Darwinian Particle Swarm Optimization.

Conference Paper · January 2005

Source: DBLP

CITATIONS
92

READS
271

4 authors, including:

Jason C Tillett
University of Rochester
24 PUBLICATIONS **276** CITATIONS
SEE PROFILE

Ferat Sahin
Rochester Institute of Technology
132 PUBLICATIONS **1,873** CITATIONS
SEE PROFILE

Raghuveer Rao
Army Research Laboratory
186 PUBLICATIONS **4,018** CITATIONS
SEE PROFILE

Darwinian Particle Swarm Optimization

Jason Tillett[1], T.M. Rao[2], Ferat Sahin[3] and Raghuveer Rao[3]

[1]University of Rochester
Rochester, NY USA
[2]SUNY Brockport
Brockport, NY USA
[3]Rochester Institute of Technology
Rochester, NY USA

Abstract. Particle Swarm Optimization (PSO), an evolutionary algorithm for optimization is extended to determine if natural selection, or survival-of-the-fittest, can enhance the ability of the PSO algorithm to escape from local optima. To simulate selection, many simultaneous, parallel PSO algorithms, each one a swarm, operate on a test problem. Simple rules are developed to implement selection. The ability of this so-called Darwinian PSO to escape local optima is evaluated by comparing a single swarm and a similar set of swarms, differing primarily in the absence of the selection mechanism, operating on the same test problem. The selection process is shown to be capable of evolving the best type of particle velocity control, which is a problem specific design choice of the PSO algorithm.

1. Particle Swarm Optimization (PSO)

The PSO [1] approach utilizes a cooperative swarm of particles, where each particle represents a candidate solution, to explore the space of possible solutions to an optimization problem. Each particle is randomly or heuristically initialized and then allowed to 'fly'. At each step of the optimization, each particle is allowed to evaluate its own fitness and the fitness of its neighboring particles. Each particle can keep track of its own solution, which resulted in the best fitness, as well as see the candidate solution for the best performing particle in its neighborhood. At each optimization step, indexed by t, each particle, indexed by i, adjusts its candidate solution (flies) according to,

$$\begin{aligned} \vec{v}_i(t+1) &= \vec{v}_i(t) + \phi_1(\vec{x}_{i,p} - \vec{x}_i) + \phi_2(\vec{x}_{i,n} - \vec{x}_i) \\ \vec{x}_i(t+1) &= \vec{x}_i(t) + \vec{v}_i(t+1) \end{aligned} \tag{1}$$

Eqn. 1 may be interpreted as the 'kinematic' equation of motion for one of the particles (test solution) of the swarm. The variables in the dynamical system of Eqn. 1 are summarized in Table 1.

Table 1- List of variables used to evaluate the dynamical swarm response

$\vec{v}_i$	The particle velocity.
$\vec{x}_i$	The particle position (test solution).
t	Time
ϕ_1	A uniform random variable usually distributed over [0,2].
ϕ_2	A uniform random variable usually distributed over [0,2].
$\vec{x}_{i,p}$	The particle's position (previous) that resulted in the best fitness so far.
$\vec{x}_{i,n}$	The neighborhood position that resulted in the best fitness so far.

Eqn. 1 can be interpreted as follows. Particles combine information from their previous best position and their neighborhood best position to maximize the probability that they are moving toward a region of space that will result in a better fitness. The uniformly distributed random variables, ϕ_1 and ϕ_2 are sampled for each i, t and dimension of the vector $\vec{x}_i$.

2. Darwinian Particle Swarm Optimization

A general problem with optimization algorithms is that of becoming trapped in a local optimum. A particular algorithm may work well on one problem but may fail on another problem. If an algorithm could be designed to adapt to the fitness function, adjusting itself to the fitness landscape, a more robust algorithm with wider applicability, without a need for problem specific engineering would result. Strategies for avoiding local optima include *stretching* of Parsopoulos[2] and other convexification[3] strategies. Nature points to a way that may help circumvent local optima. We propose a strategy based on natural selection in which, when a search tends to a local optimum, the search in that area is simply discarded and another area is searched instead. This is the type of search designed and analyzed in this paper.

In a typical implementation of PSO, a single swarm of test solutions is utilized. To implement natural selection with a single swarm, the algorithm must detect when stagnation has occurred. Since a single swarm is unable to differentiate between a global optimum and a local optimum it cannot simply be extended to model natural selection. One could "time-out" the optimization and restart the algorithm[4] or delete information about the current global optimum in hopes that the swarm will not return to it. Angeline[5] implemented a type of selection process. At the end of each swarm update, the current fitnesses of the particles are used to order the particles. The top half of the particles are then duplicated and replace the positions and velocities of the bottom half of the particles. The personal bests of the particles are not changed. The author is able to achieve better convergence on some test problems.

In search of a better model of natural selection using the PSO algorithm, we formulate what we call a Darwinian PSO, in which many swarms of test solutions may exist

at any time. Each swarm individually performs just like an ordinary PSO algorithm with some rules governing the collection of swarms that are designed to simulate natural selection. The selection process implemented is a selection of swarms within a constantly changing collection of swarms.

2.a. Natural Selection in PSO

The basic assumptions made to implement Darwinian PSO are:

- The longer a swarm lives, the more chance it has of possessing offspring. This is achieved by giving each swarm a constant, small chance of spawning a new swarm.
- A swarm will have its life-time extended (be rewarded) by finding a more fit state.
- A swarm will have its life-time reduced (be punished) for failing to find a more fit state.

These simple ideas implement an algorithm imitating natural selection. In nature, individuals or groups that possess a favorable adaptation are more likely to thrive and procreate. The favorable adaptation is assumed to prolong the lifetime of the individual. Unfavorable adaptations shorten the lifespan of an individual or group.

The single swarm PSO algorithm possesses free parameters that can be adjusted to optimize the algorithm to a specific problem. Extending the PSO algorithm to multiple swarms will expand the parameter set and complicate the task of parameter selection. Particle birth and death within a swarm and swarm birth and death within the collection of swarms must be characterized. Under what conditions should a new particle in a swarm be created? Should the swarm be fixed in size? When should a new swarm be created? When should nature kill an existing swarm? All of these questions introduce complexity into the possible number of implementations.

We will learn, if allowing multiple swarms to represent possible evolutionary tracks of parallel implementations of PSO, whether or not selection, as analogous to natural selection, can help to circumvent local optima.

2.b. Darwinian PSO—The Algorithm Details

2.b.i. Particle and swarm initialization

Each PSO particle is an array of N numbers; the array could contain a binary string[6]. The choice of the domain of the particle array elements, $\vec{x}_i$ as well as the encoding of the test solution as an array of numbers is motivated by the particular optimization problem. The discussion of these details is therefore deferred until the test problems are discussed. Each dimension of each particle is randomly initialized on an appropriate range $\vec{x}_{\min} \le \vec{x}_i \le \vec{x}_{\max}$. The velocities are also randomly initialized on a range, $\vec{v}_{\min} \le \vec{v}_i \le \vec{v}_{\max}$, that allows particles to traverse a significant fraction of the

range of $\vec{x}_i$ in a single iteration when moving at $|\vec{v}_i| = v_{\max}$. Note that when a particle is created, its velocity is randomized to encourage exploration. Each swarm is initialized with a population of particles.

2.b.ii. The Algorithm

At each step of the algorithm, labeled *Main Program Loop* in the pseudo code below, the *Evolve Swarm* algorithm, also shown in the pseudo code below, operates on each swarm. After evolving each swarm, each is allowed to spawn a new swarm with a fixed probability as discussed in Section 2.b.v.. After spawning, the selection process is executed. All swarms that are no longer progressing are deleted.

To evolve an individual swarm, the fitnesses of all of the particles in the swarm are evaluated. The neighborhood and individual best positions of each of the particles are updated. The swarm spawns a new particle if a new global best fitness is found. A particle is deleted if the swarm has failed to find a more fit state in an allotted number of steps. The details of how many steps are allowed before a particle is deleted is discussed in Section 2.b.v..

Main Program Loop (1 step)

```
For each swarm in the collection
    Evolve the swarm (Evolve
    Swarm Algorithm: right)
For each swarm in the collection
    Allow the swarm to spawn
Delete "failed" swarms
```

Evolve Swarm Algorithm

```
For each particle in the swarm
    Update Particle Fitnesses
For each particle in the swarm
    Update Particle Bests
For each particle in the swarm
    Move Particle
If swarm gets better
    Reward swarm : spawn particle : extend
    swarm life
If swarm has not improved
    Punish swarm : possibly delete particle :
    reduce swarm life
```

2.b.iii. Condition for deleting a swarm

A swarm's particle population, m is bounded such that, $m_{\min} \le m \le m_{\max}$. When a swarm's population falls below $m_{\min}$, the swarm is deleted.

2.b.iv. Condition for deleting a particle

The worst performing particle in the swarm is deleted using the following algorithm. The number of times a swarm is evolved without finding an improved fitness is tracked with a search counter, SC. If the swarm's search counter exceeds a maximum critical threshold, $SC_c^{\max}$, a particle is deleted from the swarm. When a swarm is created, its search counter is set at zero. When a particle is deleted, the swarm's search counter is reset not to zero but to a value approaching $SC_c^{\max}$ as the time during which the swarm makes no improvement in fitness increases. The purpose of this reduction

in tolerance for stagnation is to try to maintain a collection of swarms that are actively improving. If N_{kill} is the number of particles deleted from the swarm over a period in which there is no improvement in fitness, then the reset value of the search counter is chosen to be

$$SC_c(N_{kill}) = SC_c^{\max}\left[1 - \frac{1}{N_{kill}+1}\right].$$

2.b.v. Condition for spawning particles and swarms

At each step of the algorithm, each swarm may spawn a new swarm. To be able to spawn a new swarm, an existing swarm must have $N_{kill} = 0$. If this condition is met and the maximum number of swarms will not be exceeded, the swarm spawns a new swarm with probability $p = f / N_s$, where f is a uniform random number on $[0,1]$ and N_s is the number of swarms. The purpose of the factor of $1/N_s$ is to suppress swarm creation when there are large numbers of swarms in existence. When a swarm spawns a new swarm, the spawning swarm (parent) is unaffected. To form the spawned (child) swarm, half of the particles in the child are randomly selected from the parent swarm and the other half are randomly selected from a random member of the swarm collection (mate). The spawned or child swarm may inherit other attributes from either parent or mate as necessary to design experimentation for the Darwinian PSO algorithm. A particle is spawned whenever a swarm achieves a new global best fitness.

3. Experiment to Test Selection as a Method for Circumventing Local Optima

3.a. Adaptation for Selection

Selection allows the environment, or fitness landscape[1], to cause swarms with good adaptations to thrive and swarms with bad adaptations to die. To design a test of selection, a swarm attribute on which to operate must be identified. How explosion of the particle velocity[7] is controlled can have a dramatic impact on the performance of the particle swarm[8]. A simple way to control the explosion of the particle velocity is to limit the velocity using a maximum velocity, $v_{\max}$. If the particle velocity exceeds $v_{\max}$, then the particle velocity is set equal to $v_{\max}$. Another approach to preventing explosion is to modify the dynamical system defined by Eqn. 1, including a constriction factor, χ, such that,

---

[1] Fitness, fitness function, fitness landscape and environment are used interchangeably depending on context.

$$\begin{aligned}\vec{v}_i(t+1) &= \chi\left(\vec{v}_i(t) + \phi_1(\vec{x}_{i,p} - \vec{x}_i) + \phi_2(\vec{x}_{i,n} - \vec{x}_i)\right) \\ \vec{x}_i(t+1) &= \vec{x}_i(t) + \vec{v}_i(t+1)\end{aligned} \quad (2)$$

Without constriction, the particles in the swarm can be made to stay 'hot', in the sense that they are capable of moving large distances in the solution space in a single step, with an appropriately selected $v_{\max}$. This can be used to maximize the exploration aspect built into PSO but may result in sub-optimal performance on fitness functions where gradient following is an effective optimization strategy. In contrast, constriction works well on objective functions for which gradient following is effective. The manner in which explosion control is handled, combined with an objective function, determines how well PSO will work. Our observations are that controlling the particle velocity through $v_{\max}$ will perform well on complicated landscapes and constriction will perform well on "well-behaved" landscapes. Therefore, the type of velocity control that a swarm uses is chosen as the adaptation on which selection is to act. The main reason for choosing particle velocity control as the adaptation on which to operate is that it is easy to show that typically one method works better than the other depending on the fitness function or environment.

3.b. Selection of Test Problems[2]

A set of test problems is selected with the goal of representing fitness landscapes ranging from "well-behaved", in which a global optimum can be found easily using constriction for particle velocity control, to "ill-behaved", in which $v_{\max}$ works better for finding the global optimum. Some of the functions are taken from the De Jong[9] test functions. The set of test problems is presented in Table 2. For the Traveling Salesman test problems, the cities are initialized either randomly or in a circular configuration on an interval [-100,100] in $\mathbb{R}^2$. For Test Functions 1-5, the particle dimension $N = 30$ and each dimension is randomly initialized on the interval [-10,10]. For the Traveling Salesman Problems, the particle dimension is twice the number of cities, and each particle dimension is randomly initialized on the interval [-100,100]. The particle encodes each city as an (x, y) pair where all x values occupy the first $N/2$ dimensions and the y values occupy the last $N/2$ dimensions.

[2] Test Problem and Test Function are used interchangeably.

Table 2 – The selected test functions. The subscript i here indexes the dimensionality of the objective function and should not be confused with the previous use of index i for indexing particles of a swarm. The Traveling Salesman Problem is abbreviated TSP

Function name	Function definition
Test Function 1: De Jong F1: Sphere	$TF1 = \sum_{i=1}^{N} x_i^2$
Test Function 2: De Jong F2 : Rosenbrock[10]	$TF2 = \sum_{i=1}^{N-1} \left(100\left(x_{i+1} - x_i^2 \right)^2 + \left(x_i - 1 \right)^2 \right)$
Test Function 3: Giunta[11]	$TF3 = \sum_{i=1}^{N} \left\{ \begin{array}{l} \sin\left((16/15)\, x_i \left(\mathrm{i} \right) \text{-} 1 \right) + \sin^2\left((16/15) x_i \text{-} 1 \right) \\ + (1/50) \sin\left(40\left((16/15) x_i \text{-} 1 \right) \right) + (3/100) \end{array} \right\}$
Test Function 4: De Jong F4	$TF4 = \sum_{i=1}^{N} x_i^4$
Test Function 5: Rastrigin	$TF5 = \sum_{i=1}^{N} \left\{ x_i^2 - 10\cos\left(2\pi x_i \right) + 10 \right\}$
Test Function 6: TSP	Circular Configuration of 25 Cities
Test Function 7: TSP	Circular Configuration of 50 Cities
Test Function 8: TSP	Circular Configuration of 100 Cities
Test Function 9: TSP	Random Configuration of 25 Cities
Test Function 10: TSP	Random Configuration of 50 Cities
Test Function 11: TSP	Random Configuration of 100 Cities

4. Results

4.a. Evaluating the Best Adaptation for Each Test Problem

Let us define type A velocity control as that using $v_{\max}$ and type B velocity control as that using the constriction factor defined in Eqn. 2. Initially when a swarm is created (not spawned), it is created as either a type A or B swarm. Once it is initialized as a particular type, it remains that type. All particles within the swarm use the same type of velocity control. In the multi-swarm Darwinian algorithm, if a swarm spawns a new swarm, the new swarm inherits the spawning swarm's velocity control type. Here

are the steps used to evaluate the best type of velocity control to use for each selected problem.

1. Execute the Darwinian algorithm 15 times or trials on the test problem. Each of the 15 executions is a full 5000 steps of the algorithm for all test problems except the TSP problems which were allowed 10000 steps.
2. Compute the average number of swarms $< S >$, particles $< P >$ and particles per swarm $< PPS >=< P > / < S >$ resulting from running the Darwinian PSO on the test problems.
3. Execute a single swarm PSO on each test problem where the number of particles in the swarm is set to $< P >$ and the neighborhood size is set to $< PPS >$. The purpose of this configuration is to make the single swarm as much like the Darwinian swarms as possible in terms of the number of particles working on the problem and the average size of the neighborhood of a particle. The single swarm algorithm is also executed 15 times. Each time, the swarm has an equal probability of being initialized as either type A or type B. The result will be that about 7 to 8 single swarm algorithms of each type will be executed on each test problem.
4. Using the results of the single swarm executions, evaluate whether type A or B velocity control is best for each particular test problem.

Table 3 – The single swarm executions allow categorization of the test functions studied into whether Type A or B velocity control yielded the best results. Type A velocity control is $v_{\max}$ velocity control. Type B velocity control uses constriction

Function number	Best Velocity Control Type
1	B
2	B
3	A
4	B
5	undetermined
6	A and B
7	A
8	A
9	A and B
10	A
11	A

Using the steps outlined above, the best type of velocity control for each test problem is evaluated. Table 3 summarizes the results of those experiments. The test functions were selected to represent a set of functions for which the velocity control type would impact the performance of the PSO algorithm. Clearly, the result was achieved with three test functions that respond well to type B velocity control, five functions that respond well to type A velocity control, two functions that respond well to either

type of velocity control and one undetermined. The undetermined result was due to a shortcoming in the current implementation of the Darwinian PSO algorithm and is discussed in Section 5.

Although our results are based upon 15 trial for each test function, we feel that the consistent and compelling outcomes of those 15 trials over 11 test functions, totaling more than 150 trials, is sufficient for supporting the conclusions of this current work. We therefore leave expansion of the number of trials as important future work.

4.b. Evaluating Selection as a Mechanism for Adaptation of the PSO Algorithm

The primary focus of this work is to determine if the selection process can circumvent local optima. To this end, a set of test functions is selected and it is shown that without a prudent choice of velocity control, the PSO algorithm can converge to a local optimum. This is graphically illustrated in Figure 1 where the results for Test Function 1 are shown. Note that a similar set of figures (not shown) results for each test function evaluated. In the figure, the dot-dashed lines are the single swarm results. In each case where the single swarm is initialized with type A velocity control, the algorithm fails to progress. In contrast, the Darwinian algorithm never fails to progress because in all 15 trials, the selection process selects type B swarms and is illustrated in Figure 2.

When a single swarm is initialized, it has an equal probability of being initialized as either type A or type B. Once its type is set, it does not change. This is evident from Figure 2 where the single swarm (dot-dashed line) constriction fraction, defined as the fraction of swarms using constriction (or type B) for velocity control, remains constant, either 0 or 1, over the entire optimization for all 15 trials. Since multiple swarms are initialized for the Darwinian algorithm, the fraction of type B swarms, or constriction factor will be somewhere between 0 and 1. As the selection process operates, the fraction of type B swarms approaches 1. This is precisely the velocity control, of the two contrasting velocity control types considered, which works best on Test Function 1.

The evidence supporting the conclusion that type B swarms work best is derived from comparing Figures 1 and 2, trial by trial. If a single swarm trial (dot-dashed line) in Figure 1 fails to progress, as indicated by a line approximately parallel to the horizontal axis, the corresponding trial of Figure 2, will with 100% coincidence, indicate that the single swarm is using type A velocity control. This supports a conclusion that type A velocity control is bad for Test Problem 1.

The analyses presented in Figures 1 and 2 and discussed above are repeated for all test functions considered. To quantify the ability of the selection process to help the algorithm escape local optima, we define a measure of the probability that the selection process will help the algorithm escape local optima. To compute the measure, the following steps are used.

1. For each trial, the Fitness vs. Objective Function Evaluation curves, like Figure 1, are examined to determine if the single swarm converges to a local optimum. This results in a set of trials $\{set1\} \subset \{All\ trials\}$

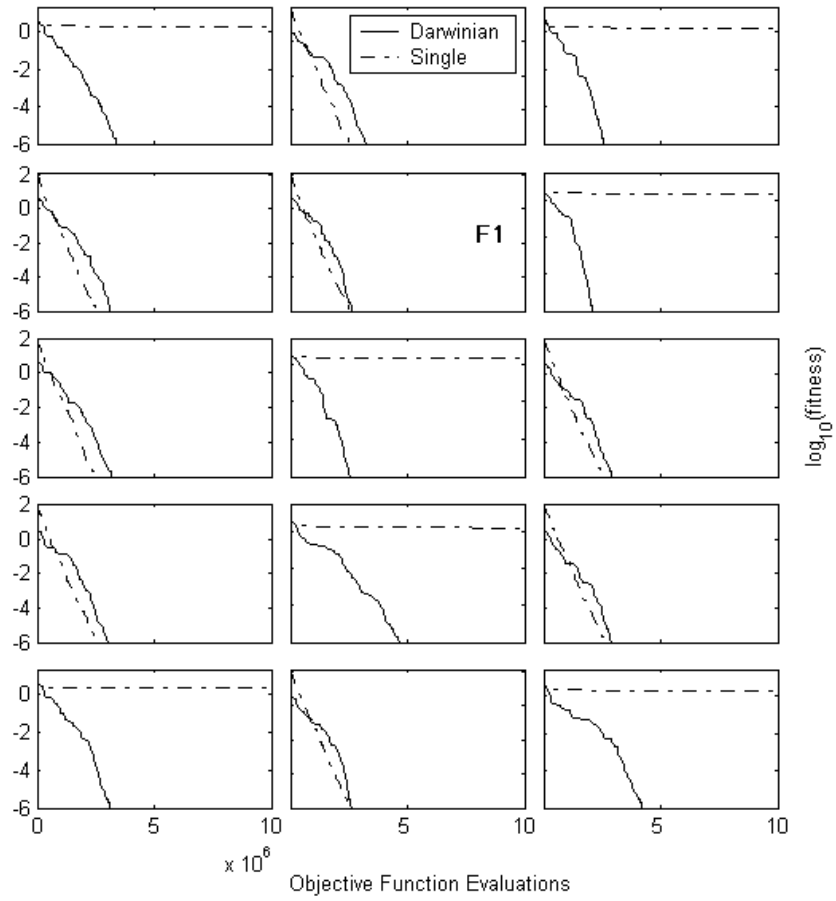

Figure 1. Above is the fitness achieved on the Test Function 1 for 15 trials of both the single swarm algorithm (dot-dashed line) and the Darwinian algorithm (solid line). The fitness (lower is better) is plotted along the vertical axis and the horizontal axis is the number of objective function evaluations

2. For those trials in which the single swarm converges to a local optimum, the trials in which the single swarm used a velocity control other than the best velocity control (refer to Table 3) are selected and result in a new set of trials, $\{set2\} \subset \{set1\}$.
3. Count the number of trials within $set2$ in which the Darwinian selection evolved a set of swarms using the best velocity control type and in which the Fitness vs. Objective Function Evaluation curves shows the Darwinian swarms achieving significantly better fitness than the single swarm.

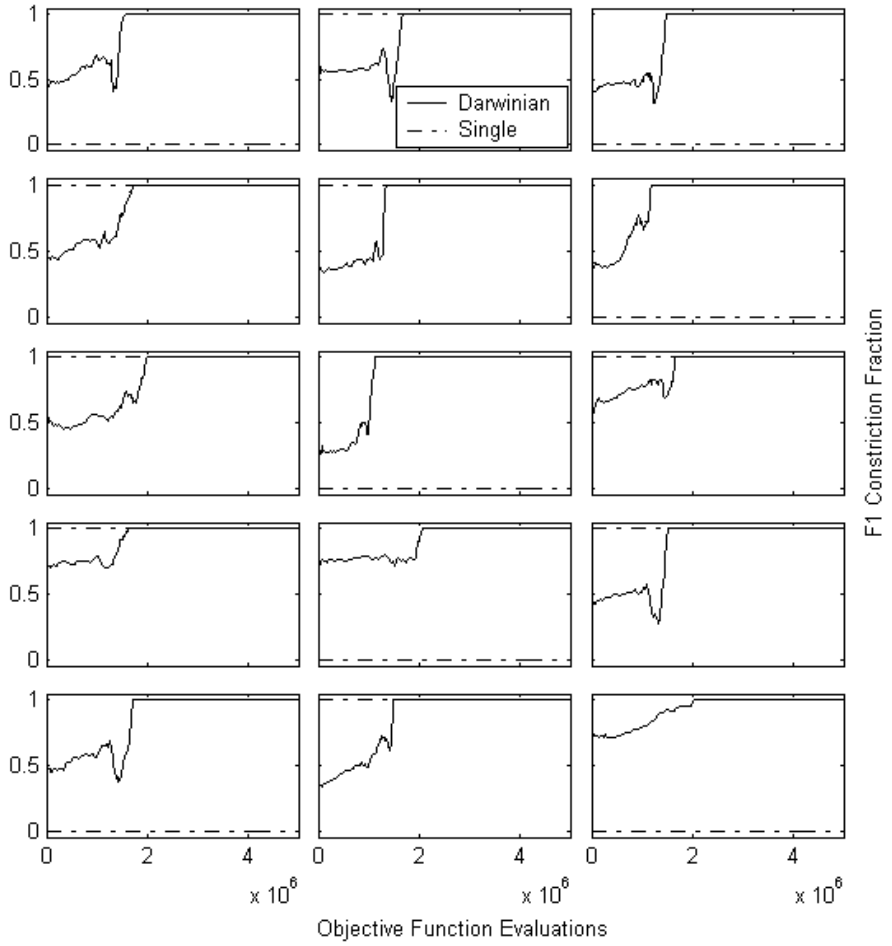

Figure 2. – The solid line is the fraction of the Darwinian swarms using type B (constriction) velocity control. The dot-dashed line shows the single swarm type. Zero represents a type A ($v_{\max}$ velocity control) swarm and 1 represents a type B swarm. The 15 trials displayed are for the F1 test function. The horizontal axis is the number of objective function evaluations

4. The probability measure is computed as the count of step 3 above divided by the total number of elements in $set2$.

The number of elements in $set2$ is 52 for our experiments. The number of those trials in which the Darwinian selection algorithm converged to the preferred type of

velocity control is 47. Therefore the probability that the selection process helps to circumvent local optima is $47/52 = 0.9$.

Main Result: Probability that the Darwinian PSO Circumvents Local Optima = 0.9

5. Discussion

The main result of the paper is that Darwinian PSO helped circumvent local optima in 9/10ths of the selected trials. Since this work is preliminary and the algorithm adapted only velocity control, it is our opinion that the result could be improved by expanding the set of possible adaptations. For example, the results of Test Function 5 in Table 3 are undetermined because the Darwinian PSO algorithm failed to maintain a pool of active swarms, shown in Figure 3. In the upper panel, the algorithm is seen to achieve an on average steady state number of swarms throughout the trial. In the lower panel, the Darwinian swarms simply peak and then die. It is possible that the Darwinian swarms were not given ample time to search the fitness landscape. Since the number of swarms in the population at any given instant is controlled by, in addition to the fitness landscape, the parameter p introduced in Section 2.b.vi., adaptation of the parameter p would be beneficial. Other parameters that are candidates for adaptation are:

- Initial particle count per swarm. We chose 20.
- Max particle count per swarm. We chose 300, which was sufficiently high to allow swarms to evolve their population with essentially no upper bound on the number of particles.
- Max spawn count: This is how many times a swarm is permitted to spawn. We chose 10000 which essentially imposed no limit.
- Maximum number of swarms. We chose 10000 and the maximum number was never approached.
- Condition for deleting a swarm. A swarm is deleted from the collection when its particle population falls below a predefined minimum threshold. We chose 10.
- The method of resetting the search counter SC, discussed in section 2.b.v..
- The social and cognitive components of the particle motion, ϕ_1 and ϕ_2 of Eqn. 1.
- The value of the maximum velocity $v_{\max}$.
- The value of the constriction coefficient χ. We chose 0.78.
- Swarm interaction. We allow swarm interaction when a new swarm is spawned. The parent swarm contributes half of its particle population to the new swarm. A random member of the collection of swarms is selected to contribute half of its particle population. Additional particles are randomly initialized and added to the new swarm until it possesses the initial number of particles allowed. For this paper the child always inherits the velocity control adaptation from the spawning parent.

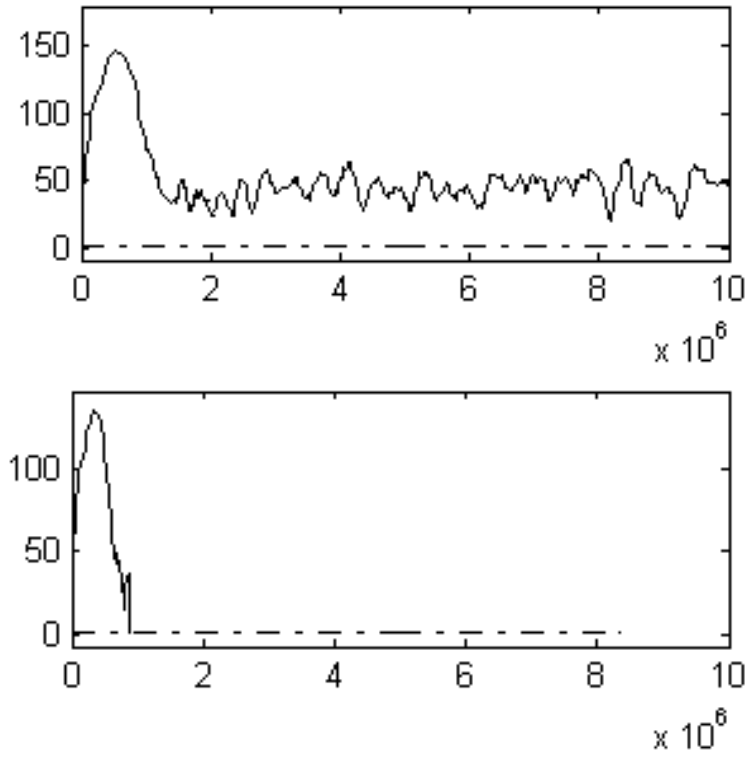

Figure 3. – The number of swarms vs. the number of objective function evaluations. The solid line is the Darwinian algorithm. The dot-dashed line is a single swarm (number of swarms=1). The top frame is a representative trial of Test Function 1 and the bottom panel is a representative trial from Test Function 5

A parameter that cannot be adapted but warrants study is the number of initial swarms. We chose 20 for all trials. For other values that we pre-selected, the values were chosen based on our experience with the PSO algorithm.

The reader may question the lack of typical performance measures of the PSO algorithm[12] and comparisons with other PSO variants. We assert that since our goal is to evolve a collection of swarms so that the resulting collection has an adaptation that is optimized for the problem, a comparison to other PSO variants in terms of convergence rate and solution quality would not be appropriate at this stage of the development of the algorithm. It is notable that comparison of this algorithm with other algorithms would necessarily be restricted to other multi-swarm algorithms because the number of objective function evaluations is dramatically increased by the existence of many swarms at each step of the algorithm. It is this added overhead of evaluating the fitness of poorly adapted swarms that allows the selection process to operate and select an optimum adaptation. The use of a single swarm in this work to determine the best type of velocity control for a specific problem should not be con-

fused with a comparison of the Darwinian PSO algorithm to a single swarm algorithm.

Since the computing demands of this algorithm are higher than a pre-engineered single swarm algorithm, a high-performance computing platform is desirable. The Darwinian PSO algorithm can be parallelized at two levels. The individual swarms' particles could be distributed across a cluster, which is beneficial when the fitness computation is lengthy. Second, the swarms could be distributed as well. The high computational demands of the algorithm motivated our choice for a lower number of trials (15 per test function).

References

1. Eberhart, R.C. and J. Kennedy. *A new optimizer using particle swarm theory*. in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 1995. Nagoya, Japan.
2. Parsopoulos, K., et al., *Improving the Particle Swarm Optimizer by Function Stretching.* Hadjisavvas N and Pardalos PM (eds) Advances in Convex Analysis and Global Optimization, Kluwer Academic Publishers, 2001: p. 445–457.
3. Tawarmalani, M. and N.V. Sahinidis, *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. 2002: Kluwer Academic Publishers, Boston MA.
4. Clerc, M. *The swarm and the queen: towards a deterministic and adaptive particle swarm optimization*. in *Congress on Evolutionary Computation*. 1999. Washington, DC.
5. Angeline, P.J. *Using Selection to Improve Particle Swarm Optimization*. in *IEEE World Congress on Computational Intelligence*. 1998. Anchorage, Alaska, USA.
6. Kennedy, J. and R.C. Eberhart. *A discrete binary version of the particle swarm algorithm*. in *Systems, Man, and Cybernetics*. 1997. Piscataway, NJ: IEEE Service Center.
7. Clerc, M. and J. Kennedy, *The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional and Complex Space.* IEEE Transactions on Evolutionary Computing, 2002. **6**(1).
8. Parsopoulos, K.E. and M.N. Vrahatis, *Recent approaches to global optimization problems through Particle Swarm Optimization.* Natural Computing, 2002. **1**: p. 235-306.
9. De Jong, K.A., *An analysis of the behavior of a class of genetic adaptive systems*. 1975, Univ. of Michigan.
10. Rosenbrock, H.H., *An automatic method for finding the greatest or least value of a function.* The Computer Journal, 1960. **3**(175).
11. Giunta, A.A. and L.T. Watson, *A Comparison of Approximation Modeling Techniques: Polynomial versus Interpolating Models.* AIAA, 1998. **98**(4758).
12. Liang, J.J., et al., *Evaluation of Comprehensive Learning Particle Swarm Optimizer.* Springer's Lecture Notes in Computer Science, ICONIP'04, 2004. **3316**: p. 230-235.