https://doi.org/10.1145/1068009.1068035

Breeding Swarms: A GA/PSO Hybrid

Matthew Settles
sett4263@uidaho.edu

Terence Soule
tsoule@uidaho.edu

Department of Computer Science
University of Idaho
Moscow, ID 83843

ABSTRACT

In this paper we propose a novel hybrid (GA/PSO) algorithm, Breeding Swarms, combining the strengths of particle swarm optimization with genetic algorithms. The hybrid algorithm combines the standard velocity and position update rules of PSOs with the ideas of selection, crossover and mutation from GAs. We propose a new crossover operator, Velocity Propelled Averaged Crossover (VPAC), incorporating the PSO velocity vector. The VPAC crossover operator actively disperses the population preventing premature convergence. We compare the hybrid algorithm to both the standard GA and PSO models in evolving solutions to five standard function minimization problems. Results show the algorithm to be highly competitive, often outperforming both the GA and PSO.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

General Terms

Algorithms

Keywords

Breeding Swarms, Particle Swarm Optimization, Genetic Algorithms, Function Minimization

1. INTRODUCTION

Genetic Algorithms (GA) and Particles Swarm Optimization (PSO) are both population based algorithms that have proven to be successful in solving a variety of difficult problems. However, both models have strengths and weaknesses. Comparisons between GAs and PSOs have been performed by Eberhart and Angeline and both conclude that a hybrid of the standard GA and PSO models could lead to further advances [1, 3]. In previous research, a comparison of the two algorithms on recurrent artificial neural networks has led us to the same conclusions [11]. In this paper we present a novel hybrid GA/PSO algorithm, Breeding Swarms (BS), combining the strengths of GAs with those of PSO. The hybrid algorithm is compared to the standard GA and PSO models in evolving solutions to five standard function minimization problems.

The PSO algorithm is conceptually simple and can be implemented in a few lines of code. A PSO individual also retains the knowledge of where in the search space it performed the best, a memory of a past experience. In a GA if an individual is not selected for elitism or crossover, the information contained by that individual is lost. However, without a selection operator PSOs may waste resources on poor individuals. A PSO's group interactions enhances the search for an optimal solution, whereas GAs have trouble finding an exact solution and are best at reaching a near optimal solution [3]. Hybrid GA models incorporating a hill climbing element are often used to overcome this problem [9].

Our hybrid algorithm combines the standard velocity and position update rules of PSOs with the ideas of selection, crossover and mutation from GAs. The algorithm is designed so that the GA facilitates a global search and the PSO performs a local search. Results show the Breeding Swarm algorithm to be highly competitive, often outperforming both the GA and PSO.

The next section discusses previous research and gives an overview of the standard GA and PSO models. Section 3 presents the Breeding Swarm model developed in this research. Section 4 describes the test problems used, initialization methods and results. Finally section 5 discusses the results and presents some conclusions.

2. BACKGROUND

Other hybrid GA/PSO algorithms have been proposed and tested on function minimization problems [8] and [10]. Lvbjerg incorporated a breeding (arithmetic crossover) operator into the PSO algorithm, where breeding occurred inline with the standard velocity and position update rules. Results using this implementation yielded little improvement over the standard PSO algorithm. Robinson tested a hybrid which used the GA algorithm to initialize the PSO population and another in which the PSO initialized the GA population. This approach yielded a small improvement when the GA was used to initialize the PSO algorithm and no improvement when using the PSO to initialize the GA.

In previous work, a GA and PSO were tested and com-

GECCO'05, June 25–29, 2005, Washington, DC, USA.

pared when evolving the weights for a fixed topology Recurrent Artificial Neural Network (RANN) [11]. Both the GA and PSO were successful in evolving weights for a RANN that produced a periodic output in response to a fixed input signal. However, the algorithms produced different results for different network sizes. Thus, demonstrating that they implement different search strategies. We believe that the correct combination of both models has the potential to achieve better results faster and to work well across a wide range of problems.

Initial research in designing an algorithm which incorporated elements of both GA and PSO showed promising results [12]. The initial hybrid algorithm was able to outperform both the GA and PSO in evolving the weights for the same fixed topology RANN. This research expands upon our previous research by more fully integrating the two algorithms.

2.1 Genetic Algorithms

Genetic algorithms were first introduced by Holland in the early 1970's [6] and have been widely successful in optimization problems, especially those in the discrete domain.

A real-valued generational Genetic Algorithm(GA) is used in these experiments. The GA uses a chromosome consisting of real values. The two best individuals are copied into the next generation (elitism). Tournament selection is used, with a tournament of size 3. Gaussian mutation was used, with mean 0.0 and variance reduced linearly each generation from 1.0 to 0.1. Each weight in the chromosome has probability of mutation $1/dimensions$.

The crossover operator chosen for use in this experiment was blended crossover (BLX-α) [4]. In blended crossover, two parents are selected using some selection scheme. Each gene in the offspring is then calculated by randomly choosing a position in the range $[min_i - \Delta \cdot \alpha : max_i + \Delta \cdot \alpha]$. Where $min_i = min(x_i, y_i)$, $max_i = max(x_i, y_i)$ and $\Delta = |x_i - y_i|$. In this experiment α was chosen to be 0.1. The crossover rate is 0.8.

2.2 Particle Swarm Optimizers

As described by Eberhart and Kennedy, the PSO algorithm is an adaptive algorithm based on a social-psychological metaphor; a population of individuals (referred to as particles) adapts by returning stochastically toward previously successful regions [7].

During each generation each particle is accelerated toward the particle's previous best position and the global best position. At each iteration a new velocity value for each particle is calculated based on its current velocity, the distance from its previous best position, and the distance from the global best position. The new velocity value is then used to calculate the next position of the particle in the search space. This process is then iterated a set number of times, or until a minimum error is achieved.

In the inertia version of the algorithm an inertia weight, reduced linearly each generation, is multiplied by the current velocity and the other two components are weighted randomly to produce a new velocity value for this particle, this in turn affects the next position of the particle during the next generation. Thus, the governing equations are:

$$\begin{aligned} \vec{v_i}(t) &= w \cdot \vec{v_i}(t-1) + c_1\varphi_1(\vec{p_i} - \vec{x_i}(t-1)) \\ &\quad + c_2\varphi_2(\vec{p_g} - \vec{x_i}(t-1)) \quad (1) \\ \vec{x_i}(t) &= \vec{x_i}(t-1) + \vec{v_i}(t) \quad (2) \end{aligned}$$

The constriction coefficient version of the algorithm, developed by Clerc [2], uses a constriction coefficient to control the convergence of the algorithm, described below.

$$\begin{aligned} \vec{v_i}(t) &= \chi[\vec{v_i}(t-1) + c_1\varphi_1(\vec{p_i} - \vec{x_i}(t-1)) \\ &\quad + c_2\varphi_2(\vec{p_g} - \vec{x_i}(t-1))] \quad (3) \\ \vec{x_i}(t) &= \vec{x_i}(t-1) + \vec{v_i}(t) \quad (4) \end{aligned}$$

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \ \ where \ \ \varphi = c_1 + c_2, \varphi > 4 \quad (5)$$

Where, $\vec{x_i}$ is particle i's position vector, $\vec{v_i}$ is particle i's velocity vector, $\vec{p_i}$ is particle i's previous best position vector and $\vec{p_g}$ is the global best particles position vector. The parameter w is the inertia weight and χ the constriction coefficient. Variables c_1,c_2, φ_1 and φ_2 are social parameters and random numbers in the range [0.0,1.0], respectively.

Clerc, et al., found that by modifying φ, the convergence characteristics of the system can be controlled [2]. For this test φ was set to 4.1, thus $\chi = 0.73$. Using the recommendations of Shi [13], A limit on the velocity parameter, V_{max}, was set to half the search space.

3. THE BREEDING SWARM MODEL

Both Angeline and Eberhart have suggested that a hybrid combination of the GA and PSO models could produce a very effective search strategy [1,3]. Our goal is to introduce an adjustable hybrid GA/PSO model. Our results show that with the correct combination of GA and PSO, the hybrid can outperform, or perform as well as, both the standard PSO and GA models.

The hybrid algorithm combines the standard velocity and position update rules of PSOs with the ideas of selection, crossover and mutation from GAs. An additional parameter, the breeding ratio (Ψ), determines the proportion of the population which undergoes breeding (selection, crossover and mutation) in the current generation. Values for the breeding ratio parameter range from (0.0:1.0).

In each generation, after the fitness values of all the individuals in the same population are calculated, the bottom $(N \cdot \Psi)$, where N is the population size, are discarded and removed from the population. The remaining individual's velocity vectors are updated, acquiring new information from the population. The next generation is then created by updating the position vectors of these individuals to fill $(N \cdot (1 - \Psi))$ individuals in the next generation. The $(N \cdot \Psi)$ individuals needed to fill the population are selected from the individuals whose velocity is updated to undergo VPAC crossover and mutation and the process is repeated. For clarity, the flow of these operations is illustrated in Figure 1 where $k = (N \cdot (1 - \Psi))$.

The crossover operator used here was developed to utilize information available in the Breeding Swarm algorithm, but not available in the standard GA implementation. The new crossover operator, Velocity Propelled Averaged Crossover

Table 1: Table of parameters for GA, PSO and BS.

Parameter	GA	PSO	BS
Population size	125	125	125
elitism	2 best	N/A	N/A
Selection type	tournament	N/A	tournament
Tournament size	3	N/A	3
Crossover rate	0.8	N/A	N/A
Mutation Rate	$1/dimensions$	N/A	$1/dimensions$
Mutation Variance	$1.0 \rightarrow 0.1$	N/A	$1.0 \rightarrow 0.1$
Social	N/A	2.0	2.0
Inertia	N/A	$0.9 \rightarrow 0.2$	$0.9 \rightarrow 0.2$

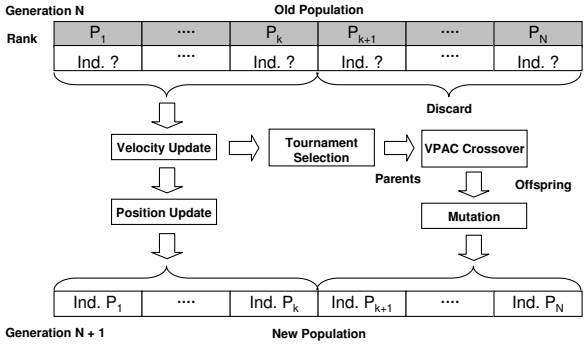


Figure 1: Flow of the Breeding Swarm Algorithm.

(VPAC), incorporates the PSO velocity vector. The goal is to create two child particles whose position is between the parent's position, but accelerated away from the parent's current direction (negative velocity) in order to increase diversity in the population. Equations 6 show how the new child position vectors are calculated using VPAC. Towards the end of a typical PSO run, the population tends to be highly concentrated in a small portion of the search space, effectively reducing the search space. With the addition of the VPAC crossover operator, a portion of the population is always pushed away from the group, increasing the diversity of the population and the effective search space.

$$\begin{aligned} c_1(x_i) &= \frac{p_1(x_i) + p_2(x_i)}{2.0} - \varphi_1 p_1(v_i) \\ c_2(x_i) &= \frac{p_1(x_i) + p_2(x_i)}{2.0} - \varphi_2 p_2(v_i) \end{aligned} \quad (6)$$

Where, $c_1(x_i)$ and $c_1(x_i)$ are the positions of child 1 and 2 in dimension i, respectively. $p_1(x_i)$ and $p_2(x_i)$ are the positions of parents 1 and 2 in dimension i, respectively. $p_1(v_i)$ and $p_2(v_i)$ are the velocities of parents 1 and 2 in dimension i, respectively. φ is a uniform random variable in the range [0.0:1.0]. The child particles retain their parent's velocity vector, $c_1(\vec{v}) = p_1(\vec{v})$,$c_2(\vec{v}) = p_2(\vec{v})$.The previous best vector is set to the new position vector, restarting the child's memory, $c_1(\vec{p}) = p_1(\vec{x})$,$c_2(\vec{p}) = p_2(\vec{x})$.

The Breeding Swarm algorithm is tested using both inertia and constriction coefficient in order to see if there is a performance difference using either technique. The velocity and position update rules remain unchanged from the standard implementation of the PSO. The social parameters are set to 2.0, inertia is linearly decreased from $0.9 \rightarrow 0.2$.

The breeding ratio was set to an arbitrary 0.5, with the expectation that the best results would be with an even mix of the GA and PSO. However, this need not be the case and other values for the breeding ratio may provide better results. All other parameters in the hybrid were kept consistent with the implementations of GA and PSO to remove any bias. Tournament selection is used, with a tournament size of 3, to select individuals as parents. Gaussian mutation was used, with mean 0.0 and variance reduced linearly each generation from 1.0 to 0.1. Each weight in the chromosome has probability of mutation = $1/dims$. Parameters used are summarized in Table 1

4. EXPERIMENTAL RESULTS

Five numeric optimization problems were chosen to compare the relative performance of the Breeding Swarm algorithm to GA and PSO. These functions are standard benchmark test functions and are all minimization problems.

4.1 Test Problems

The first two functions are unimodal, while the remaining three are multimodal. All functions are designed to have a global minima near the origin.

The first test function is the Ellipsoidal function given by the equation:

$$f_1(x) = \sum_{i=1}^{n} i x_i^2 \quad (7)$$

x is a real-valued vector of dimension n and x_i is the ith element in the vector. The second function is the Rosenbrock function given by:

$$f_2(x) = \sum_{i=1}^{n} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \quad (8)$$

The third test function is the generalized Rastrigin function:

$$f_3(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10) \quad (9)$$

The fourth function is the generalized Griewank function:

Table 2: Initialization ranges for test problems.

function	Symmetric Initialization Range	Asymmetric Initialization Range
f_1	$(-100, 100)$	$(50, 100)$
f_2	$(-30, 30)$	$(15, 30)$
f_3	$(-5.12, 5.12)$	$(2.56, 5.12)$
f_4	$(-600, 600)$	$(300, 600)$
f_5	$(-32.768, 32.768)$	$(16.384, 32.768)$

$$f_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \qquad (10)$$

The final function is the Ackley function:

$$f_5(x) = 20 + e - 20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)} \qquad (11)$$

These functions have been widely used in a number of different studies involving evolutionary optimization (see [1], and others.)

4.2 Initialization Method

Often in comparative experiments, researchers choose to set the initial population uniformly distributed about the search space and usually symmetric about the origin. Furthermore, many test functions have global optima at the origin, including those used in this experiment. Fogel and Beyer has shown that this method of initialization can give a false impression in relative performance [5]. Given that the location of the global optima is generally not known, they suggest that the initialization range not include the global optima. For this experiment tests were performed using both symmetric and asymmetric initializations. A detailed description of the initialization parameters for all test function are shown in Table 2.

Each function is ran for 50 trials. Tests using 10 dimensions were allowed to run for 1000 generations. Tests using 20 dimensions were allowed to run for 1500 generations. Finally tests using 30 dimensions were allowed to run for 2000 generations.

4.3 Results

Tables 3 and 4 show the mean best fitness value along with the standard deviation for each test case using symmetric initialization and asymmetric initialization, respectively.

In many test cases the PSO algorithm was able to find near optimal solutions in a majority of trials. For the majority of these cases the population would converge on a suboptimal solution preventing the population from improving. BS did not have this problem, the GA inspired VPAC operator prevented the population from converging on a suboptimal solution by constantly pushing members of the population away from the global best particle.

Figures 2, 3, 4, 5 and 6 show the results of each algorithm in 30 dimensions, using both symmetric initialization and asymmetric initialization. In general the GA performed poorly when using asymmetric initialization. However, using symmetric initialization the GA was able to outperform at least one of the PSO versions in a number of cases, in particular function f_3 in all dimensions and function f_4 in all dimensions (see Table 3). PSO with constriction was able to outperform PSO with inertia in the unimodal test cases, f_1 and f_2. PSO with inertia performed better than PSO with constriction in test case f_3 and the two performing equally well on the remaining test cases f_4 and f_5.

In general, both versions of the BS algorithm, constriction and inertia, outperformed the GA algorithm in every test case. Likewise, the Breeding Swarm algorithm was able to improve on its PSO counterpart in every test case. However, the BS with constriction version was able to perform as well as, or better than BS with inertia in all test cases.

5. CONCLUSIONS

As the results above demonstrate, the performance of Breeding Swarms is competitive with both the GA and PSO. The Breeding Swarm algorithm was able to locate an optimal, or near optimal, solution significantly faster than either GA or PSO. This is most likely due to the VPAC crossover actively dispersing the population, allowing the population to cross greater distances in the search space faster.

The Breeding Swarm algorithm developed here is highly customizable in order to allow future researchers large latitude in implementing the algorithm. Future research will include investigation into different selection types, crossover types and different values for the breeding ratio parameter for its effects on performance. The use of a other mutation operators also needs to be investigated.

An additional advantage of the Breeding Swarm algorithm may be the ability to implement a variable length string to represent potential solutions, a trait not available in the standard PSO implementation. Through the crossover operator, the individual may be allowed to grow or shrink depending on its needs, such as in GAs. Further research will look into this possibility.

6. ACKNOWLEDGMENTS

This work is supported by NSF EPSCoR EPS-0132626. These experiments were performed on a Beowulf cluster built with funds from NSF grant EPS-80935 and a generous hardware donation form Micron Technologies. Thanks to Dr. Hiromoto of the University of Idaho for a conversation that sparked many of these ideas.

7. REFERENCES

[1] P. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto and et al., editors, *Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 601–610. Springer, 1998.

[2] M. Clerc. The swarm and the queen: Towards a determininistic and adaptive particle swarm optimization. In *Congress on Evolutionary Computation (CEC99)*, pages 1951–1957, 1999.

[3] R. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In e. a. V. William Porto, editor, *Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 611–616. Springer, 1998.

[4] L. Eshelman and J. Schaffer. Real-coded genetic algorithms and interval-schemata. In L. D. Whitley,

Table 3: Symmetric initialization results. Summary of mean best results in the final generation for each trial.

Problem	Dims	Gens	GA Mean Best (std-dev)	PSO inertia Mean Best (std-dev)	PSO constriction Mean Best (std-dev)	BS inertia Mean Best (std-dev)	BS constriction Mean Best (std-dev)
f_1	10	1000	2.43E-05 (1.12E-05)	1.12E-46 (3.80E-46)	0 (0)	0 (0)	0 (0)
	20	1500	0.0001172 (3.37E-05)	2.17E-39 (5.54E-39)	1.37E-45 (1.96E-46)	1.69E-40 (5.81E-40)	0 (0)
	30	2000	0.000271 (7.01E-05)	3.59E-27 (1.30E-26)	1.68E-45 (5.61E-46)	2.99E-30 (8.65E-30)	5.33E-46 (9.25E-46)
f_2	10	1000	6.041 (1.319)	2.386 (2.114)	0.6022 (1.128)	0.1374 (0.8999)	3.71E-06 (2.92E-06)
	20	1500	25.86 (22.99)	30.36 (42.41)	9.517 (16.05)	7.412 (1.867)	0.1431 (0.5608)
	30	2000	38.81 (23.74)	57.52 (47.46)	20.18 (24.18)	32.3 (24.1)	6.248 (4.211)
f_3	10	1000	0.001106 (0.0004573)	1.592 (1.053)	2.349 (1.614)	0 (0)	0 (0)
	20	1500	0.002937 (0.0008124)	10.51 (3.337)	20.95 (6.808)	0.4179 (0.6)	0 (0)
	30	2000	0.004746 (0.001144)	24.74 (5.708)	46.17 (10.83)	1.91 (1.721)	1.78E-16 (6.41E-16)
f_4	10	1000	0.03553 (0.03697)	0.06262 (0.02384)	0.05525 (0.02063)	0.05099 (0.09193)	0.01512 (0.0195)
	20	1500	0.01952 (0.0236)	0.026 (0.02631)	0.02581 (0.02332)	0.003152 (0.006287)	0.006055 (0.009193)
	30	2000	0.003944 (0.005308)	0.01495 (0.01623)	0.01105 (0.0148)	0.003989 (0.006765)	0.01749 (0.01986)
f_5	10	1000	0.003014 (0.0005979)	3.11E-15 (2.37E-30)	3.32E-15 (8.44E-16)	2.34E-16 (1.07E-15)	4.06E-16 (1.60E-15)
	20	1500	0.003292 (0.000591)	7.09E-15 (2.09E-15)	5.67E-15 (1.60E-15)	3.25E-15 (9.64E-16)	3.11E-15 (6.96E-16)
	30	2000	0.003587 (0.000474)	1.57E-14 (5.83E-15)	0.2991 (0.5584)	1.93E-14 (5.95E-14)	5.74E-15 (3.50E-15)

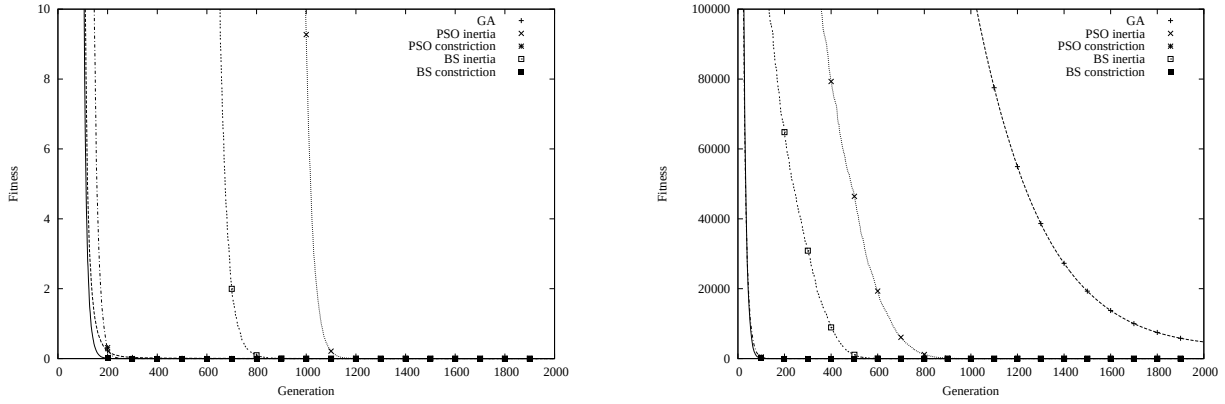
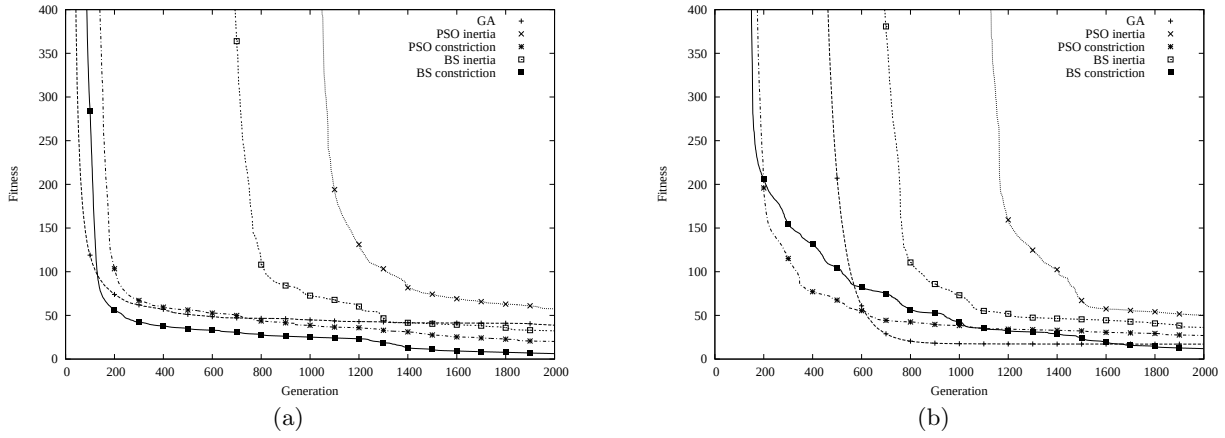


Figure 2: Results for Ellipsoidal function (f_1) in 30 dimensions: (a) Performance of the each algorithm using symmetrical initialization. (b) Performance of each algorithm using asymmetrical initialization.

Table 4: Asymmetric initialization results. Summary of mean best results in the final generation for each trial.

Problem	Dims	Gens	GA Mean Best (std-dev)	PSO inertia Mean Best (std-dev)	PSO constriction Mean Best (std-dev)	BS inertia Mean Best (std-dev)	BS constriction Mean Best (std-dev)
f_1	10	1000	2.52E-05 (-1.11E-05)	0 (0)	0 (0)	0 (0)	0 (0)
	20	1500	5.328 (14.56)	3.10E-39 (1.11E-38)	1.12E-45 (5.61E-46)	1.48E-40 (6.61E-40)	2.80E-47 (1.96E-46)
	30	2000	4788 (1078)	4.66E-27 (2.74E-26)	1.60E-45 (4.86E-46)	6.44E-29 (3.74E-28)	7.85E-46 (1.38E-45)
f_2	10	1000	16.77 (1.543)	8.081 (10.23)	6.622 (11.55)	1.121 (3.914)	0.0117 (0.05151)
	20	1500	17.24 (1.726)	22.75 (26.93)	11.46 (26.83)	19.32 (54.09)	1.75 (3.686)
	30	2000	17.12 (1.647)	50.19 (48.57)	27.01 (33.51)	36.2 (34.47)	11.93 (24.26)
f_3	10	1000	0.001149 (5.236E-4)	1.935 (1.197)	3.741 (1.873)	0 (0)	0 (0)
	20	1500	1.576 (1.816)	9.213 (2.93)	23.54 (8.73)	0.7363 (1.224)	1.07E-16 (4.22E-16)
	30	2000	10.13 (4.469)	23.28 (5.582)	67.48 (21.86)	2.149 (1.704)	1.07E-16 (4.22E-16)
f_4	10	1000	150.3 (6.373)	0.05795 (0.01963)	0.05426 (0.02393)	0.04 (0.03556)	0.01334 (0.01118)
	20	1500	416.9 (11.17)	0.0268 (0.02205)	0.02493 (0.02464)	0.00467 (0.009937)	0.007278 (0.01246)
	30	2000	708.8 (14.81)	0.01549 (0.01718)	0.0117 (0.01477)	0.003202 (0.006865)	0.01797 (0.01885)
f_5	10	1000	19.78 (0.04309)	2.009 (6.028)	0.7799 (3.821)	2.03E-17 (1.30E-15)	2.64E-16 (1.52E-15)
	20	1500	19.81 (0.0249)	2.801 (6.943)	5.086 (8.583)	3.25E-15 (9.64E-16)	3.39E-15 (1.15E-15)
	30	2000	19.83 (0.0218)	5.604 (8.987)	11.9 (9.473)	9.86E-15 (9.52E-15)	6.38E-15 (2.56E-15)



(a) (b)

Figure 3: Results for Rosenbrock function (f_2) in 30 dimensions: (a) Performance of the each algorithm using symmetrical initialization. (b) Performance of each algorithm using asymmetrical initialization.

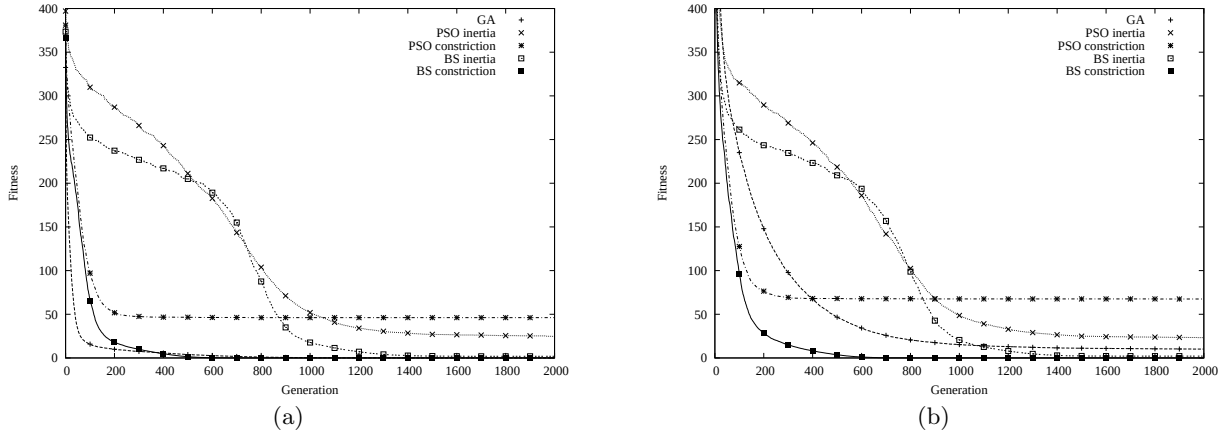


Figure 4: Results for Griewank function (f_3) in 30 dimensions: (a) Performance of the each algorithm using symmetrical initialization. (b) Performance of each algorithm using asymmetrical initialization.

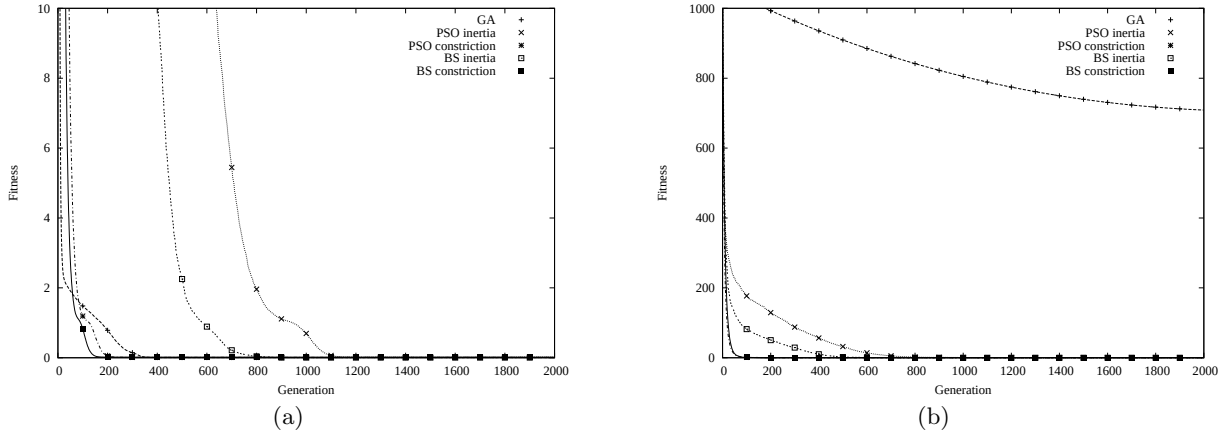


Figure 5: Results for Rastrigin function (f_4) in 30 dimensions: (a) Performance of the each algorithm using symmetrical initialization. (b) Performance of each algorithm using asymmetrical initialization.

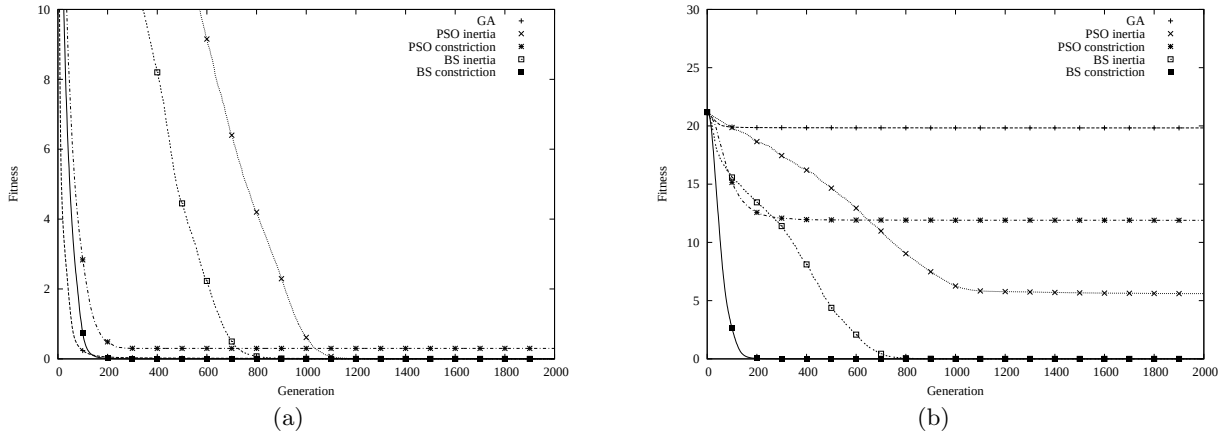


Figure 6: Results for Ackley function (f_5): (a) Performance of the each algorithm using symmetrical initialization. (b) Performance of each algorithm using asymmetrical initialization.

editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, San Mateo, CA, 1993.

[5] D. Fogel and H. Beyer. A note on the empirical evaluation of intermediate recombination. *Evolutionary Computation*, 3(4):491–495, 1996.

[6] J. Holland. *Adaptation in Natural and Artificial Systems: 2nd ed.* University of Michigan Press, Ann Arbor, MI, 1992.

[7] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2001.

[8] M. Lvbjerg, T. Rasmussen, and T. Krink. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-2001*, LNCS. Springer-Verlag, 2001.

[9] G. Mahinthakumar and M. Sayeed. Hybrid genetic algorithm - local search approaches for groundwater source identification problems. *Special Issue on Evolutionary Computation, ASCE Jounal of Water Resources Planning and Management, In Press*, 2004.

[10] J. Robinson, S. Sinton, and Y. Rahmat-Samii. Particle swarm, genetic algorithm, and their hybrids: Optimization of a profiled corrugated horn antenna. In *IEEE Antennas and Propagation Society International Symposium and URSI National Radio Science Meeting*, San Antonio, TX, 2002.

[11] M. Settles, B. Rodebaugh, and T. Soule. Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. In E. Cantú-Paz and et. al., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 148–149, Chicago, 12-16 July 2003. Springer-Verlag.

[12] M. Settles and T. Soule. A hybrid ga/pso to evolve artificial recurrent neural networks. In C. Dagli and et. al., editors, *Intelligent Engineering Systems Through Aritificial Neural Networks (ANNIE-2003)*, volume 13, pages 51–56, St. Louis, 2-5 Nov. 2003. ASME Press.

[13] Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. In V. W. Porto and et al., editors, *Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 591–600. Springer, 1998.