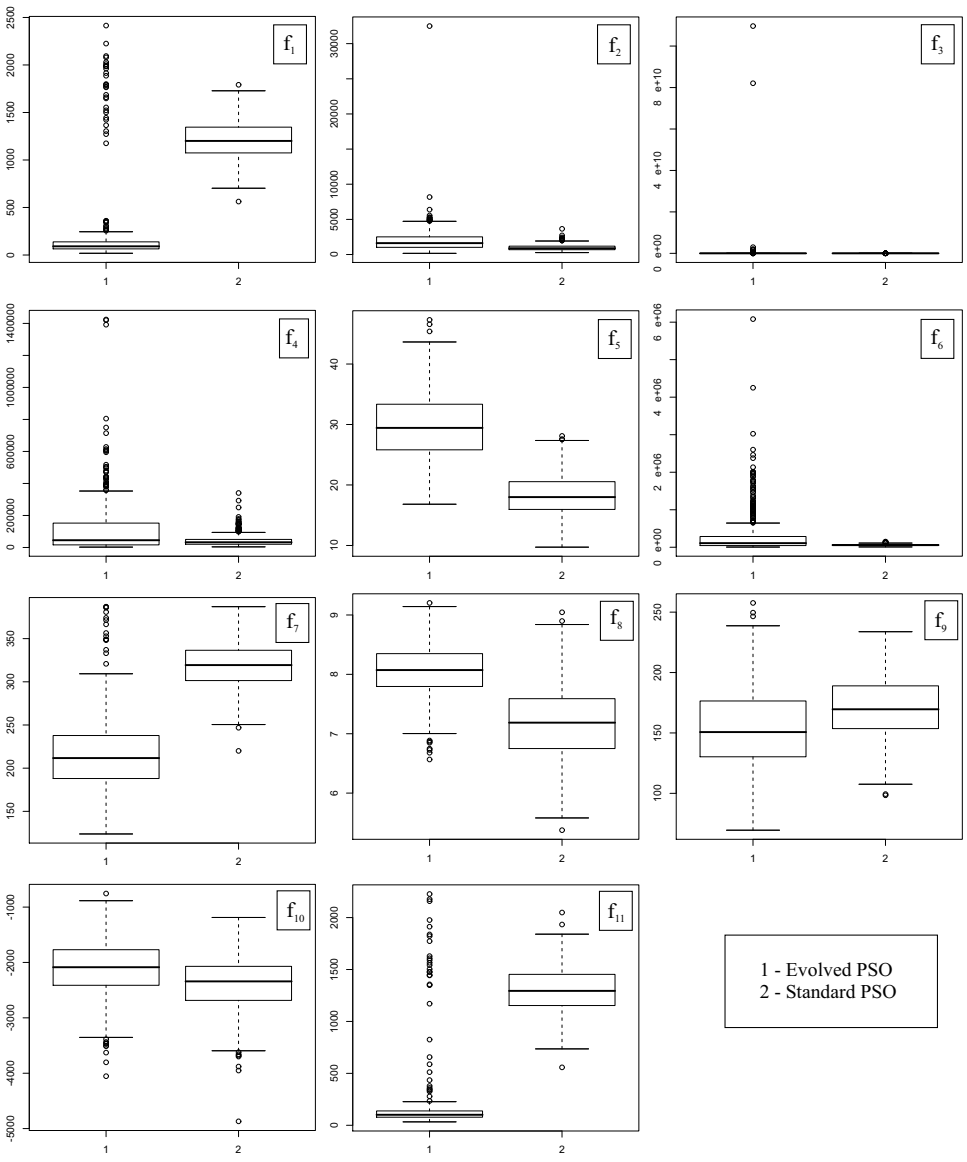

Fig. 4. Boxplots giving the distribution of solution values achieved for 500 runs of each algorithm on the eleven test functions - for each function, the number of dimensions is n = 30.

A F-test and a t-test were used for determining whether the results obtained with

Evolved PSO and Standard PSO algorithms have different level of the performance diversity. The P-values of the F-test and of the two-tailed t-test with 499 degrees of freedom (for $n = 10$, $n = 15$, $n = 20$, $n = 25$, $n = 30$, respectively) are given in Tables 10 and 11. According to these results, the differences between the two tested algorithms are statistically significant ($P < 0.05$) in 5 cases for $n = 10$ and in 3 cases for $n = 15$, $n = 20$, $n = 25$ and $n = 30$, respectively.

Table 10. The P-values associated to F-test and to t-test for Experiment 4.

Functions	$n = 10$		$n = 15$		$n = 20$	
	F-test	t-test	F-test	t-test	F-test	t-test
f_1	2.15E−01	1.06E−063	6.31E−02	5.90E−003	1.75E−01	1.35E−209
f_2	2.96E−04	1.40E−002	7.16E+00	1.52E−052	6.54E+01	8.22E−121
f_3	1.85E−08	1.42E−004	2.40E−08	5.14E−004	5.40E−13	1.19E−001
f_4	8.83E−03	2.03E−017	2.58E−02	2.85E−011	6.27E−02	7.39E−018
f_5	1.95E−01	1.14E−004	6.10E−01	1.20E−022	5.66E−01	6.94E−006
f_6	5.13E−08	7.67E−006	2.12E−09	3.26E−003	3.50E−05	3.40E−009
f_7	2.61E−01	3.26E−130	4.04E−01	0.00E+000	6.01E−01	7.33E−094
f_8	1.93E−02	3.31E−002	2.44E−02	6.27E−012	1.38E+00	3.09E−089
f_9	5.47E−01	8.39E−080	1.04E+00	7.61E−132	5.60E−01	5.24E−011
f_{10}	7.69E−01	1.79E−021	8.43E−01	9.19E−012	8.13E−01	8.27E−007
f_{11}	1.63E−01	2.75E−037	1.12E−01	5.87E−071	5.26E−02	0.00E+000

Table 11. The P-values associated to F-test and to t-test for Experiment 4.

Functions	$n = 25$		$n = 30$	
	F-test	t-test	F-test	t-test
f_1	4.18E−02	0.00E+000	2.75E−01	8.59E−277
f_2	6.46E−02	4.80E−001	5.74E−02	1.40E−030
f_3	1.03E−16	7.62E−002	3.08E−20	7.03E−002
f_4	7.78E−02	1.96E−016	4.78E−02	8.85E−021
f_5	6.36E−01	1.32E−040	3.57E−01	3.71E−205
f_6	2.79E−05	1.87E−044	1.37E−03	1.61E−023
f_7	7.76E−01	1.89E−267	4.49E−01	2.15E−245
f_8	6.56E−01	7.33E−109	2.45E+00	7.88E−104
f_9	5.04E−01	1.05E−001	6.08E−01	1.50E−018
f_{10}	7.65E−01	4.10E−012	8.77E−01	8.31E−017
f_{11}	3.85E−01	2.50E−162	5.18E−01	0.00E+000

5. Strengths and weaknesses

Advantages and weaknesses of the described method are discussed in this section.

5.1. *Strengths*

The described method is able to automatically generate a update strategy for PSO algorithm taking into account the problem to be solved.

The way in which the particles are updated by the described method is more realistic. Updating particles one-by-one rather than generation-based is considered an approach closer to nature because in real-world swarms the individuals do not wait for the entire swarm to be updated before making a new move.

5.2. *Weaknesses*

The time required for evolving an update strategy of the PSO algorithm is quite big in some cases. This is because the PSO algorithm is run each time when its structure is changed. However, this is not such a big problem because we want to generate an algorithm by taking into account only the information about the problem(s) to be solved. We have the problem to be solved and we want to obtain an algorithm for solving it. It might require years for a researcher to derive a good algorithm for the problem. But, by using our approach one can obtain a good algorithm in several hours or several days using an average personal computer.

The method in its current state does not scale up related to the swarm size. The number of particles in the Evolved PSO cannot be changed. For fixing this issue, we plan to experiment with 2 alternatives:

- We will introduce another parameter for each Evolved PSO: the swarm size. This will help us to evolve PSO algorithms with variable population sizes, but after the evolution stage the obtained algorithm will still work with a fixed-size swarm.
- We will try to evolve patterns[36]. Each pattern will be a piece of code, which can be repeatedly applied for generating new individuals. This method has two main advantages: it will reduce the time for generating the algorithm and the obtained PSO algorithm will be able to deal with virtually any swarm size.

6. Conclusion and further work

A new hybrid technique for evolving the update strategy of a PSO algorithm has been described in this paper. The model has been used for evolving the update strategy of the PSO algorithms for optimisation problems. Numerical experiments have shown that the Evolved PSO algorithm performs similarly and sometimes even better than the Standard PSO algorithm for the considered test problems.

Note that according to the No Free Lunch theorems[47] we cannot expect to design a perfect PSO which performs the best for all the optimisation problems. This is why any claim about the generalization ability of the Evolved PSO should be made only based on the results provided by numerical experiments.

Further work will be focused on:

- evolving more parameters of the PSO algorithm (independently, one by one, or synchronously)
- finding patterns in the evolved update strategies. This will help us design PSO algorithms that use larger swarms.
- evolving better PSO algorithms for optimisation.
- evolving PSO algorithms for multi-objective problems[12,19,21].

References

1. Euronext stock exchange, http://www.euronext.com.
2. *Extending Particle Swarm Optimisation via Genetic Programming* (2005), vol. 3447.
3. Brameier, M., and Banzhaf, W. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE-TEVC 5* (2001), 17–26.
4. Brameier, M., and Banzhaf, W. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines 2*, 4 (2001), 381–407.
5. Brameier, M., and Banzhaf, W. Explicit control of diversity and effective variation distance in linear genetic programming. *Lecture Notes in Computer Science 2278* (2002), 37–49.
6. Carlisle, A., and Dozier, G. An off-the-shelf pso. In *Particle Swarm Optimization Workshop* (2001), pp. 1–6.
7. Carlisle, A., and Dozier, G. Tracking changing extrema with particle swarm optimizer. Tech. rep., 2001.
8. Chang, T.-J., Meade, N., Beasley, J. E., and Sharaiha, Y. M. Heuristics for cardinality constrained portfolio optimisation. *Computers and Operations Research 27* (2000), 1271–1302.
9. Clerc, M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Congress on Evolutionary Computation CEC'1999* (1999), pp. 1951–1057.
10. Clerc, M., and Kennedy, J. The particle swarm - explosion, stability, and convergence in a multi-dimensional complex space. *IEEE-TEVC 6* (2002), 58–73.
11. Coello Coello, C. A., and Salazar Lechuga, M. MOPSO: A proposal for multiple objective particle swarm optimization. In *Congress on Evolutionary Computation CEC'2002* (2002).
12. Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation CEC'2002* (2002).
13. DeJong, K. A. *Analysis of Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, The University of Michigan, 1975.
14. DeJong, K. A. Genetic algorithms are NOT function optimizers. In *Foundations of Genetic Algorithms 2* (1993), pp. 5–17.
15. Dioşan, L. A multi-objective evolutionary approach to the portfolio optimization problem. In *International Conference on Computational Intelligence for Modeling Control and Automation, CIMCA 2005* (2005), pp. 183–188.
16. Eberhart, R. C., and Shi, Y. Comparison between genetic algorithms and particle swarm optimization. *Lecture Notes in Computer Science 1447* (1998), 611–616.
17. Eberhart, R. C., and Shi, Y. Particle swarm optimization: Developments, applications, and resources. In *Congress on Evolutionary Computation CEC'2001* (2001), pp. 81–86.
18. Eshelman, L. J. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algo-*

rithms (1991), G. J. E. Rawlins, Ed., Morgan Kaufmann Publishers, pp. 2665–283.
19. Farina, M., Deb, K., and Amato, P. Dynamic multi-objective optimization problems: Test Cases, Approximations, and Applications. *IEEE-TEVC 8*, 5 (Oct. 2004), 425–442.
20. Goldberg, D. E. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.
21. Grosan, C. A comparison of several algorithms and representations for single objective optimization. vol. 3102, pp. 788–789.
22. Holland, J. H. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.
23. Howden, W. E. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering 8*, 4 (July 1982), 371–379.
24. Hu, X., and Eberhart, R. Multi-objective optimization using dynamic neighborhood particle swarm optimization. In *Congress on Evolutionary Computation CEC'2002* (2002).
25. Hu, X., Eberhart, R. C., Shi, Y., and Com, Y. S. Swarm intelligence for permutation optimization: Case study of n-queens problem, June 09 2003.
26. Hu, X., Shi, Y., and Eberhart, R. Recent advances in particle swarm. In *Congress on Evolutionary Computation CEC'2004* (2004), pp. 90–97.
27. Kennedy, J. Minds and cultures: Particle swarm implications. Tech. rep., 1997.
28. Kennedy, J. The behavior of particles. *Lecture Notes in Computer Science 1447* (1998), 581–589.
29. Kennedy, J., and Eberhart, R. A discrete binary version of the particle swarm algorithm. In *World Multiconference on Systemics, Cybernetics and Informatics 1997* (1997), pp. 4104–4109.
30. Kennedy, J., and Eberhart, R. C. Particle swarm optimization. In *IEEE Int. Conf. on Neural Networks* (1995), pp. 1942–1948.
31. Kennedy, J., and Eberhart, R. C. The particle swarm: Social adaptation in information-processing systems. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw-Hill, London, 1999, pp. 379–387.
32. Langdon, W. B., Poli, R., and Stephens, C. R. Kernel methods for PSOs. Tech. Rep. CSM-443, Computer Science, University of Essex, UK, Dec. 2005.
33. Markowitz, H. Portfolio selection. *Journal of Finance 7* (1952), 77–91.
34. Michalewicz, Z. *Genetic algorithms + Data structures = Evolution programs*, third ed. Springer-Verlag, 1996.
35. Miller, B. L., and Goldberg, D. E. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems 9* (1995), 193–212.
36. Oltean, M. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation 13*, 3 (2005), 387–410.
37. Oltean, M., and Grosan, C. Evolving evolutionary algorithms using multi expression programming. vol. 2801, pp. 651–658.
38. Ozcan, E., and Mohan, C. Particle swarm optimization: surfing the waves. pp. 1939–1944.
39. Parsopoulos, K. E., and Vrahatis, M. N. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing 1*, 2-3 (2002), 235–306.
40. Shi, Y., and Eberhart, R. C. Empirical study of particle swarm optimization. In *Congress on Evolutionary Computation CEC'1999* (1999), pp. 1945–1950.
41. Snedecor, G. W., and Cochran, W. G. *Statistical methods*. Iowa State Un.Press, Ames IO, 1967.

42. Srinivasan, D., and Seow, T. H. Particle swarm inspired evolutionary algorithm (PS-EA) for multi-objective optimization problem. In *Congress on Evolutionary Computation CEC'2003* (2003).
43. Storn, R., and Price, K. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, International Computer Science Institute, Berkeley, CA, Mar. 1995.
44. Syswerda, G. A study of reproduction in generational and steady state genetic algorithms. In *Foundations of Genetic Algorithms* (1991), G. J. E. Rawlins, Ed., Morgan Kaufmann Publishers, pp. 94–101.
45. Weisstein, E. Box-and-whisker plot. from mathworld a wolfram web resource : http://mathworld.wolfram.com/box-and-whiskerplot.html.
46. Whitley, D., Mathias, K., Rana, S., and Dzubera, J. Building better test functions. In *Proceedings of the Sixth International Conference on Genetic Algorithms* (1995), pp. 239–246.
47. Wolpert, D. H., and Macready, W. G. No free lunch theorems for optimization. *IEEE-TEVC 1*, 1 (1997), 67–82.
48. Yao, X., Liu, Y., and Lin, G. Evolutionary programming made faster. *IEEE-TEVC 3*, 2 (1999), 82–102.