International Journal of Swarm Intelligence and Evolutionary Computation

Mange and Pace, Int J Swarm Intel Evol Comput 2016, 5:2
DOI: 10.4172/2090-4908.1000133

Research Article Open Access

Scheduling Functions for Position Updating in Population Based Optimization Algorithms

Jeremy Mange* and Sara Pace

US Army - TARDEC, Computational Methods and System Behavior, Warren, MI, USA

Abstract

In many population-based optimization algorithms (Evolutionary Algorithms, Particle Swarm Optimization, etc.), each iteration of the algorithm involves a procedure-specific set of operations for each population member, followed by a resulting update of the position of that member within the problem search space. However, for algorithms in which these operations involve only a single population member and not the population as a whole, there is no inherent need to update every member at every iteration. In this paper, we propose a generalization of this updating procedure wherein a "scheduling" function is defined to dictate the ordering of updates through the application of algorithm, thus considering the typical procedure of updating every population member at every iteration as a particular "round-robin" schedule. Using the standard Particle Swarm Optimization algorithm (SPSO-2011) as a basis for demonstrating the concept, we compare a number of different scheduling functions and show that several of these functions outperform the typical round-robin schedule for a set of benchmark optimization problems.

Keywords: Optimization; Particle swarm optimization; Scheduling functions; Population-based methods; Multi armed bandit algorithms

Introduction

Population based optimization algorithms are algorithms in which a set of input values (or candidate solutions, or positions within the problem search space) is maintained through the course of the algorithm, as opposed to non-population-based methods which generally examine only one input value at each iteration of the algorithm. We shall refer to these input values as "individuals", although common terminology varies based on the context of and (often biological) inspiration for specific algorithms. In general terms, a typical procedure for a population-based single-objective optimization algorithm is as follows:

Initialize all individuals within the search space

while Termination condition not met **do**

Choose next individual to examine based on

Scheduling function

Apply algorithm-specific operations

Update the individual's position within the search space

end

Choose the best individual for the final optimized value

Note that in the **for** loop above, each individual is processed in order at every iteration. This is the dominant method of updating individuals' positions for most population-based optimization algorithms. However, when the individuals are processed separately within the algorithm, it is not necessary to follow this update ordering throughout the optimization. In this paper, we introduce a generalization of this concept in which a *scheduling function* is defined in order to choose the next individual to examine at each iteration. Thus, the outline above becomes instead:

Initialize all individuals within the search space

while Termination condition not met **do**

for Each Individual in Population **do**

Apply algorithm-specific operations

Update the individual's position within the search space

end

end

Choose the best individual for the final optimized value

With this generalization defined, the typical procedure outlined first becomes a special case of a "round-robin" schedule function, in which each individual is examined in a clockwork manner. We will also show in this paper that even other simple schedule functions significantly outperform the typical round-robin schedule.

In order to show this, we will employ one of the most widely used simple population-based optimization algorithms, Particle Swarm Optimization (PSO). For the sake of consistency and reproducibility, we will use the Standard Particle Swarm Optimization implementation standardized in 2011, known as SPSO-2011 [1-3]. Using a generalization of this algorithm to include schedule functions, we will compare several such schedule functions on a series of common optimization benchmark problems, and show that some of these functions consistently outperform the standard round-robin schedule used in many population-based algorithms.

Particle swarm optimization

Particle Swarm Optimization (PSO) [4] is a population-based optimization method involving a collection of "particles" iteratively moving within a problem search space according to position updating formulas. These formulas include movement toward the current best-known position by the individual particle, movement toward

***Corresponding author:** Jeremy Mange, TARDEC, Computational Methods and System Behavior, Warren, MI, USA, Tel: 248-742-5111; E-mail: jeremy.mange@gmail.com

Received April 09, 2016; **Accepted** April 18, 2016; **Published** April 22, 2016

Citation: Mange J, Pace S (2016) Scheduling Functions for Position Updating in Population Based Optimization Algorithms. Int J Swarm Intel Evol Comput 5: 133. doi: 10.4172/2090-4908.1000133

the current best-known position by the swarm as a whole, and some random movement. These position update formulas form the heart of the PSO algorithm [5,6]. In basic form for a single particle, they are:

$$v_d = \omega v_d + \phi_p r_p (p_d - x_d) + \phi_g r_g (g_d - x_d) : d = 1...D$$
$$x = x + v$$

where:

D is the number of dimensions in the search space

x_2 is dimension *d* of the particle's position

v_d is dimension *d* of the particle's velocity

r_p and r_s are random numbers in [0, 1]

pd is dimension *d* of the particle's best-known location

g_d is dimension *d* if the swarm's best-known location ω, ϕ_p and ϕ_g are parameters to control behavior of the algorithm

Much research has been done on the selection of PSO parameters (e.g., [7-9]), and many improvements to the algorithm have been proposed since its introduction. Many of these improvements have been incorporated into the successive standard PSO implementations, with the latest being SPSO-2011, the version utilized in this paper. A complete discussion of these improvements and alterations is outside the scope of this paper, but both descriptions of the SPSO-2011 implementation and source code in various languages are available at [3,10,11].

Scheduling functions

Our aim in this paper is not to find the "best" scheduling function for population-based algorithms of the type under consideration, but rather to demonstrate that some other relatively simple scheduling functions outperform the standard round-robin schedule. Since one of the most basic considerations in any optimization task is the tradeoff between exploration and exploitation, a natural choice for potential scheduling algorithms is those based on the widely studied Multi-Armed Bandit problem [12-14], which is a classic exploration/exploitation problem in which a player at a series of slot machines seeks to maximize overall reward. In particular, we will examine some of the best-performing schedule functions from the comparison work of Kuleshov and Precup [15] and others [16,17] – the ε -greedy algorithm, Boltzmann Exploration (Softmax), and Upper Confidence Bounds, in addition to the standard round-robin schedule and a random scheduling function.

All together, we will test the following eight scheduling functions, about which details are given in the following sections: Round-robin Schedule

- Random Schedule
- Fixed τ greedy Schedule
- Adaptive ε greedy Schedule
- Fixed τ Softmax Schedule
- Adaptive τ Softmax Schedule
- UCB1 Schedule (Upper Confidence Bounds)
- UCB1-Tuned Schedule

ε-Greedy schedule

The ε -greedy algorithm is a simple and popular decision algorithm. At each decision point, the algorithm selects the option with the highest current reward with probability $1-\varepsilon$, and a random option with probability ε. That is, given rewards at time t of $r_1(t),...,r_k(t)$, the probability at time t+1 of selecting option *i* is given by:

$$p_i(t+1) = \begin{cases} 1-\varepsilon+\varepsilon/k & \text{if } i=\arg\max_{j=1...K} r_j(t) \\ \varepsilon/k & \text{otherwise} \end{cases}$$

Clearly, the choice of ε significantly affects this algorithm. A full exploration of ideal values for ε in this context is outside the scope of this paper, but based on results in the literature and from our own testing, we will use a fixed value of ε = 0 and a simple linear decay from ε = 1 to ε = 0 for our fixed and adaptive ε -greedy strategies, respectively.

Boltzmann exploration (Softmax)

In general, Softmax algorithms arise from Luce's choice axiom [18], which states that the probability of choosing an item *i* from a pool of *j* items is given by:

$$p(i) = \frac{w_i}{\sum_j w_j}$$

where w is some context-appropriate weighting. Boltzmann Exploration is one such Softmax method that uses a Boltzmann distribution for selection. That is, given rewards at time *t* of $r_1(t),...,r_k(t)$, the probability at time *t+1* of selecting option *i* is given by:

$$p_i(t+1) = \frac{e^{r_i(t)/\tau}}{\sum_{j=1}^{k} e^{r_j(t)/\tau}}$$

Where Γ is a "temperature" parameter controlling the exploration/exploitation tradeoff. Again, a full discussion of ideal values for Γ in this context is outside the scope of this paper, but based on available literature we will use a fixed value of $\Gamma = 0.05$ and a simple linear decay from $\Gamma = 1$ to $\Gamma = 0.05$ for our fixed and adaptive Softmax strategies, respectively.

Upper confidence bounds

Upper Confidence Bounds (UCB) algorithms arose from the Multi-Armed Bandit problem to produce a limit on regret for decision problems of that type [19,20]. These algorithms have proved successful in a number of applications, including, notably, Artificial Intelligence programs for the game of Go.

For these algorithms, one additional variable is needed – the number of times option i has been selected, denoted by n_i. Unlike the previous two schedule functions, the basic UCB algorithm, UCB1, does not assign a probability to each option, but rather first chooses each option once, and from then on, given rewards at tim *t* of $r_1(t),...,r_k(t)$, always chooses option *i* that maximizes:

$$r_i(t) + \sqrt{\frac{2 \ln t}{n_i}}$$

The creators of this algorithm also propose an alternative "tuned" version, UCB1-Tuned [19], which has been found to perform better in practice. This version incorporates the variance of each option ($\hat{\sigma}_i^2$) into the calculation of the next choice. UCB1-Tuned, given rewards at time *t* of $r_1(t),...,r_k(t)$, always chooses option *i* that maximizes:

$$r_i(t) + \sqrt{\frac{\ln t}{n_i} \min\left(\frac{1}{4}, V_i(n_i)\right)}$$

where

$$V_i(t) = \hat{\sigma}_i^2(t) + \sqrt{\frac{2 \ln t}{n_i(t)}}.$$

We will use both UCB1 and UCB1-Tuned as separate scheduling functions in our experiments.

Round-robin and random schedules

The round-robin and random schedules are straightforward. The round-robin schedule always chooses the next option in a clockwork manner throughout the algorithm. The random schedule chooses an option (pseudo-) randomly from a discrete uniform distribution.

Benchmark functions

In order to compare the effects of these different scheduling functions, we will use as the optimization problem a set of common optimization benchmark functions. These minimization functions are some of the most common to appear in optimization literature (e.g., [21-23]), and were chosen to display a variety of function characteristics. Those functions are shown in Table 1, where D is the number of search space dimensions (Table 2).

For all of these functions, the global minimum is $f(\vec{0})$. Each of these functions is defined for an arbitrary number of dimensions, as discussed in the following section.

Experimental design

In this section, we detail the design of our experiments to compare the scheduling functions on the benchmark problems defined above. Of particular consideration are the parameters to be used for the Particle Swarm Optimization algorithm, the number of dimensions used for the benchmark functions, and the methods of comparing the schedule effects.

PSO parameters

In order to isolate the effect of the scheduling functions, as little as possible was changed from the SPSO-2011 implementation. Thus the PSO parameters were left at their default values, which included a swarm size of 40, and parameter values of:

$$\omega = \frac{1}{2\ \log(2)}$$

$$c_1 = c_2 = \frac{1}{2} + \log(2)$$

See [1] and [2] for discussion and definition of parameters in this context.

All other details of the SPSO-2011 implementation will be left unchanged aside from the introduction of the schedule functions defined above, and one other small detail – the initial particle positions will be randomly calculated once, then those same starting positions will be used for each schedule and each independent run. This again is to isolate the effect of the schedules by eliminating random starting biases.

Name	Function	Range
Sphere	$f(x) = \sum_{i=1}^{D} x_i^2$	$-100 \le x_i \le 100$
Rastrigin (generalized)	$f(x) = \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$	$-2\pi \le x_i \le 2\pi$
Rosenbrock	$f(x) = \sum_{i=1}^{D-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$	$-2 \le x_i \le 2$
Griewank (generalized)	$f(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \le x_i \le 600$
Schwefel (normalized)	$f(x) = \frac{\sum_{i=1}^{D} -x_i \sin\left(\sqrt{\text{abs}(x_i)}\right)}{D}$	$-512 \le x_i \le 512$
Salomon	$f(x) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2}$	$-100 \le x_i \le 100$

Table 1: Benchmark functions.

Schedule	Average Function	Performance relative
	Value	to Round-Robin
Round-robin	1.000	100 %
Random	0.846	118 %
Fixed ε -greedy	0.272	368 %
Adaptive ε -greedy	**0.199**	**502 %**
Fixed τ Softmax	1.085	92 %
Adaptive τ Softmax	0.883	113 %
UCB1	0.840	119 %
UCB1-Tuned	0.833	120 %

Table 2: Averaged results - 2 dimensions.

Benchmark function dimensions

Each of the benchmark functions we will use is defined for an arbitrary number of dimensions, and it is common in benchmarking optimization algorithms to run an algorithm over a variety of numbers of dimensions in order to see performance characteristics at different levels. We will also adopt this approach and show results for each function at 2, 10, and 50 dimensions.

Other considerations

In the Results section, we will use two different methods of comparing the different scheduling functions. First, we will compare the best-known function values after a specified number of objective function evaluations, as a way of showing a snapshot of the performance of each schedule. Specifically, we will compare the results after $50d + N$ function evaluations, where d is the number of dimensions of the problem and N is the number of PSO particles, in this case 40 (to account for the evaluations of the initial placements of the particles). In addition, we will show a trace of the best-known function values after each objective function evaluation, as a way of showing the performance of each schedule over time.

Since SPSO-2011 is a stochastic algorithm, for both of these methods we will show the best-known function values averaged over 500 independent runs.

Finally, since several of the scheduling functions defined above require an estimate of current reward, we will use each particle's current best-known objective function value as that estimate within the schedule. We also conducted experiments using the particle's current function value as the reward estimate, but the results did not differ significantly.

Results

In presenting our results, we begin with a summary of the overall performance of the different scheduling algorithms, and then break down these results to show more specific details of the schedules, the benchmark functions, and the number of dimensions involved in the experiments.

These results demonstrate the main thesis of the paper – that significant performance gains can be achieved in population-based optimization algorithms through the use of alternative scheduling functions for updating. In particular, the results indicate that massive performance gains are possible at low dimensions, and more modest

gains at higher dimensions with the relatively simple scheduling functions tested. With more advanced functions or hybrid approaches, it is very likely that scheduling functions can be defined that provide consistent advantages across a broad spectrum of problems and numbers of dimensions.

Overall summary

Overall, the adaptive -greedy schedule performed the best by far at low dimensions, outperforming the standard round-robin schedule by over five-fold on average on 2-dimensional benchmark problems. On the 10-dimensional benchmark problems, again the adaptive ε-greedy schedule performed the best, narrowly outperforming the round-robin schedule, which itself outperformed all others. On the 50-dimensional benchmark problems, most of the scheduling functions performed similarly, with the adaptive Softmax schedule performing the best, and the fixed ε -greedy schedule as a low-performing outlier.

Average comparisons for 2 dimensions, 10 dimensions, and 50 dimensions are shown in three Tables 3 and 4, with the columns normalized by the round-robin schedule results (Table 3).

Results in 2 dimensions

Both of the ε-greedy scheduling functions were by far the best performers for the 2-dimensional benchmark functions. A bar graph of the average function values for each schedule after 100 evaluations is shown in Figure 1. Each function value is normalized by the round-robin schedule average value, in order to show the relative performance of each alternative scheduling function.

As a more complete view of the best-known function value over time for each schedule, Figure 2 shows a trace of these values for each number of function evaluations on each benchmark function. Note that the PSO swarm for each scheduling function was set to the same randomized starting location, so the initial best-known values for each are identical.

Schedule	Average Function	Performance relative
	Value	to Round-Robin
Round-robin	1.000	100 %
Random	1.539	65 %
Fixed ε -greedy	4.727	21 %
Adaptive ε -greedy	**0.958**	**104 %**
Fixed τ Softmax	1.698	59 %
Adaptive τ Softmax	1.775	56 %
UCB1	1.690	59 %
UCB1-Tuned	1.706	59 %

Table 3: Averaged results - 10 dimensions.

Schedule	Average Function	Performance relative
	Value	to Round-Robin
Round-robin	1.000	100 %
Random	0.856	117 %
Fixed ε -greedy	17.270	6 %
Adaptive -greedy	1.916	52 %
Fixed τ Softmax	0.969	103 %
Adaptive τ Softmax	**0.857**	**117 %**
UCB1	0.958	104 %
UCB1-Tuned	0.961	104 %

Table 4: Averaged results - 50 dimensions.

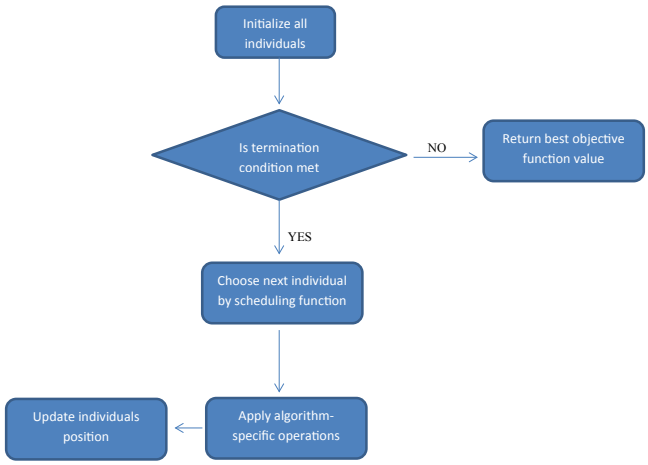

Figure 1: Modified general optimization algorithm.

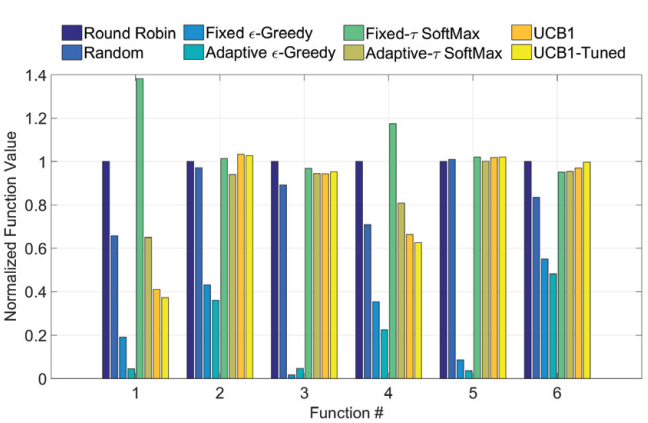

Figure 2: Final function value comparison for 2 dimensions.

This figure in particular illustrates how well the ε-greedy schedules performed. The figure also illustrates how closely aligned the behavior of the UCB1 and UCB1-Tuned algorithms are over these benchmark functions.

Results in 10 dimensions

For the 10-dimensional benchmark functions, again the adaptive ε-greedy schedule was the best performer, although in this case the standard round-robin schedule was also one of the best. A bar graph of the average function values for each schedule after 500 evaluations is shown in Figure 3. For the sake of clarity in this figure, high values are clipped for the fixed ε -greedy schedule on the Sphere, Rosenbrock's function, and Griewank's function. The averaged values for this schedule are listed in Table 4.

Figure 4 shows a trace of the best-known function values for each number of function evaluations on each benchmark function. This figure illustrates that the fixed ε-greedy schedule's performance begins to fall off as the number of dimensions in the optimization function increases. This trend continues to the 50-dimensional case, where the fixed ε -greedy scheduling function shows by far the worst performance.

Results in 50 dimensions

For the 50-dimensional benchmark functions, the adaptive- τ Softmax schedule performed the best, although the performance of

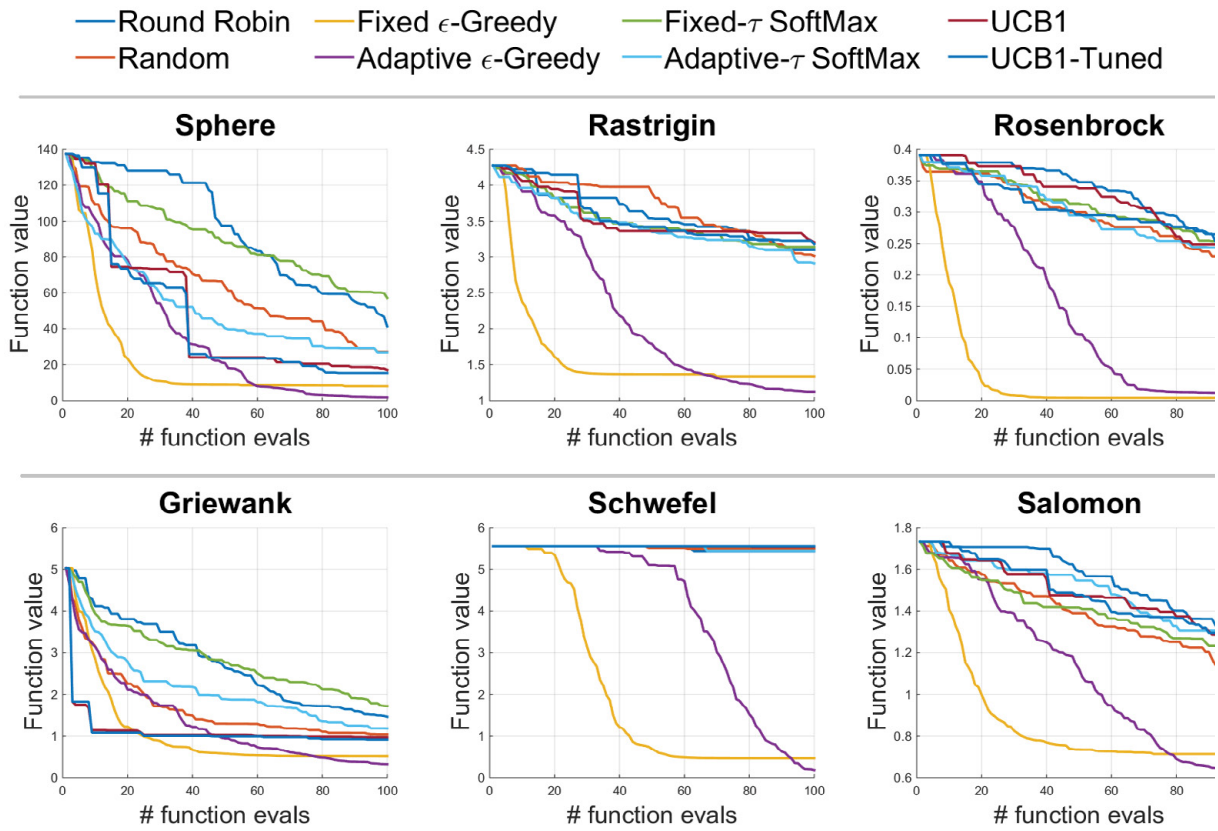

Figure 3: Best-known function values trace for 2 dimensions.

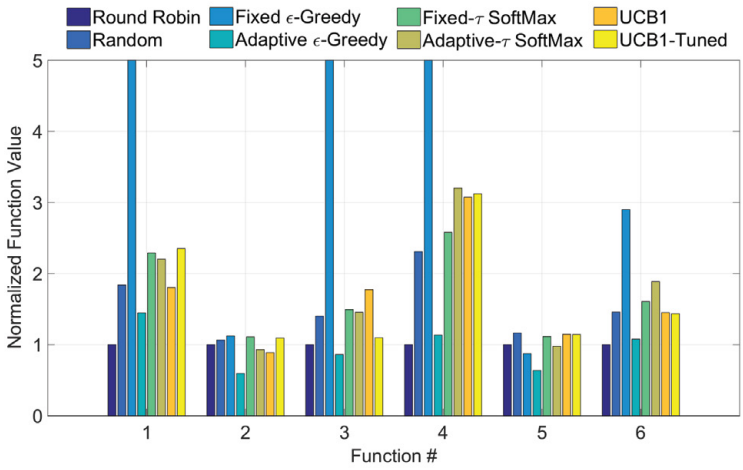

Figure 4: Final function value comparison for 10 dimensions.

many of the scheduling functions was similar. Although the adaptive ε-greedy algorithm was the top performer in the 2-dimensional and 10-dimensional cases, and still outperformed all other schedules on two of the 50-dimensional benchmark functions (Rastrigin's function and Schwefel's function), overall it did not perform as well as the Softmax or UCB schedules for the 50-dimensional case.

A bar graph of the average function values for each schedule after 2500 evaluations is shown in Figure 5. Again, for the sake of clarity in this figure, high values are clipped for the fixed ε-greedy schedule on the Sphere, Rosenbrock's function, and Griewank's function. The averaged values for this schedule are listed in Table 1.

Figures 6 and 7 shows a trace of the best-known function values for each number of function evaluations on each benchmark function.

This figure illustrates how poorly the fixed ε-greedy scheduling

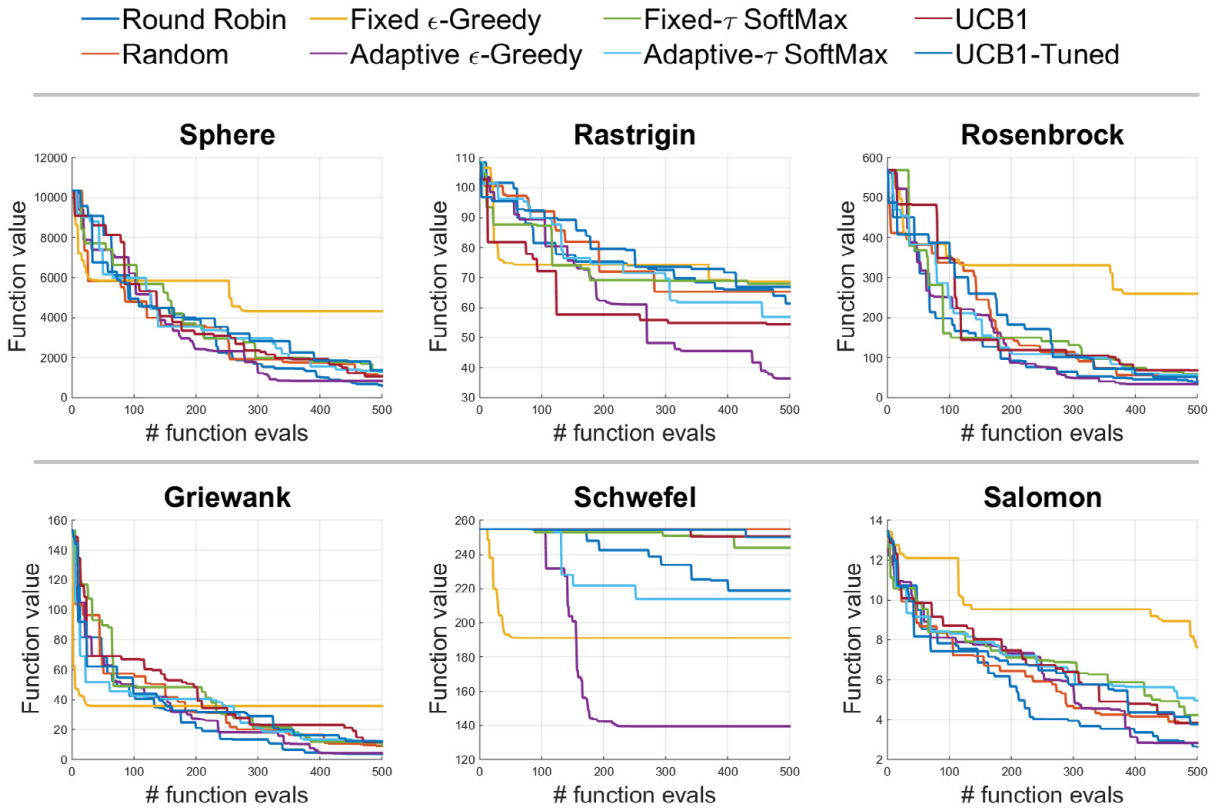

Figure 5: Best known function values trace for 10 dimensions.

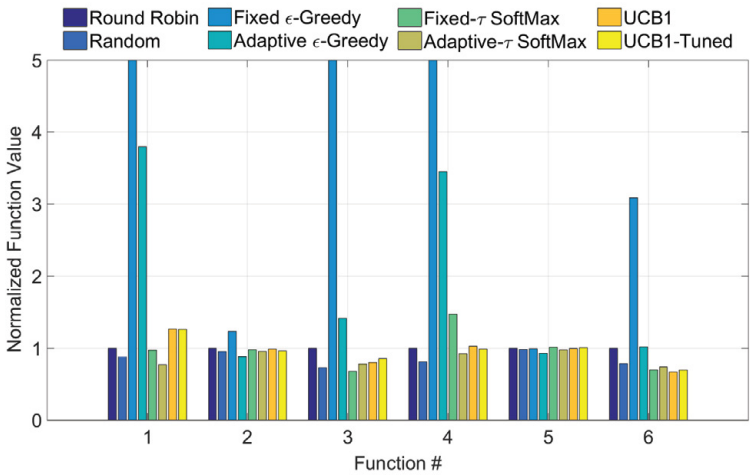

Figure 6: Final function value comparison for 50 dimensions.

function performs in high dimensions. This is understandable, since this schedule tends to be highly exploitative, whereas for high-dimensional optimization problems, a good deal of exploration is needed before exploitation yields fruitful results (Table 4).

Conclusions

Within population-based optimization algorithms, the updating of the search-space position of each individual at each iteration of the algorithm is generally assumed. However, this is often not inherently necessary. We have proposed a generalized approach in which scheduling functions are used to define the order of these position updates.

The main conclusion of this work is that there is great potential for improvement of the performance of population-based optimization algorithms through defining these scheduling functions. Even with the relatively simple scheduling functions examined in this paper, significant performance increases were shown in several contexts. Perhaps the most striking example of this was a performance increase

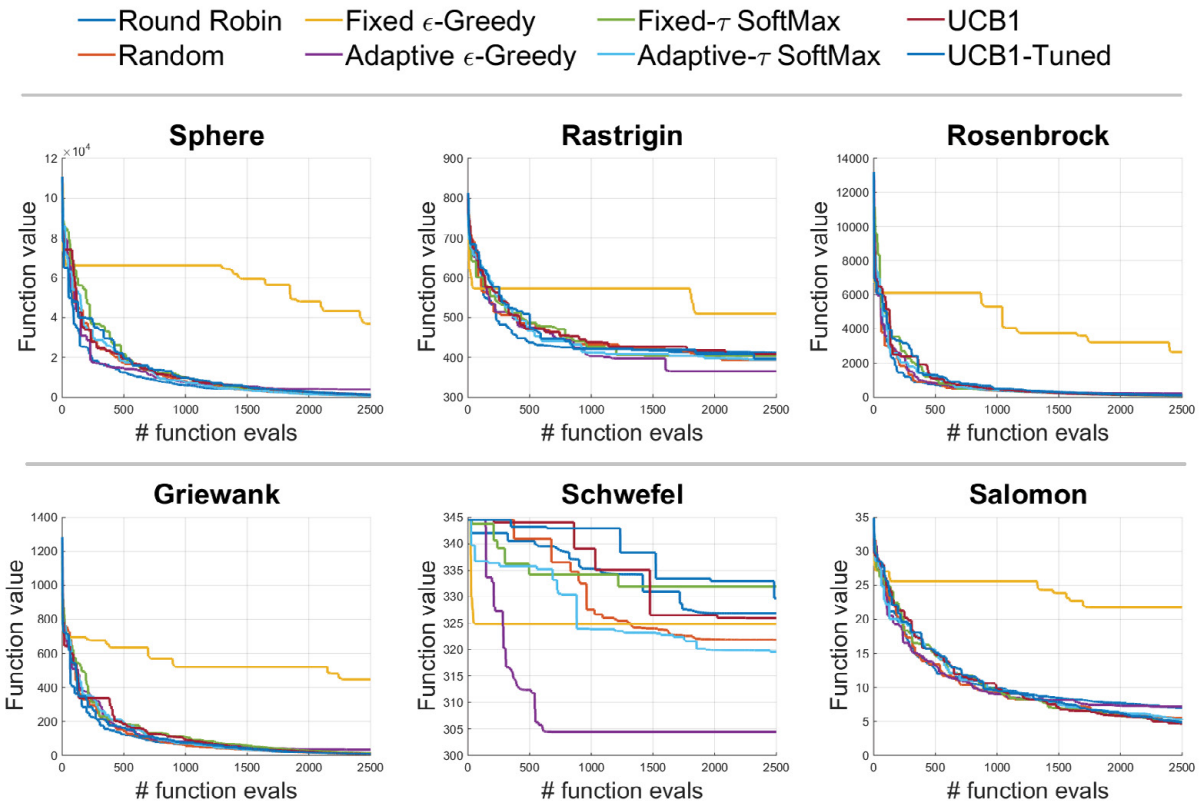

Figure 7: Best known function values trace for 50 dimensions.

of over 500% for the 2-dimensional set of benchmark problems, merely by using a simple adaptive ε-greedy schedule instead of the standard round-robin schedule within the Standard Particle Swarm Optimization (SPSO-2011) algorithm [24-31].

We highlight the relative simplicity of these scheduling functions to emphasize the need for further research in this area, particularly to define schedules that provide the most advantage for the types of problems relevant to the contexts of various practitioners. There is much potential for improvement, both for general-purpose population-based algorithms through the use of more advanced scheduling function or hybrid approaches, and for more specific applications through the same type of comparison and analysis shown in this paper, but focused on the particular attributes of the problems and data sets under consideration within those more specific contexts.

References

1. Daniel B, Kennedy J (2007) Defining a standard for particle swarm optimization. Swarm Intelligence Symposium 2007.
2. Maurice C (2012) Standard particle swarm optimisation.
3. PSC (2016) Particle Swarm Central.
4. Kennedy J, Eberhart RC (1995) Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks p: 4.
5. Kennedy J (2011) Particle swarm optimization. Encyclopaedia of machine learning. Springer US pp: 760-766.
6. Riccardo P, Kennedy J, Blackwell T (2007) Particle swarm optimization." Swarm intelligence 1.1: 33-57.
7. Shi Y, Eberhart RC (1998) Parameter selection in particle swarm optimization". Proceedings of Evolutionary Programming VII (EP98) p: 591600.
8. Eberhart RC, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimization. Proceedings of the Congress on Evolutionary Computation p: 8488.
9. Taherkhani M, Safabakhsh R (2016) A novel stability-based adaptive inertia weight for particle swarm optimization. Applied Soft Computing 38: 281295.
10. Cristian TL (2003) The particle swarm optimization algorithm: convergence analysis and parameter selection. Information processing letters 85: 317-325.
11. Russell CE, Shi Y (2001) Particle swarm optimization: developments, applications and resources. Evolutionary Computation, Proceedings of the 2001 Congress on IEEE.
12. Peter A (1995) "Gambling in a rigged casino: The adversarial multi-armed bandit problem." Foundations of Computer Science. Proceedings 36th Annual Symposium on IEEE.
13. Jean-Yves A, Munos A, Szepesvari C (2009) "Exploration-exploitation trade off using variance estimates in multi- armed bandits. Theoretical Computer Science 410: 1876-1902.
14. Joannes V, Mohri M (2005) Multi-armed bandit algorithms and empirical evaluation. Machine learning: ECML 2005. Springer Berlin Heidelberg pp: 437-448.
15. Volodymyr K, Precup D (2014) Algorithms for multi-armed bandit problems. arXiv preprint ar 15: 6028.
16. Laurent P, Garcia F (2004) On-line search for solvingMarkov decision processes via heuristic sampling. learning 16: 2.
17. Levente K, Szepesva´ri C (2006) "Bandit based monte-carlo planning." Machine Learning: ECML. Springer Berlin Heidelberg pp: 282-293.
18. Ducan LR (1959) Individual Choice Behavior: a Theoretical Analysis. John Wiley and Sons.
19. Peter A (2000) Using upper confidence bounds for online learning. Foundations of Computer Science Proceedings. 41st Annual Symposium on IEEE.
20. Peter A (2003) Using confidence bounds for exploitation-exploration trade-offs. The Journal of Machine Learning Research 3: 397-422.

21. Yuhui S, Eberhart RC (1999) Empirical study of particle swarm optimization. Evolutionary Computation. Proceedings of the 1999 Congress on IEEE.

22. Ke T (2007) Benchmark functions for the CEC2008 special session and competition on large scale global optimization. Nature Inspired Computation and Applications Laboratory, USTC, China pp: 153-177.

23. Jakob V, Thomsen R (2004) A comparative study of differ- ential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. Evolutionary Computation, 2004. CEC2004 Congress on IEEE.

24. Sukumar S (2014) Practical Applications of Swarm Intelligence and Evolutionary Computation, Hybrid soft computing. International Journal of Swarm Intelligence and Evolutionary Computation.

25. Li M, Forouraghi B (2014) Multi-objective Particle Swarm Optimization with Gradient Descent Search. International Journal of Swarm Intelligence and Evolutionary Computation.

26. Renbin X (2015) The Significance of Problem-Oriented Approach to Swarm Intelligence. International Journal of Swarm Intelligence and Evolutionary Computation.

27. He S (2004) An improved particle swarm optimization for optimal power flow. Power System Technology. Power Con International Conference on IEEE.

28. Pandian VM (2013) Meta-Heuristics Optimization Algorithms in Engineering, Business, Economics, and Finance. IGI Global pp: 1-734.

29. Pandian VM (2015) Handbook of Research on Artificial Intelligence Techniques and Algorithms. IGI Global pp: 1-796.

30. Pandian VM (2014) Handbook of Research on Novel Soft Computing Intelligent Algorithms: Theory and Practical Applications. IGI Global pp: 1-1018.

31. Bevrani H (2012) Intelligent frequency control in an AC microgrid: online PSO-based fuzzy tuning approach. Smart Grid, IEEE Transactions on 4: 1935-1944.

Citation: Mange J, Pace S (2016) Scheduling Functions for Position Updating in Population Based Optimization Algorithms. Int J Swarm Intel Evol Comput 5: 133. doi: 10.4172/2090-4908.1000133