Self–Adaptive Bacteria Swarm for Optimization

Mario A. Muñoz, Jesús A. López, Eduardo F. Caicedo
Grupo de Investigación Percepción y Sistemas Inteligentes
Universidad del Valle, Cali, Colombia, South America
e–mail: {andremun,jesuslop,ecaicedo}@univalle.edu.co

Abstract

This paper presents a self–adaptive bacteria swarm optimization algorithm, and its application in a suite of optimization benchmark problems, where the self–adaptive algorithm outperformed in most cases the non adaptive version. The algorithm follows a methodology that uses some concepts included in the Evolution Strategies for the parameter control, allowing the algorithm to select online the best parameter set.

1 Introduction

In nature, it is common to see animals forming groups to perform survival tasks, e.g. we can see ants looking for food, fish forming schools to face predators, birds fly in particular formations to optimize energy consumption, and so on. Social behavior is key for the success of several species and, because of that, many scientists have studied in depth its characteristics and have created computational and mathematical models [13]. From these studies has arised a set of techniques in the field of computational intelligence called Swarm Intelligence (SI), based on the collective behavior of self–organized and decentralized systems used to solve practical optimization problems [25].

For an algorithm to be classified as SI, it must use a population of simple computational agents capable of sensing and change their environment locally. This characteristic makes the communication between agents possible through the detection of the changes in the environment made by their peers [24]. Although there is no central control structure that determines the agent behavior, the local interactions between agents allow the appearance of a global intelligent behavior. There are several techniques that belong to SI. The more representative are Ant Colony Optimization (ACO) [4–6] and Particle Swarm Optimization (PSO) [7, 10, 11]. Other algorithms that can be classified as SI are the Bacteria Swarm Foraging Optimization (BSFO) [13, 16, 17] and several algorithms based on beehive behavior [2, 9, 15, 18, 21].

Those techniques have been successfully applied to a variety of problems; however, they have, as main drawback, the need to specify several parameters that control the behavior of the agents. Select an appropriated parameter set is a difficult task because of the interdependence of the parameters and its relation to the problem specification. For this reason, the algorithm tuning process requires time–consuming tests.

There has been some attempts to obtain a general parameter set that behave correctly in several optimization problems, but most of the published works suggest values based in "rules of thumb". Today it is recognized that each problem requires a specific parameter set and that search for a general optimal parameter set is a lost battle [8], because it does not consider the dynamic nature of the search process. It has been accepted that the parameters must change on–line taking as measure the general performance of the algorithm. This mechanism called **parameter control**, modifies the search parameters by deterministic, adaptive or self–adaptive methods. One algorithm that uses a self–adaptive method to control its search parameters is the Evolution Strategies (ES), which uses mutation, recombination, and selection in the strategy and objective parameters.

This paper presents a Self–Adaptive SI technique based on the Bacteria Swarm Foraging Optimization (BSFO). Our goal is to allow the on–line search of a good parameter set for this continuous optimization technique. The resulting algorithm, the Self–Adaptive Bacteria Swarm Foraging Optimization, not only searches for a good solution of the problem but also modifies its parameters according to the general performance of the algorithm. The paper is organized as follows: First, in section 2, we present some general concepts of BSFO. Next, in section 3, we describe the self–adaptive bacteria swarm algorith. Then, in section 4, we continue with the performance evaluation of the algorithm presented by using a suite of benchmark functions found in the literature and the so–called t–test. Finally, in section 5 we present some conclusions and recommendations for further development.

978-0-7695-3320-9/08 $25.00 © 2008 IEEE
DOI 10.1109/CERMA.2008.97

2 Bacteria Swarm Foraging Optimization

BSFO was proposed by Passino in 2000 [16], inspired by the chemotactic behavior of the E. Coli. Chemotaxis is the phenomenon in which bodily cells, bacteria, and other single-cell or multicellular organisms direct their movements according to certain chemicals substances in their environment. Bacteria go toward places with large concentrations of nutrients and they move away from places with low concentrations of nutrients or places with harmful substances. In BSFO, the quantity of nutrients is related to the value of the performance function. A big value of the performance function indicates that the bacterium is in a place with plenty of food (nutrients). The artificial bacteria try to move in a direction to increase the concentration of nutrients. A bacterium is influenced by the behavior of other bacteria. In BSFO, this element is imitated by using a repellent-attracting function. If the effect of the attraction if strong, the bacteria try to form groups or swarms. BSFO has been used to tune a proportional–integral–derivative (PID) controllers [12] and in the temperature regulation in a temperature grid [1].

3 Self–Adaptive Bacteria Swarm Foraging Optimization

When we use an algorithm to solve an optimization problem, the selection of parameters of the algorithm is equivalent to place a point in the M–dimension parameter space, where M is the number of parameters, such that it achieves the best possible general performance value. The selection of these parameters is, in fact, another optimization problem which can be as or more complex than the main problem, since we do not know the relationship between the general performance of the algorithm, the values of the parameters, and the objective function. Because of that, the common approach is to choose these parameters by trial and error which has, among others, the following limitations [8]:

- The parameters are not independent
- The tuning process requires time–consuming tests
- The parameter set is related to problem specification

Today, change on–line the parameters of the optimization algorithm is widely accepted idea. This mechanism called **parameter control**, modifies the search parameters by deterministic, adaptive or self–adaptive methods. Self–adaptive refers to the use of evolutionary methods for the parameter control by means of a general performance function, G, or secondary optimization problem, directly related to the cost function, J, or main optimization problem. To define a general method for swarm self–adaptation, we select some basic concepts of the ES to solve the problem of finding a good set of parameters.

First, we defined μ populations of κ swarm agents. Each population has a strategy parameter set that is mutated using a normal distribution to generate λ new populations; each one seeks a solution of the main optimization problem. We define the general performance of the l population as the average of cost value of the k agent at the i–th iteration, $J(l,k,i,\theta)$, according to Equation 1. After the general performance has been calculated, we select the best μ populations, who become the parents of the next generation.

$$G(l,i) = \frac{\sum_{k=1}^{\kappa} J(l,k,i,\theta)}{\kappa} \tag{1}$$

Next we use the $1/5$th rule to modify the values of the mutation strength [19]. This deterministic rule states that if performance of the new populations is better than the last populations, then we have found a better position in the parameters' space of the algorithm and we can decrease the mutation strength by using a factor of 0.82, reducing the subspace where the algorithm search in the parameters' space. Otherwise, we increased the mutation strength by using a factor of 1.22, increasing the subspace.

These simple rules can generate different architectures, depending of the values of κ, μ, λ, ρ, and the use of elitist or non elitist selection. To simplify the description of a self–adaptive swarm (SAS), we propose the notation $(\kappa \rightarrow \mu/\rho, +\lambda)$–SAS based in the ES notation. With this notation, we can describe any SAS algorithm, including the presented by Miranda and Fonseca [14], which can be codified as $(1 \rightarrow \mu/\rho, \lambda)$–SAS.

Based on this notation, we define a $(\kappa \rightarrow 1, \lambda)$–SAS, which means κ swarm agents, one parent population, λ offspring populations, with elitism and no recombination, which is the simplest architecture possible using multi–member populations.

For the implementation of a self–adaptive algorithm based on BSFO, it was necessary to make some adjustments over the original algorithm. BSFO has some parameters that control the workings of additional heuristics besides the chemotaxis search; those are the reproduction, the elimination/dispersion, and the communication by attractants and repellents. These heuristics were added to improve the performance of the algorithm, but its design based in FOR cycles, makes them time consuming and inefficient, especially in vector oriented languages like MATLAB.

To determine the influence of the algorithm parameters in the search, we performed several preliminary tests, by which we concluded that the main parameter is the step size, C. Other parameters that have influence in the chemotactic behavior are the number of chemotactic steps, N_c, the magnitude of the attractants and repellents, m, and width of the

```
Select the initial strategy parameters
Mutate the strategy parameters to obtain λ sets
Initialize the position θ of the population randomly
Calculate the initial cost of the population
FOR i = 1 TO epochs DO
   FORl = 1 TO λ DO
     Calculate J_cc
     IF J+J_cc ≥ J_last+J_cc THEN
       Generate a new ψ_k for the bacteria
     END IF
     Calculate the value of Δθ with the parameter set l
     Calculate the population's cost J(l,κ,i,θ)
     Calculate the population's general performance value G(l,i)
   END IF
   Select the population with the best G
   Update the variables of the population
   IF i > 1 THEN
     IF ΣG_ant > ΣG THEN
       Decrease the mutation strength by 0.82
     ELSE
       Increase the mutation strength by 1.22
     END IF
   END IF
   Mutate the strategy parameters to obtain λ sets
   ΣG_ant = ΣG
END FOR
```

Table 1. Self–Adaptive Bacteria Swarm Foraging Optimization

attractants, a, and repellents, r. Because the reproduction and elimination/dispersion do not have a great influence in the search mechanism, we decide to simplify the algorithm by using only chemotaxis search and the communication mechanism.

Next, for the bacteria algorithm in the i iteration for the k individual with the group of parameters, we define a new possition for the agent $\theta_k(i+1)$ value by the equation 2, where $\psi_k(i)$ represents the direction followed by the bacterium. The cost value is composed by the nutrients values and the attractants/repeallants values as shown by the equation 3. The equation 4 defines the attractants/repeallants values, with $r > a$ because their size is inversely proportional.

$$\theta_k(i+1) = \theta_k(i) + C_l(i)\,\psi_k(i) \tag{2}$$

$$J(\theta_k, i) = J_n(\theta_k) + J_{ar}(\theta_k, i) \tag{3}$$

$$J_{ar}(\theta_k, i) = m_l(i)\left(e^{-r_l(i)\sum(\theta_k-\theta_m)^2} - e^{-a_l(i)\sum(\theta_k-\theta_m)^2}\right) \tag{4}$$

The self–adaptive algorithm is constructed with Equations 1, 2, 4, and 3. The result is the algorithm shown in Table 1.

The algorithm follows this procedure. After initiate and generate set of parameters, the population is distributed in random locations over the complete search space, and the cost of each position is acquired. At this point, the main loop begins. The next step is to calculate the new positions, cost, and general performance value, with each group of parameters. Then, the best population is selected. After, the algorithm determines if the performance of all the population has improved, allowing the reduction of the mutation strength, otherwise the mutation strength is increased. Finally, the memory for the general performance is updated, and the main loop ends. The algorithm as proposed is designed to minimize a function. To maximize, the cost function can be expressed as $J(\theta_k, i) = -J_n(\theta_k) - J_{ar}(\theta_k, i)$.

It is important at this stage to determine the performance of the algorithm against other techniques, specially the BSFO algorithm. The next section shows the results of this study.

4 Function Optimization through Self–Adaptive Bacteria Swarm

We conducted a simulation study by using a suite of ten benchmark functions and five SI algorithms for continuous optimization to test the algorithm proposed: Bacteria Swarm Foraging Optimization (BSFO) [13, 16, 17], Particle Swarm Optimization (PSO) [11], the Inertia Weight Particle Swarm Optimization (IWPSO) [20], the PSO with Trelea's first parameter set (T1PSO), and the PSO with Trelea's second parameter set (T2PSO) [22], the Constriction Factor Particle Swarm Optimization (CFPSO) [3], and the Artificial Bee Hive Algorithm (ABHA) [15]. The parameters used for each algorithm are for BSFO $p_{ed} = 0.10$, $p_{re} = 0.20$, $C_i = 0.05d$, $N_s = 10$, for PSO $mv = 0.01d$, $c_1 = c_2 = 2.000$, for IWPSO $mv = 0.01d$, $w = 0.600$, $c_1 = c_2 = 2.000$, for T1PSO $mv = 0.01d$, $w = 0.600$, $c_1 = c_2 = 1.700$, for T2PSO $mv = 0.01d$, $w = 0.729$, $c_1 = c_2 = 1.494$, for CFPSO $mv = 0.01d$, $\chi = 0.7298$, $c_1 = 2.8000$, $c_2 = 1.3000$, for ABHA $p_{rs} = 0.5$, $p_e = 0.8$, $\sigma = 0.2$, $\rho = C = 0.1$, $SR = 0.95$, and for SABSFO $\kappa = 50$, $\lambda = 5$, $C_i = 0.05d$, $N_s = 10$; where d is the diagonal of the search space. The test was carried out for 100 iterations of 1000 epochs. The number of agents for all algorithms in each of the iterations was constant and equal to 50 agents. All the benchmark functions used are non-linear and they are commonly used in optimization algorithm benchmarks. These are the SCB [26], PFUNC [17], MATLAB Peaks, DeJong F1, Grienwank, Rastrigin, Ackley, DeJong F2, Schaffer F6, and Schewefel functions [27]. The Table 2 shows the equations of the benchmark functions and the Table 3 shows the number of dimensions, range, an minimum value.

For a one–on–one performance comparison, we used the t–test with a 95% confidence range [23]. The t–test allows the verification of the statistical validity of the result and at the same time if the algorithm under test can be considered statistically better than the control algorithm. The t–test, considered to be a signal to noise ratio, calculates a difference between the two groups and it is calculated by the equation 5, where T is the test group and C the control group, n is the number of samples and t the probability in a t–student distribution. The t value will be positive if the first mean is higher than the second and negative if its

Name	Equation
SCB	$J = \left(4 - 2.1\theta_1^2 + \frac{1}{3}\theta_1^4\right)\theta_1^2 + \theta_1\theta_2 + \left(-4 + 4\theta_2^2\right)\theta_2^2$
PFUNC	$J = \sum_{i=1}^{10} A_i e^{-\alpha_i\left((\theta_1 - X_i)^2 + (\theta_2 - Y_i)^2\right)}$ $A = [5\ -2\ 3\ 2\ -2\ -4\ -2\ -2\ 2\ 2]$ $\alpha = [0.1\ 0.08\ 0.08\ 0.1\ 0.5\ 0.1\ 0.5\ 0.5\ 0.5\ 0.5]$ $X = [15\ 20\ 25\ 10\ 5\ 15\ 8\ 21\ 25\ 5]$ $Y = [20\ 15\ 10\ 10\ 10\ 5\ 25\ 25\ 16\ 14]$
Peaks	$J = 3(1-\theta_1)^2 e^{-(\theta_2+1)^2 - \theta_1^2} -$ $\frac{1}{3} e^{-(\theta_1+1)^2 - \theta_2^2} -$ $10\left(\frac{\theta_1}{5} - \theta_1^3 - \theta_2^5\right) e^{-\theta_1^2 - \theta_2^2}$
DeJong F1	$J = \sum_{i=1}^{n} x_i^2$
Grienwank	$J = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Rastrigin	$J = \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) + 10$
Ackley	$J = -20e^{-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{30}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$
DeJong F2	$J = 100\left(x^2 - y\right)^2 + (1-x)^2$
Schaffer F6	$J = 0.5 - \frac{\left(\sin\sqrt{x^2+y^2}\right)^2 - 0.5}{\left(1 + 0.001\left(x^2+y^2\right)\right)^2}$
Schewefel	$J = \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i\rvert}$

Table 2. Benchmark functions equations

Name	D	Range	J_{min}
SCB	2	$[-2,2;-1,1]$	-1.0316
PFUNC	2	$[0,30]^2$	-3.9867
Peaks	2	$[-3,3]^2$	-6.5511
DeJong F1	30	$[-100,100]^n$	0.0000
Grienwank	30	$[-100,600]^n$	0.0000
Rastrigin	30	$[-5.12,5.12]^n$	0.0000
Ackley	30	$[-32,32]^n$	0.0000
DeJong F2	2	$[-100,100]^2$	0.0000
Schaffer F6	2	$[-100,100]^2$	0.0000
Schewefel	30	$[-500,500]^n$	-12569.5

Table 3. Benchmark functions parameters

lower [23]. If we have $n = 100$ for both groups, the lower limit for a 95% confidence range is 1.66055. According to t value is possible to conclude about the performance of the test algorithm, if $t \leq -1.66055$ the performance is better than the control algorithm, if $-1.66055 < t < 1.06605$ the performance is equal, and the performance is lower if $1.66055 \leq t$.

$$t = \frac{\bar{x}_T - \bar{x}_C}{\sqrt{\frac{\sigma_T^2}{n_T} + \frac{\sigma_C^2}{n_C}}} \tag{5}$$

The results obtained in the t–test are show in Table 4, where the boldface values represent a better or equal performance of the test algorithm. The results show that the self–adaptive version of the bacteria algorithm perform better than its similar non adaptive in eight of ten tests, nevertheless it does not achieve a better performance than the particle algorithms.

5 Conclusions

This paper presented a self–adaptive bacteria swarm algorithm for optimization, using a methodology that uses some concepts included in the Evolution Strategies for the parameter control, allowing the algorithm to select online the best parameter set. The methodology considers the swarm as a middle level structure characterized by its strategy parameters, which are modified through artificial evolution in order to find better ones.

The implementation of the SABSFO allowed us to understand how the self–adaptation mechanism aided in the search for better results in several test functions and in the adaptive control design problem. The performance of the self–adaptive algorithm was in most cases better than the non adaptive one. We conclude that the self–adaptation allows a higher level of space exploration and it can be compared to a controlled systematic test of a parameter set. The variations allow the algorithm to find better values than the non adaptive one, which is confirmed with the lower values in the t–test.

Further work should include the study of convergence and stability of the proposed algorithm, and the search of new applications. The self–adaptation represents an improvement that can be useful in search spaces where the cost function is dynamic, e.g. the error surface of an adaptive control system.

Acknowledgements

This work was partly supported by Colciencias and Universidad del Valle, through a graduate research scholarship awarded to Mario A. Muñoz, grant No. 1106–11–17707.

References

[1] W. Alfonso. Regulación de temperatura en la plataforma uv–ptm01 basada en agentes cooperativos para la asignación dinámica de recursos. Technical report, Universidad del Valle, 2007.

[2] W. Alfonso, M. Muñoz, J. López, and E. Caicedo. Optimización de funciones inspirada en el comportamiento de búsqueda de néctar en abejas. In *Congreso Internacional de Inteligenicia Computacional (CIIC2007)*, 2007.

[3] M. Clerc. The swarm and the queen: towards a deterministic and adaptiveparticle swarm optimization. In *1999 Congress on Evolutionary Computation (CEC 99)*, 1999.

[4] M. Dorigo and G. Di Caro. *The Ant Colony Optimization Meta-Heuristic*, chapter 2, pages 11–32. McGraw-Hill, 1999.

	BSFO	PSO	IWPSO	T1PSO	T2PSO	CFPSO	ABHA
F1	**-2.3453**	2.7741	2.7742	2.7742	2.7742	2.7742	**1.4092**
F2	**-5.9848**	**-2.0273**	**-1.4164**	**-2.0273**	**-2.0277**	**-0.9932**	**-2.3908**
F3	**-7.5561**	**-1.4190**	2.8932	2.8932	2.8932	**2.8932**	**-4.8163**
F4	**-32.040**	8.9695	8.8671	8.4584	8.6750	8.6015	**-21.654**
F5	**-26.691**	9.5866	9.4790	9.1368	9.3111	9.2879	**-18.476**
F6	**-3.6627**	40.063	37.556	34.735	35.582	36.870	**-4.7470**
F7	6.9485	322.38	144.04	121.43	119.70	116.19	4.4015
F8	3.2030	3.8583	3.8583	3.8583	3.8583	3.8583	3.6540
F9	**-4.8165**	2.6909	2.6909	2.6909	2.6909	2.6909	**-3.2023**
F10	**-9.3677**	3.3809	2.4608	**-0.1131**	**0.1202**	3.1521	**-4.5197**

Table 4. T–test results for the SABSFO algorithm

[5] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: An autocatalytic optimizing process. Technical Report 91-016, Politecnico di Milano, Italy, 1991.

[6] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In F. Glover and G. Kochenberger, editors, *Metaheuristics Handbook*, International Series in Operations Research and Management Science. Kluwer, 2001.

[7] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[8] E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[9] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, October 2005.

[10] J. Kennedy. The particle swarm: Social adaptation of knowledge. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[11] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1498, 1995.

[12] D. H. Kim and J. H. Cho. Adaptive tuning of pid controller for multivariable system using bacterial foraging based optimization. In *Advances in Web Intelligence*, volume 3528 of *Lecture Notes in Computer Science*, pages 231–235. Springer, 2005.

[13] Y. Liu and K. M. Passino. Biomimicry of social foraging bacteria for distributed optimization: Models, principles and emergent behaviors. *Journal of Optimization Theory and Applications*, 115(3):603–628, 2002.

[14] V. Miranda and N. Fonseca. Epso – evolutionary particle swarm optimization, a new algorithm with applications in power systems. In *Asia Pacific Transmission and Distribution Conference and Exhibition*, volume 2, pages 745–750, 2002.

[15] M. Muñoz, J. López, and E. Caicedo. An artificial bee hive for continuous optimization. Aceptado en International Journal of Intelligent Systems, 2008.

[16] K. M. Passino. Distributed optimization and control using only a germ of intelligence. In *2000 IEEE International Symposium on Intelligent Control*, pages 5–13, 2000.

[17] K. M. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, 22(3):52–67, 2002.

[18] D. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm – a novel tool for complex optimisation problems. In *Innovative Production Machines and Systems Virtual Conference*, 2006.

[19] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann–Holzboog, 1973.

[20] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 591–600, 1998.

[21] D. Teodorovic and M. Dell'Orco. Bee colony optimization – a cooperative learning approach to complex transportation problems. In *10th EWGT Meeting and 16th Mini-EURO Conference*, 2005.

[22] I. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85:317–325, 2003.

[23] W. Trochim. The t–test, 2006.

[24] T. White. What is swarm intelligence?, 2006.

[25] Wikipedia. Swarm intelligence, 2008.

[26] L. Yan-jun and W. Tie-jun. An adaptative ant colony system algorithm for continuous-space optimization problems. *Journal of Zhejiang University SCIENCE*, 4(1):40–46, 2003.

[27] J. Yisu, J. Knowles, L. Hongmei, L. Yizeng, and D. B. Kell. The landscape adaptive particle swarm optimizer. *Applied Soft Computing*, 8(1):295–304, 2007.