Breeding swarm: a hybrid PSO and GA

Swarm Intelligence, natcomp.liacs.nl

Xiwen Cheng
Leiden Institute of Advanced Computer Science
Niels Bohrweg 1
Leiden, The Netherlands
xcheng@liacs.nl

ABSTRACT

In this essay Breeding Swarm, a hybrid of Particle Swarm Optimzer and Genetic Algorithm is analyzed by benchmarking it on the Ackley, Griewank and Rastrigrin functions. The reasoning behind combining these two types of optimizers is that PSO is best in exploitation while GA ensures diversity in the population, thus exploration. The main objective is to confirm the effectiveness of combining the two optimizers and what their contributions are to the hybrid algorithm.

1. INTRODUCTION

Breeding Swarm (BS) is a hybrid variant introduced by Settles and Soule [8]. This hybrid combines Particle Swarm Optimizer (PSO) with Genetic Algorithm (GA). They showed it performs as good as their singular counterparts or sometimes even better by comparing the three in training an Artificial Recurrent Neural Network [7].

In this document the effectiveness of BS is tested on four standard test functions. Namely the famous generalized Ackley function [1, 2], Griewank[4], Rastrigin [6] and Rosenbrock [5]. In short the algorithm described in the original paper by Settles and Soule was implemented and tested against the benchmark optimization functions. To verify the hybrid does indeed yield better results, the subsystems *PSO* and *GA* were tested individually where possible. The GA part was improved by introducing *uniform crossover* along with *VPAC* (Velocity Propelled Averaged Crossover) proposed by Settles and Soule.

In sections 2, 3 and 4 the algorithms are outlined. Section 6 lists the test functions used to benchmark the algorithms. Followed by section 7 and 8 describing the used parameters and results respectively. Finally this document is ended with conclusions in section 9.

2. PARTICLE SWARM OPTIMIZER

Particle Swarm Optimizer is inspired by the behavour of bird flocking. It was proposed by Kennedy and Eberhart [3] in 1995. A population, called *swarm* having candidate solutions (*particles*) represents a state of the algorithm. The position of a particle is updated based on its memory influenced by best solutions found so far. It is best described in an algorithm in the next section.

2.1 Standard PSO

It begins with randomly initializing the positions and velocities of each particle in the swarm, being in time $t = 0$. The fitness of each particle in the swarm is evaluated. A personal best *pBest* and a neighbour best *nBest* are selected. In some cases *nBest* is called *gBest* when the algorithm accounts all particles as being neighbours. For a number of iterations, or until an objective criteria is met do the following: Update the velocity of each particle using the equation described in section 2.2. Use the newly calculated velocity to update the particle's location. Evaluate the fitness of the new position, update *pBest* and *nBest* if necessary. See algorithm 1 for the algorithm outline.

Algorithm 1 Standard PSO algorithm

```
$t \leftarrow 0$;
$randomly\ initialize\ V(t)$;
$randomly\ initialize\ P(t)$;
$evaluate\ P(t)$;
$update\ pBest$;
$update\ nBest$;
repeat
  $V(t+1) \leftarrow updateVelocity(V(t))$;
  $P(t+1) \leftarrow P(t) + V(t+1)$;
  $evaluate\ P(t+1)$;
  $update\ pBest$;
  $update\ nBest$
  $t \leftarrow t+1$;
until $stop\ requirement$
```

2.2 Velocity and Position update

The new velocity denotes the direction and speed driving a particle. This is updated based on several social factors: *pBest*, *nBest*, etc... Formally the velocity and position are updated as follow:

$$\begin{aligned} V(t+1) &= w \times V(t) \\ &\quad + c_1 \times rand_1() \times (pBest - P(t)) \\ &\quad + c_2 \times rand_2() \times (nBest - P(t)) \\ P(t+1) &= P(t) + V(t+1) \end{aligned}$$

Where:

$pBest$	=	best solution found so far by the particle
$nBest$	=	best solution found by n neighbourhood
w	=	inertia weight (damping weight)
c_1, c_2	=	social parameters: memory and social respectively
$rand_1$, $rand_2$	=	normal distributed random values

The maximum velocity is enforced using $vMax$. The social parameters dictate how strong the influences of $pBest$ and $nBest$ are. w *scales* the current velocity.

3. GENETIC ALGORITHM

Genetic Algorithm is inspired by evolution. Originally proposed by Holland in 1970's. In GA a population of bitstrings are used. Where a bitstring encodes candidate solutions. An individual is changed by crossover and mutation. The new population is composed of the newly created offspring.

3.1 Standard GA

The population is also initialized randomly and then evaluated. For each consecutive generation mates are selected; they undergo crossover creating a set of offspring. The offspring are then mutated introducing new variants in the population. The offspring become the new population of which their fitnesses are evaluated. See algorithm 2 for the formal outline.

Algorithm 2 Standard GA

```
$t \leftarrow 0$;
$initialize\ P(t)$;
$evaluate\ P(t)$;
**repeat**
    $P'(t) \leftarrow select\text{-}mates(P(t))$;
    $P''(t) \leftarrow crossover(P'(t), p_c)$;
    $P'''(t) \leftarrow mutation(P''(t), p_m)$;
    $evaluate\ P'''(t)$;
    $P(t+1) \leftarrow P'''(t)$;
    $t \leftarrow t+1$;
**until** $stop\ requirement$
```

3.2 Operators

GA has several types of operators. They are *selection*, *crossover* and *mutation*. In turn each type can be implemented in different ways exhibiting different behaviors.

3.2.1 Selection

This operator selects which individual is allowed to mate. The most general forms are proportional where the probability of a individual being selected is proportional to its fitness with regard to the total fitness of the population. Another variant is tournament selection where two individuals compete with each other, the winner (better fitness) is selected.

3.2.2 Crossover

Two parents are randomly selected from the candidate mates and recombined into two offspring. The general types of recombination are single point, two point and uniform. In single point, one point in the bitstring is randomly chosen, cut and swap the substrings. For two points it happens in the same way, but with two crossover points. In uniform each bit is swapped with a probability, mostly $p_c = 0.5$.

3.2.3 Mutation

In mutation each bit in the bitstring is flipped with a probability p_m. This operator guarantees variety in the population and therefore increases explorative ability. A crucial property in finding the global optimum and not getting stuck at a local optimum.

4. BREEDING SWARM

It is important to note that the effectiveness of PSO greatly depends on the initial population. Because particles are driven by their social knowledge, it may never find the global optimum. However, if it searches at the *correct* area (being near the global optimum) it will be able to find better solutions. On the other side GA's are not that great at fine tuning solutions but thrives in being explorative. See table 1 for the comparison between these two. Breeding swarm (BS) combines the ideas of using velocity and position update rules of PSO with the selection, crossover and mutation mechanism of GA.

PSO	**GA**
Directional updates	Omnidirectional mutation
Mixes neighbours	Mixes *random* individuals
Exploitative	Explorative

Table 1: Comparison PSO and GA

4.1 Generic Hybrid Algorithm

Just like other population based optimizers the initial population is randomly initialized. Because PSO is part of the algorithm each particle has its own velocity, hence they are also initialized in the same fashion. The initial population is then evaluated. In the evolution loop a subset of best candidates are selected. A copy of this subset, P_{pso} undergoes velocity and location update as described in PSO. The rest is constructed using GA with crossover and mutation. See algorithm 3.

Where:

N	Number of individuals
N_{elite}	Number of elites
Ψ	*Breeding ratio*, proportion that undergoes breeding; [0.0:1.0]
$select_1, select_2$	Two selection methods, may be different
p_c, p_m	Crossover rate and mutation rate respectively

Algorithm 3 Generic Hybrid algorithm

```
$t \leftarrow 0$;
$initialize\ P(t), V(t)$;
$evaluate\ P(t)$;
**repeat**
    $P_{elite}(t) \leftarrow copyBest(P(t), N_{elite})$;
    $P_{pso}(t) \leftarrow select_1(P_{elite}(t), (N - N_{elite}) * \Psi)$;
    $V'(t) \leftarrow updateVelocity(V(t), P_{pso}(t))$;
    $P'_{pso}(t) \leftarrow updatePosition(P_{pso}(t), V'_{pso}(t))$;
    $P_{ga}(t) \leftarrow select_2(P_{elite}(t), (N - N_{elite}) * (1 - \Psi))$;
    $P'_{ga}(t) \leftarrow crossover(P_{ga}(t), p_c)$;
    $P''_{ga}(t) \leftarrow mutation(P'_{ga}(t), p_m)$;
    $P(t+1) \leftarrow P_{elite} \cup P'_{pso} \cup P''_{ga}$;
    $V(t+1) \leftarrow V'(t)$;
    $evaluate\ P(t+1)$;
    $t \leftarrow t + 1$;
**until** $stop\,requirement$
```

4.2 Breeding Swarm Algorithm

Breeding Swarm is an instantiation of the hybrid algorithm described above. It uses tournament selection and a non-standard crossover operator named VPAC(See section 4.3. The mutation operator mutates each individual with probability p_m. See figure 1 for the evolution loop. It is designed such that GA performs global search and PSO performs local search. Standard velocity and position update rules are used. However:

N_{elite}	$= 0$
Ψ	$= 0.5$, even distribution between PSO and GA
$select_1 = select_2$	= tournament selection with size of 2
$crossover$	$= VPAC$
$mutation$	= Gaussian with mean 0 and variance reduced linearly each generation from 1.0 to 0.0
$inertia\ weight$	= reduced linearly each generation from 0.7 to 0.4
$social\ parameter$	= 2, $c1$ and $c2$ in position update
V_{max}	$= \pm 1$

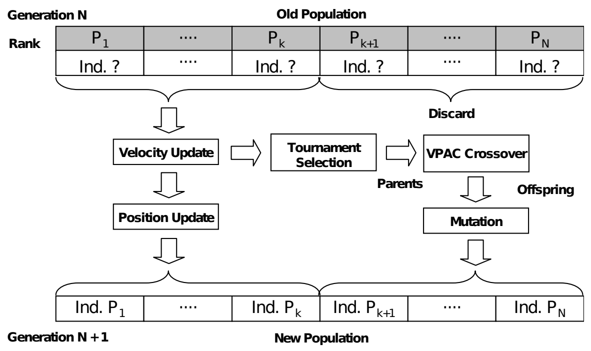


Figure 1: Breeding Swarm [7]

4.3 Velocity Propelled Averaged Crossover

Because GA operators normally operate on bitstrings, a special crossover was developed that uses velocity. VPAC proposed by Settles and Soule takes the overage velocity of two individuals and then update the location according to the following rule:

$$c_1(x_i) = \frac{p_1(x_i) + p_2(x_i)}{2.0} - \varphi_1 p_2(v_i) \qquad (1)$$

$$c_2(x_i) = \frac{p_1(x_i) + p_2(x_i)}{2.0} - \varphi_2 p_1(v_i) \qquad (2)$$

Where:

$c_1(x_i), c_2(x_i)$	Positions of childrens in dimension i
$p_1(x_i), p_2(x_i)$	Positions of parents in dimension i
$p_1(v_i), p_2(v_i)$	Velocities of parents in dimension i
φ_1, φ_2	Uniform random variable in [0.0:1.0]

The idea is to accelerate away from parent's current direction and therefore increase diversity in the population.

5. IMPLEMENTED OPTIMIZERS

To compare the BS proposed by Settles and Soule, some simplified variants of BS were implemented. This is needed to analyze how the each segment of the algorithm performs individually. The algorithms were implemented in a single matlab function:

$$[fxmin, xmin, history] = bs(objfunc, args, dim, n, s, i, vmax, xmax, mode)$$

Where:

$objfunc$	=		name of objective function to be minimized
$args$	=		set of constant arguments passed to $objfunc$
dim	=		dimension of an individual
n	=		population size
s	=		survivors, where:
		$s == 0$	-> GA with VPAC
		$s == n$	-> plain PSO
		$0 < s < n$	-> BS
i	=		number of iterations to run
$vmax$	=		maximum velocity
$xmax$	=		maximum x
$mode$	=		Crossover mode, where:
		$mode == 0$	-> VPAC only
		$mode == 1$	-> uniform crossover only
		$mode == 2$	-> VPAC then uniform crossover
		$otherwise$	-> no crossover at all

And the return values:

$fxmin$	=	best fitness
$xmin$	=	best individual corresponding to $fxmin$
$history$	=	best solution per iteration

The behaviour of the algorithm can be influenced using the s variable and by setting a $mode$. Whenever $s = n$, BS behaves like a normal PSO because no particles undergo breeding. And for $s = 0$ the algorithm assumes no particle is updated by PSO and thus is the equivalence of performing GA operations only. In this case the velocities remain unchanged between generations! $mode$ changes what type of crossover is applied. In uniform crossover, elements of two parents are swapped with probability $p_c = 0.5$ to create new offspring. In total 7 combinations were tested:

GA0	=	no PSO, VPAC only
GA1	=	no PSO, Uniform Crossover only
GA2	=	no PSO, VPAC then Uniform Crossover
PSO	=	no GA
BS0	=	PSO, GA with VPAC only
BS1	=	PSO, GA with Uniform Crossover only
BS2	=	PSO, GA with VPAC then Uniform Crossover

6. TEST FUNCTIONS

In order to analyze the behaviours of the algorithms they were put to the test against four well known test functions. In contrast to the original paper, this hasn't been done.

6.1 Ackley

The Ackley [1] problem is a minimzation problem. It has been generalized to N dimensions [2]:

$$\begin{aligned} F(\vec{x}) &= -20 \cdot exp(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}) \\ &\quad -exp(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(2\pi x_i)) + 20 + exp(1) \end{aligned}$$

Figure 2 shows a plot of the Ackley function in two dimensions. We can observe that it has a large quantity of local optimas with a single global optimum $\vec{x} = (0_1, ..., 0_n)$ with $F(\vec{x}) = 0$.

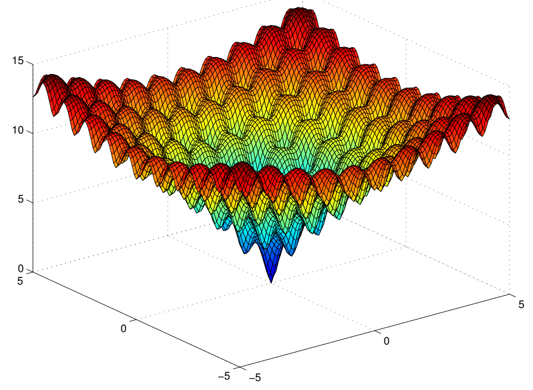


Figure 2: Ackley function in two dimensions

6.2 Griewank

The Griewank test function is also interesting to look at because it has an enormous amount of local minimas with a single optima $\vec{x} = (0_1, ..., 0_n)$, just like the Ackley function:

$$F(\vec{x}) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \frac{cos(x_i)}{\sqrt{i}} + 1$$

Note that the distance between optimas are larger compared to the Ackley function given the same search space size. Figure 3 is plotted on the same scale as the rest.

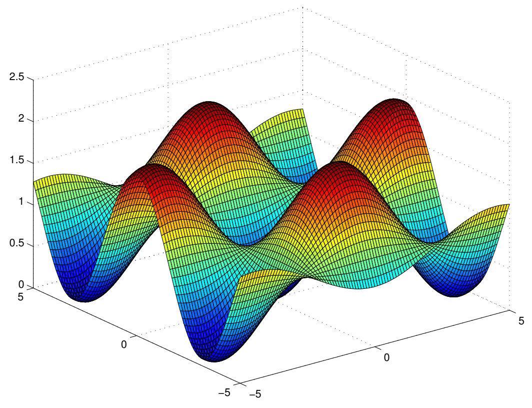


Figure 3: Griewank function in two dimensions

6.3 Rastringrin

The Rastrigrin function is also similar to the previous two. However given the same search space size it contains way more local optimas. So this is very *dense* (See figure 4). Its corresponding equation is as follow:

$$F(\vec{x}) = n \cdot 10 + \sum_{i=1}^{n})[x_i^2 - 10 \cdot cos(2\pi x_i)]$$

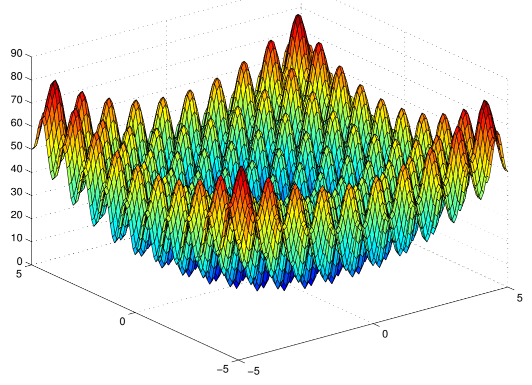


Figure 4: Rastrigrin function in two dimensions

6.4 Rosenbrock

Rosenbrock is the simplest in terms of number of optimas (See figure 5). However they are not trivial to converge to them. The global optimum is the vector $\vec{x} = (1_1, ..., 1_n)$. The function looks as follow:

$$F(\vec{x}) = \sum_{i=1}^{n-1}[100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

7. PARAMETERS

Each function was tested using the algorithms listed in section 5. Some parameters has been set statically while others changed linearly. These parameters are:

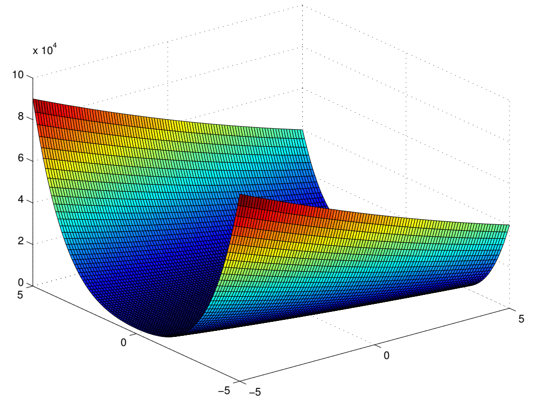


Figure 5: Rosenbrock function in two dimensions

N	=	40, population size
s	=	20, number of survivors
d	=	200, dimensionality
i	=	300, number of iterations
$inertia$	=	decreases from 0.7 to 0.4 proportional to i, inertia weight
p_m	=	decreases from 1 to 0 proportional to i, mutation rate
p_c	=	0.5, crossover rate
$vmax$	=	1, maximum velocity
$xmax$	=	10, maximum values
$runs$	=	20, number of runs per algorithm per function

8. RESULTS

For each test function there are two graphs. One containing the mean plots of each algorith. The other on the right is its corresponding standard deviations. It is remarkable that *BS1* performs best in 4 out of 4 functions (figures 6-9 when it comes to finding good solutions. The Rosenbrock results (figure 9) is very interesting because almost all algorithms converge at the same pace except *PSO*.

Plain *PSO* performs the worse in all cases. This is mainly the case because of the many *traps* it could fall into which it fails to explore the search space for potential better solutions. However the standard deviation of 3 out of 4 (figures 7-9) is fairly high compared to the others. This gives the impression it doesn't exploit its social behavior that well. The reason is probably that it jumps between local optimas of the similar fitnesses. This could be helped by reducing the maximum velocity over time.

The overall convergence speed is highest in all cases at the very beginning. The algorithms begin to stagnate very quickly right after. This is probably the result of p_m and $inertia$ being too low to introduce actual change in the population. This raises the question whether linearly decreasing those parameters is a good idea. Perhaps a logaithmic reduction is more suitable to encourage exploration.

Back to the original question: Does combining PSO with GA really improve the performance? *BS1* performs by far the best, but that's mainly because *GA1* is already very good. Both uses *Uniform Crossover* instead of *VPAC*. For the Ackley and Griewank functions the other pair *BS2* and *GA2* shows slight increase of variane in the standard deviations. *BS0*, proposed by Settles and Soule, decreases the variance compared to *GA0*.

For the four test functions using uniform crossover yields better results then VPAC. Applying uniform crossover after VPAC doesn't seem to enhance the search strategy at all. The performance of Breeding Swarm greatly depends on the performance of the GA subsystem. The difference between a GA and its BS variant is quite small, but there's slight improvement.

9. CONCLUSIONS

In the results reported by Settles and Soule [7] the recurrent neural training exhibits a remarkable behavior: Quickest improvements occur towards the end of the run. However that is not the case in the results reported above. In fact the quickest improvements occur in the beginning of the runs. This is the biggest difference between the findings. As their actual algorithms weren't made public there's a high probability for the implementations to differ.

Nevertheless their original idea is still valid: GA contributes for a great part in exploring the search space while PSO will fine-tune the solutions. Based on above results, it is probably better to distance from linearly decreasing the parameters and use a non-linear function with small slopes at the beginning like $ln(x)$. Or even better yet adopt self-adaptive idea from Evolutionary Strategies.

Combining these is not a bad idea at all. Eventhough PSO doesn't seem to contribute much to the algorithm. Where as it is clear from the results that BS relies heavily on the GA performance. PSO still is able to improve these results further. The margin may be small but noticable.

Instead of mixing PSO and GA within one iterations, it is probably a good idea to *chain* the optimizers. This is interesting when fitness evaluations are expensive. By using GA at the beginning the search space is explored for potential good candidates. Then PSO could optimize these candidates fine tuning the solutions.

10. REFERENCES

[1] D. H. Ackley. *A connectionist machine for genetic hillclimbing.* Kluwer, Boston, 1987.

[2] T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, 1996.

[3] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948, August 2002.

[4] M. Locatelli. A note on the griewank test function. *J. of Global Optimization*, 25:169–174, February 2003.

[5] A. Mulawa and M. Machura. Algorithm 450: Rosenbrock function minimization. *Commun. ACM*, 16:482–483, August 1973.

[6] H. MÃijhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17(6-7):619 – 632, 1991.

[7] M. Settles, P. Nathan, and T. Soule. Breeding swarms:

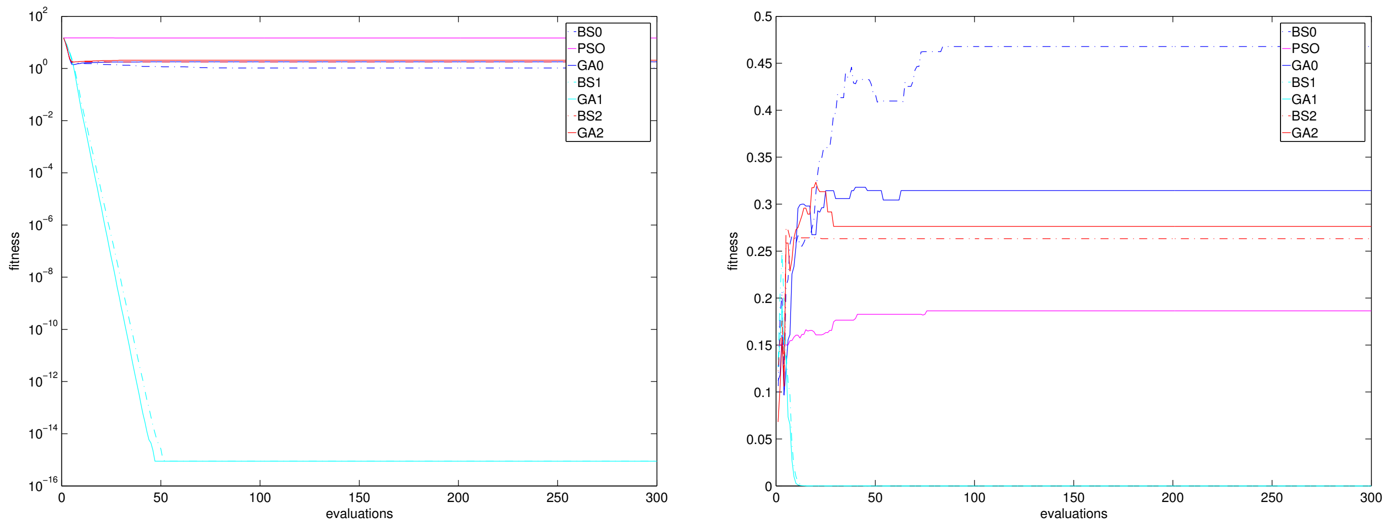


Figure 6: Ackley: mean and standard deviation

a new approach to recurrent neural network training. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 185–192, New York, NY, USA, 2005. ACM.

[8] M. Settles and T. Soule. Breeding swarms: a ga/pso hybrid. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 161–168, New York, NY, USA, 2005. ACM.

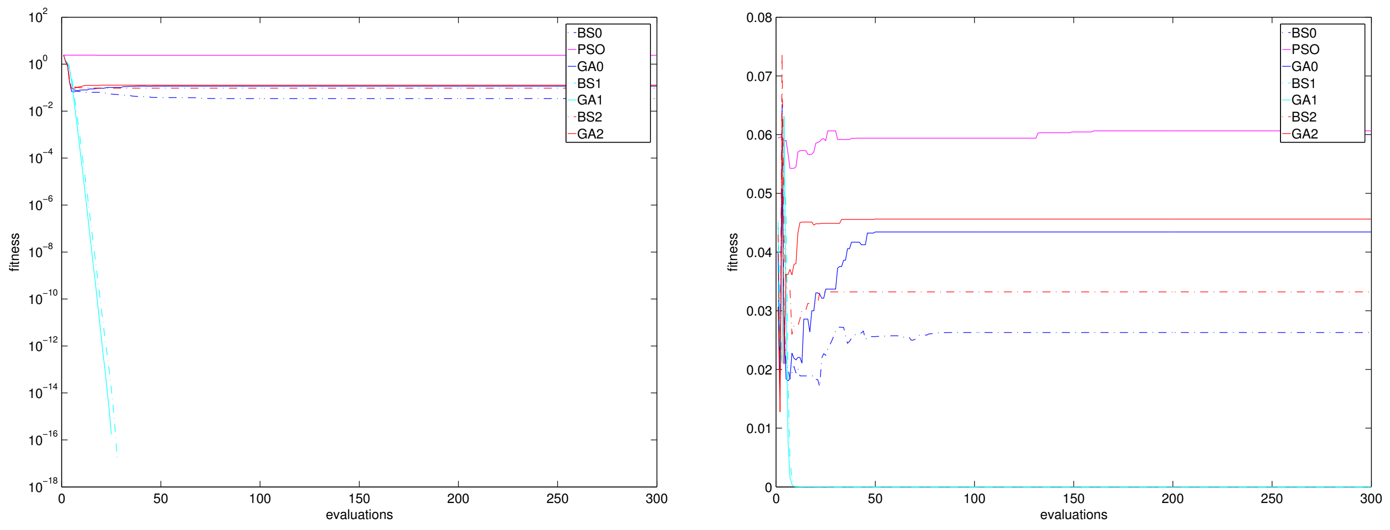


Figure 7: Griewank: mean and standard deviation

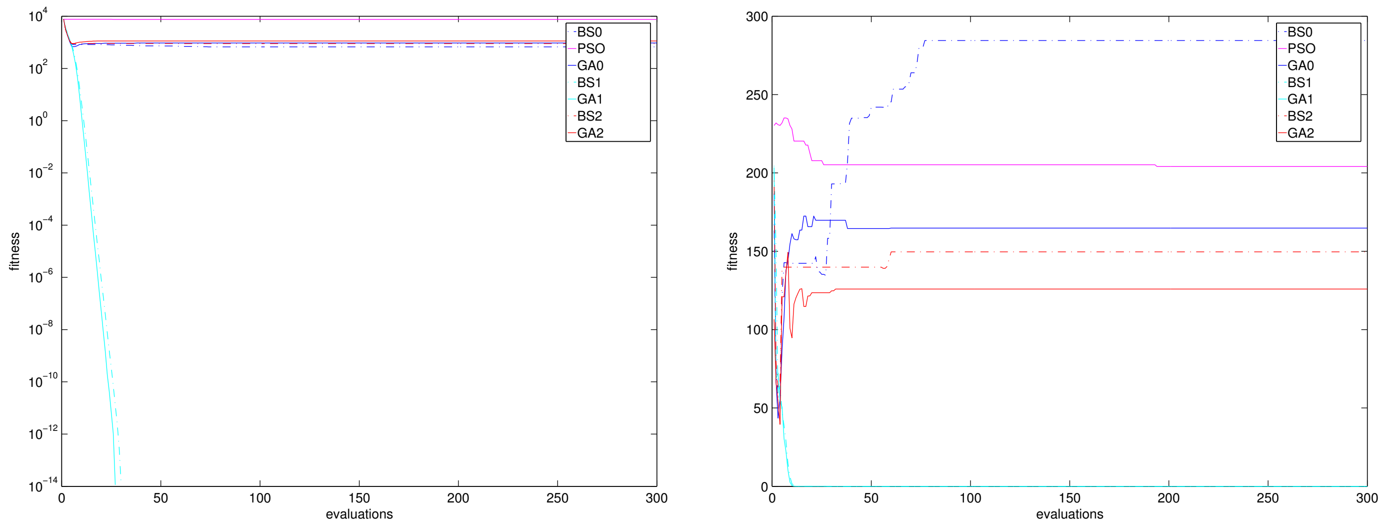
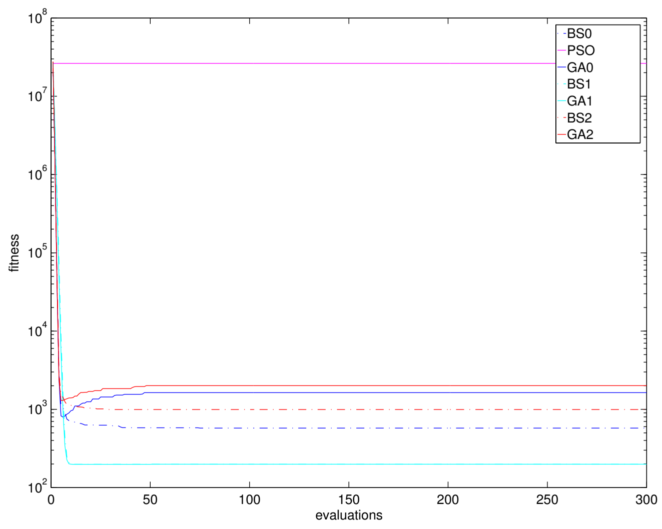


Figure 8: Rastrigrin: mean and standard deviation



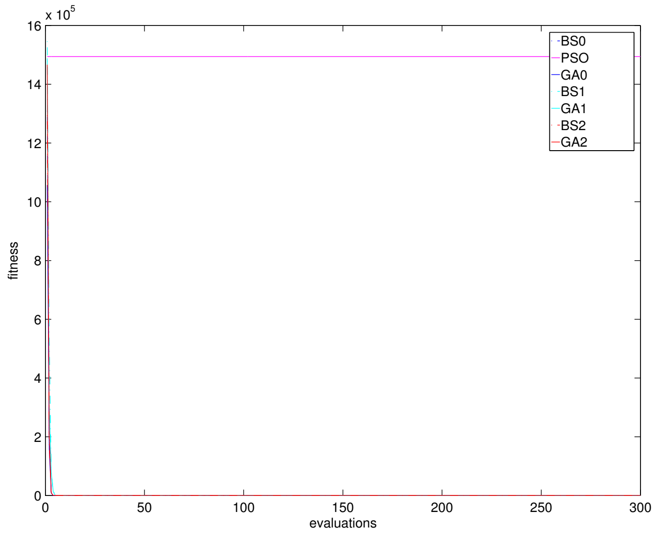


Figure 9: Rosenbrock: mean and standard deviation