Compact Bacterial Foraging Optimization*

Giovanni Iacca, Ferrante Neri, and Ernesto Mininno

Department of Mathematical Information Technology,
P.O. Box 35 (Agora), 40014 University of Jyväskylä, Finland
giovanni.iacca@jyu.fi, ferrante.neri@jyu.fi, ernesto.mininno@jyu.fi

Abstract. Compact algorithms are Estimation of Distribution Algorithms which mimic the behavior of population-based algorithms by means of a probabilistic representation of the population of candidate solutions. Compared to an actual population, a probabilistic model requires a much smaller memory, which allows algorithms with limited memory footprint. This feature is extremely important in some engineering applications, e.g. robotics and real-time control systems. This paper proposes a compact implementation of Bacterial Foraging Optimization (cBFO). cBFO employs the same chemotaxis scheme of population-based BFO, but without storing a swarm of bacteria. Numerical results, carried out on a broad set of test problems with different dimensionalities, show that cBFO, despite its minimal hardware requirements, is competitive with other memory saving algorithms and clearly outperforms its population-based counterpart.

1 Introduction

Bacterial Foraging Optimization (BFO), see [8,18], is a meta-heuristic inspired by the foraging behavior of the E. coli bacteria within some environment with a non-uniform distribution of nutrients. The basic idea is to explore the search space performing tentative moves similar to the swim foraging pattern (called "chemotaxis") observed in motile bacteria. Bacterial chemotaxis is a complex combination of two types of moves, namely tumbling (i.e. changes of direction) and swimming (i.e. moves along a successful direction), which respectively enable the bacteria to search for nutrients in random directions and rapidly approach higher concentrations of nutrients. In other words, the alternation between "swims" and "tumbles" guarantees a balance between exploitation and exploration of the search space, thus making BFO robust and versatile.

Like other Swarm Intelligence algorithms, BFO has been successfully applied to many practical problems. For example, in [10] BFO is applied in image processing. In [13] a hybrid algorithm composed of BFO and a GA is used to tune a PID controller. In [20], BFO is used to design UPFC controllers. In [7], BFO is used to calibrate a volatility option pricing model.

* This research is supported by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing and Tutkijatohtori 140487, Algorithmic Design and Software Implementation: a Novel Optimization Platform.

L. Rutkowski et al. (Eds.): SIDE 2012 and EC 2012, LNCS 7269, pp. 84–92, 2012.

Despite its versatility, however, BFO shows a poor convergence behavior, compared to other meta-heuristics, especially over high dimensional complex optimization problems. To overcome these issues, different strategies have been proposed. In [5,19], cooperative approaches are used to improve the performance of standard BFO. In [3], instead, BFO has been hybridized with Particle Swarm Optimization. Recently, some adaptive and self-adaptive variants of the original BFO have been proposed, see e.g. [4,6] and [9,10]. In [21] the foraging mechanism is combined with an EDA and applied in predictive control.

This paper introduces a compact implementation of BFO, called cBFO. The cBFO algorithm belongs to the class of compact Evolutionary Algorithms (cEA), i.e. optimization algorithms which do not store and process an entire population of individuals, but make use of a probabilistic representation of the population. Thus, a run of these algorithms requires a limited memory compared to their correspondent standard EAs. These algorithms have been developed in order to address industrial optimization problems characterized by limited memory resources, e.g. in mobile robots and control systems, where a powerful computer may be unavailable due to cost and/or space limitations. The remainder of this paper is organized as follows. Section 2 describes the proposed cBFO. Section 3 shows the numerical results of an extensive test on the performance of cBFO compared to a set of algorithms. Section 4 gives the conclusion of this work.

2 Compact Bacterial Foraging Optimization

The classical BFO consists of three phases, namely: 1) chemotaxis, 2) reproduction, and 3) dispersal. During chemotaxis, the movement of the i-th bacterium is modeled as $x_i = x_i + C_i \cdot \Delta_i / \sqrt{\Delta_i^T \Delta}$, where Δ_i is the direction vector of the chemotactic step (being Δ_i^T its transpose), and C_i is a parameter which controls the step size. In tumbles, Δ_i is a random vector whose elements are uniformly distributed in $[-1, 1]$; in swims instead, Δ_i is the same as the last chemotactic step, thus allowing the bacterium to exploit a promising direction. To mimic the asexual reproduction of E. coli, at each iteration BFO sorts all the bacteria according to their fitness and selects the best half of the swarm. Each survivor is then splitted into two replicas, thus keeping the swarm size constant. Finally, in order to prevent premature convergence and keep a high diversity rate, after a fixed number of chemotaxis/reproduction steps a few bacteria are chosen, with some probability, for being replaced with new random individuals.

The original population-based BFO framework can be implemented as a compact algorithm almost straightforwardly. For the sake of clarity, the resulting algorithm, here called cBFO, is shown in Alg. 1. Without loss of generality, let us assume that parameters are normalized so that each search interval is $[-1, 1]$. cBFO consists of the following. A $2 \times n$ matrix, namely perturbation vector $PV = [\mu, \sigma]$, is generated. μ values are set equal to 0 while σ values are set equal to a large number $\lambda = 10$. The value of λ is empirically set in order to simulate a uniform distribution at the beginning of the optimization process. A solution x_e is then sampled from a multivariate Gaussian Probability Distribution Function

(PDF) characterized by a mean value μ and a standard deviation σ. For further details on sampling, see [15]. The solution x_e is called *elite*. Subsequently, at each chemotactic step, a new solution is sampled and a combination of tumble/swim moves is attempted, in the same way as in the population-based BFO. Every time a new offspring is generated, either by sampling or tumble/swim, its fitness is computed and compared with the fitness of the current *elite*. On the basis of their fitness values, a *winner* solution (solution displaying the best fitness) and a *loser* solution (solution displaying the worst fitness) are detected. The winner solution biases the virtual population by affecting the PV values, according to the following update rules:

$$\begin{aligned} \mu^{t+1} &= \mu^t + \tfrac{1}{N_p}\left(winner - loser\right) \\ \sigma^{t+1} &= \sqrt{\left(\sigma^t\right)^2 + \left(\mu^t\right)^2 - \left(\mu^{t+1}\right)^2 + \tfrac{1}{N_p}\left(winner^2 - loser^2\right)} \end{aligned} \tag{1}$$

where N_p is a parameter, namely virtual population size. Details for constructing Eg. 1 are given in [14]. In addition to the PV values, also the *elite* is updated, according to a persistent elitism scheme, see [2].

The compact implementation of reproduction and elimination/dispersal deserves an explanation. While BFO keeps the best $S/2$ bacteria and replicate them, cBFO "shifts" the PDF towards the *elite* and "shrinks" over it. In other words, the fitness-based comparison described above is applied to μ and *elite*, and the PV is updated accordingly. In this way, asexual reproduction is crudely approximated by a forced update of the PDF. As for the elimination/dispersal step, the injection of new randomly generated bacteria into the swarm is modeled by means of a perturbation of PV. More specifically, the following perturbation is applied, see [17]:

$$\begin{aligned} \mu^{t+1} &= \mu^{t+1} + 2\tau \cdot rand\left(0,1\right) - \tau \\ \sigma^{t+1} &= \sqrt{\left(\sigma^{t+1}\right)^2 + \tau \cdot rand\left(0,1\right)} \end{aligned} \tag{2}$$

where $\tau = 0.1$ is a constant parameter.

3 Numerical Results

The numerical results are divided in three groups, namely results from the testbed in [16] (24 problems) in 10, 20 and 40 dimensions. For each of the three groups, the following algorithms, with the parameter setting suggested in the original paper, have been compared to cBFO:

- Simplified Intelligence Single Particle Optimization: ISPO proposed in [23], with acceleration $A = 1$, acceleration power factor $P = 10$, learning coefficient $B = 2$, learning factor reduction ratio $S = 4$, minimum threshold on learning factor $E = 1e - 5$, and particle learning steps $PartLoop = 30$;

```
{** PV initialization **}
initialize μ = 0̄ and σ = 1̄ · 10
generate elite by means of PV
while budget condition do
   {** chemotaxis **}
   for i = 1 : N_p do
      generate 1 individual x_i by means of PV
      [winner, loser] = compete(x_i, elite)
      update μ, σ and elite
      J_last = f_{x_i}
      {** tumble and move **}
      x_i = x_i + C_i · Δ_i / sqrt(Δ_i^T Δ), with random Δ_i ∈ [−1, 1]^n
      {** swim **}
      for i = 1 : N_s do
         [winner, loser] = compete(x_i, elite)
         update μ, σ and elite
         if f_{x_i} < J_last then
            J_last = f_{x_i}
            x_i = x_i + C_i · Δ_i / sqrt(Δ_i^T Δ), with same direction vector Δ_i
         end if
      end for
   end for
   {** reproduction: shift μ towards elite **}
   [winner, loser] = compete(μ, elite)
   update μ and σ
   {** elimination/dispersal: perturb PV **}
   perturb PV according to Eq. 2
end while
```

Algorithm 1: cBFO pseudo-code

- compact Differential Evolution: cDE proposed in [15], employing rand/1/ mutation, binary crossover and persistent elitism, with virtual population size $N_{pop} = 300$, scale factor $F = 0.5$, and crossover rate $Cr = 0.3$;
- Adaptive Bacterial Foraging Optimization: ABFO0 (hereafter simply called ABFO) proposed in [6], with number of bacteria $S = 50$, initial chemotactic step size $C_{initial} = 0.1$, swim steps $N_s = 4$, probability of elimination/ dispersion $P_{ed} = 0.25$, initial epsilon $\varepsilon_{initial} = 100$, adaptation generations $n = 10$, C_i reduction ratio $\alpha = 10$, and ε reduction ratio $\beta = 10$.

As for cBFO, the following parameter setting has been used: number of bacteria $N_p = 300$, chemotactic step size $C_i = 0.1$, swim steps $N_s = 4$. Similarly to the cDE scheme, in this case the number of bacteria represents the virtual population size used in the probabilistic model of the population. The reason for setting the value of N_p much larger than S is that, since it controls the convergence of the compact framework, a lower value would cause premature convergence. On the other hand, a high value of N_p guarantees a fair balance between exploration - in the early stages - and exploitation in the later stages. The value $N_p = 300$ has been chosen empirically after a series of numerical experiments. It should be noticed that the aforementioned set of algorithms has been chosen diverse in terms of search logic. ABFO is a typical population based algorithm which requires a proper population size dependent on the dimensionality of the problem. This fact makes the memory requirement of ABFO heavily dependent on the dimensionality of the problem. On the other hand, ISPO, cDE and cBFO

Table 1. Average final results ± standard deviation and Wilcoxon test in 10 dimensions

Fcn.	ISPO	W	cDE	W	ABFO	W	cBFO
f_1	**7.948e+01 ± 1.26e-14**	-	**7.948e+01 ± 1.07e-14**	-	7.950e+01 ± 3.33e-03	+	**7.948e+01 ± 5.84e-04**
f_2	**-2.099e+02 ± 9.65e-14**	-	**-2.099e+02 ± 1.00e-10**	-	3.171e+03 ± 2.72e+03	+	2.289e+02 ± 2.72e+02
f_3	-3.966e+02 ± 7.41e+01	=	**-4.507e+02 ± 5.00e+00**	-	5.323e+04 ± 7.91e+04	+	-4.191e+02 ± 9.77e+00
f_4	-4.319e+02 ± 1.55e+01	=	**-4.339e+02 ± 1.21e+01**	-	-1.572e+02 ± 7.42e+01	+	-4.233e+02 ± 1.78e+01
f_5	**-9.210e+00 ± 0.00e+00**	-	**-9.210e+00 ± 3.87e-09**	-	-3.895e+00 ± 1.84e+00	+	-9.182e+00 ± 4.95e-03
f_6	1.006e+04 ± 1.34e+04	+	3.052e+02 ± 3.31e+02	+	8.305e+03 ± 3.19e+03	+	**3.710e+01 ± 3.41e-01**
f_7	1.004e+02 ± 5.59e+00	+	1.075e+02 ± 1.10e+01	+	9.966e+01 ± 3.22e+00	+	**9.445e+01 ± 8.16e-01**
f_8	1.843e+02 ± 7.71e+01	=	1.587e+02 ± 1.49e+01	=	1.591e+02 ± 3.76e+00	+	**1.559e+02 ± 2.13e+00**
f_9	**1.262e+02 ± 1.12e+00**	-	1.756e+02 ± 5.90e+01	+	1.319e+02 ± 9.31e-01	=	1.388e+02 ± 2.50e+01
f_{10}	3.789e+05 ± 4.23e+05	+	5.998e+03 ± 5.88e+03	=	3.413e+03 ± 1.92e+03	=	**3.239e+03 ± 2.39e+03**
f_{11}	2.673e+04 ± 1.34e+04	+	2.318e+03 ± 9.99e+02	+	8.704e+03 ± 4.03e+03	+	**8.985e+01 ± 6.14e+00**
f_{12}	**-5.732e+02 ± 3.16e+01**	-	-5.375e+02 ± 2.85e+02	-	1.360e+04 ± 6.23e+03	+	1.699e+03 ± 2.20e+03
f_{13}	5.572e+01 ± 1.78e+01	=	**4.584e+01 ± 1.46e+01**	-	6.169e+01 ± 1.15e+01	=	5.485e+01 ± 1.66e+01
f_{14}	**-5.235e+01 ± 9.04e-05**	-	**-5.235e+01 ± 8.57e-03**	-	-1.150e+01 ± 6.83e+00	+	**-5.234e+01 ± 1.38e-03**
f_{15}	1.580e+03 ± 2.25e+02	+	**1.036e+03 ± 1.44e+01**	=	1.086e+03 ± 1.97e+01	+	1.043e+03 ± 1.74e+01
f_{16}	8.363e+02 ± 7.75e+02	+	**7.670e+01 ± 4.31e+00**	-	1.306e+03 ± 3.45e+02	+	7.999e+01 ± 4.30e+00
f_{17}	2.517e+01 ± 2.03e+01	+	-1.537e+01 ± 6.97e-01	+	9.247e+00 ± 1.05e+01	+	**-1.598e+01 ± 7.91e-01**
f_{18}	1.995e+02 ± 1.38e+02	+	-1.135e+01 ± 3.70e+00	+	7.463e+01 ± 3.25e+01	+	**-1.376e+01 ± 1.87e+00**
f_{19}	-7.443e+01 ± 1.31e+01	+	**-1.006e+02 ± 6.62e-01**	-	-3.394e+01 ± 1.95e+01	+	-1.000e+02 ± 7.79e-01
f_{20}	-5.453e+02 ± 2.68e-01	=	**-5.455e+02 ± 3.46e-01**	-	-5.103e+02 ± 5.80e+01	+	-5.453e+02 ± 2.72e-01
f_{21}	5.317e+01 ± 1.40e+01	=	**4.450e+01 ± 5.11e+00**	-	5.004e+01 ± 1.18e+01	=	4.728e+01 ± 5.57e+00
f_{22}	-9.889e+02 ± 1.74e+01	=	**-9.980e+02 ± 1.40e+00**	=	-9.261e+02 ± 1.00e+01	+	-9.963e+02 ± 4.88e+00
f_{23}	1.108e+01 ± 5.90e+00	+	**7.601e+00 ± 2.62e-01**	-	9.848e+00 ± 1.18e+00	+	8.126e+00 ± 2.69e-01
f_{24}	5.951e+02 ± 1.12e+02	+	1.515e+02 ± 1.76e+01	=	1.808e+02 ± 1.45e+01	+	**1.467e+02 ± 1.13e+01**

Table 2. Average final results ± standard deviation and Wilcoxon test in 20 dimensions

Fcn.	ISPO	W	cDE	W	ABFO	W	cBFO
f_1	**7.948e+01 ± 2.96e-15**	-	8.005e+01 ± 1.31e+00	=	7.952e+01 ± 8.02e-03	+	**7.949e+01 ± 1.21e-03**
f_2	**-2.099e+02 ± 5.81e-14**	-	5.836e+00 ± 4.53e+02	-	1.582e+04 ± 1.10e+04	+	2.847e+03 ± 1.33e+03
f_3	**-3.565e+02 ± 8.22e+01**	-	3.295e+02 ± 4.77e+02	+	8.079e+05 ± 5.97e+05	+	-3.064e+02 ± 4.05e+01
f_4	**-3.919e+02 ± 2.29e+01**	-	-3.149e+02 ± 5.65e+01	=	1.012e+01 ± 3.63e+01	+	-2.871e+02 ± 6.12e+01
f_5	**-9.210e+00 ± 0.00e+00**	-	-8.718e+00 ± 7.82e-01	-	-2.516e+00 ± 1.16e+00	+	-8.025e+00 ± 1.70e+00
f_6	7.985e+03 ± 6.94e+03	+	4.603e+03 ± 1.71e+03	+	9.264e+06 ± 3.46e+06	+	**4.142e+01 ± 1.81e+00**
f_7	1.486e+02 ± 6.80e+01	+	1.671e+02 ± 3.18e+01	+	1.113e+02 ± 9.17e+00	+	**1.049e+02 ± 4.71e+00**
f_8	**1.711e+02 ± 3.03e+01**	-	4.695e+02 ± 6.53e+02	+	1.900e+02 ± 1.71e+01	+	1.831e+02 ± 3.09e+01
f_9	**1.350e+02 ± 2.56e+00**	-	4.284e+02 ± 3.52e+02	+	1.457e+02 ± 1.70e+00	-	1.674e+02 ± 4.47e+01
f_{10}	8.883e+05 ± 8.17e+05	+	1.517e+05 ± 1.25e+05	+	**1.108e+04 ± 4.28e+03**	=	1.675e+04 ± 1.34e+04
f_{11}	8.075e+04 ± 3.37e+04	+	4.852e+03 ± 1.32e+03	+	2.465e+04 ± 6.03e+03	+	**1.046e+02 ± 1.06e+01**
f_{12}	5.344e+06 ± 2.62e+07	+	2.939e+06 ± 4.56e+06	+	6.616e+04 ± 1.97e+04	+	**7.354e+03 ± 3.86e+03**
f_{13}	**5.491e+01 ± 2.15e+01**	-	3.682e+02 ± 1.40e+02	+	7.406e+01 ± 7.61e+00	=	6.921e+01 ± 2.49e+01
f_{14}	**-5.235e+01 ± 2.92e-04**	-	-4.930e+01 ± 2.37e+00	+	-6.687e+00 ± 3.03e+00	+	-5.233e+01 ± 4.07e-03
f_{15}	2.516e+03 ± 4.52e+02	+	1.178e+03 ± 4.71e+01	+	1.434e+03 ± 7.67e+01	+	**1.150e+03 ± 4.12e+01**
f_{16}	1.183e+03 ± 6.96e+02	+	1.440e+02 ± 5.22e+01	+	2.394e+03 ± 3.74e+02	+	**9.041e+01 ± 7.95e+00**
f_{17}	1.638e+01 ± 1.19e+01	+	**-1.298e+01 ± 9.02e-01**	=	7.463e+00 ± 1.13e+01	+	**-1.297e+01 ± 1.53e+00**
f_{18}	1.352e+02 ± 5.22e+01	+	-1.782e+00 ± 3.71e+00	=	9.490e+01 ± 3.57e+01	+	**-3.647e+00 ± 6.73e+00**
f_{19}	-2.505e+01 ± 2.97e+01	+	**-9.775e+01 ± 1.73e+00**	=	-1.657e+01 ± 1.26e+01	+	-9.747e+01 ± 6.99e-01
f_{20}	**-5.452e+02 ± 2.19e-01**	-	-5.450e+02 ± 3.89e-01	=	-4.238e+02 ± 9.65e+01	+	-5.449e+02 ± 2.68e-01
f_{21}	5.860e+01 ± 2.02e+01	=	5.076e+01 ± 7.63e+00	=	**4.777e+01 ± 7.89e+00**	=	4.998e+01 ± 1.22e+01
f_{22}	-9.873e+02 ± 1.12e+01	=	**-9.919e+02 ± 9.48e+00**	=	-9.001e+02 ± 5.66e+00	+	-9.911e+02 ± 1.25e+01
f_{23}	1.483e+01 ± 5.62e+00	+	**8.193e+00 ± 4.61e-01**	-	1.495e+01 ± 2.22e+00	+	8.800e+00 ± 2.82e-01
f_{24}	1.495e+03 ± 1.97e+02	+	2.830e+02 ± 3.95e+01	+	2.954e+02 ± 3.30e+01	+	**2.542e+02 ± 3.27e+01**

can be considered memory saving heuristics, as they require a fixed amount of memory slots which does not depend on the problem dimension. In other words, if one of these algorithms is used to tackle a large scale problem, although the slots are as long as the problem dimensionality, these slots do not increase in number. More specifically, the ISPO scheme is a typical single solution algorithm, requiring only two memory slots, one for the current best solution and the other for a trial candidate solution. The cDE and cBFO structures are memory-wise slightly more expensive than ISPO as they require, on the top of the two slots for single solution algorithms, two extra slots for the virtual population PV. This compromise is made in order to have the advantages of a population-based search and a still low memory usage.

Table 3. Average final results ± standard deviation and Wilcoxon test in 40 dimensions

Fcn.	ISPO	W	cDE	W	ABFO	W	cBFO
f_1	**7.948e+01 ± 1.22e-14**	-	1.120e+02 ± 1.73e+01	+	7.960e+01 ± 9.75e-03	+	7.952e+01 ± 4.83e-03
f_2	**-2.099e+02 ± 1.12e-13**	-	4.907e+04 ± 6.04e+04	+	7.397e+04 ± 2.54e+04	+	9.396e+03 ± 2.90e+03
f_3	**-2.497e+02 ± 1.07e+02**	-	1.322e+04 ± 5.53e+03	+	2.838e+06 ± 1.44e+06	+	-3.848e+00 ± 7.46e+01
f_4	**-1.855e+02 ± 1.27e+02**	-	3.605e+02 ± 1.11e+02	+	4.467e+02 ± 1.16e+02	+	7.343e+01 ± 1.03e+02
f_5	**-9.210e+00 ± 0.00e+00**	-	1.155e+01 ± 7.56e+00	+	-1.921e+00 ± 7.38e-01	-	5.845e+00 ± 5.35e+00
f_6	2.640e+04 ± 2.19e+04	+	1.945e+04 ± 4.09e+03	+	1.983e+07 ± 4.90e+06	+	**1.019e+02 ± 4.10e+01**
f_7	6.123e+02 ± 5.34e+02	+	5.442e+02 ± 1.26e+02	+	3.082e+02 ± 6.89e+01	+	**1.623e+02 ± 2.18e+01**
f_8	**1.848e+02 ± 8.94e+01**	-	2.187e+04 ± 1.36e+04	+	2.305e+02 ± 2.61e+01	=	2.576e+02 ± 7.97e+01
f_9	**1.629e+02 ± 2.67e+01**	-	1.480e+04 ± 1.24e+04	+	1.828e+02 ± 3.98e+01	+	1.697e+02 ± 2.05e+01
f_{10}	2.349e+06 ± 2.09e+06	+	6.739e+05 ± 2.49e+05	+	6.860e+04 ± 2.07e+04	+	**5.332e+04 ± 3.01e+04**
f_{11}	2.131e+05 ± 1.01e+05	+	1.105e+04 ± 2.10e+03	+	4.285e+04 ± 6.09e+03	+	**1.436e+02 ± 1.76e+01**
f_{12}	**-6.032e+02 ± 1.13e+01**	-	7.085e+07 ± 2.97e+07	+	1.136e+05 ± 1.11e+04	+	7.161e+04 ± 4.75e+04
f_{13}	**4.581e+01 ± 2.17e+01**	-	1.319e+03 ± 2.46e+02	+	1.048e+02 ± 8.92e+00	+	9.299e+01 ± 3.23e+01
f_{14}	**-5.235e+01 ± 3.05e-04**	-	-3.288e+01 ± 5.51e+00	+	-3.147e+00 ± 1.36e+00	+	-5.231e+01 ± 7.71e-03
f_{15}	4.278e+03 ± 6.81e+02	+	1.693e+03 ± 1.18e+02	+	2.318e+03 ± 1.71e+02	+	**1.493e+03 ± 9.54e+01**
f_{16}	1.895e+03 ± 6.68e+02	+	4.456e+02 ± 1.01e+02	+	3.467e+03 ± 3.72e+02	+	**1.252e+02 ± 1.27e+01**
f_{17}	2.634e+01 ± 1.98e+01	+	-8.793e+00 ± 1.35e+00	=	2.020e+01 ± 2.17e+01	+	**-9.086e+00 ± 1.58e+00**
f_{18}	1.784e+02 ± 6.92e+01	+	**1.037e+01 ± 7.07e+00**	=	1.875e+02 ± 9.10e+01	+	1.050e+01 ± 6.20e+00
f_{19}	-3.630e-01 ± 2.66e+01	+	-8.942e+01 ± 2.43e+00	+	-7.683e+00 ± 2.79e+00	+	**-9.484e+01 ± 8.41e-01**
f_{20}	**-5.453e+02 ± 1.35e-01**	-	2.786e+03 ± 3.42e+03	+	-3.340e+02 ± 1.10e+02	+	-5.446e+02 ± 2.47e-01
f_{21}	7.488e+01 ± 2.37e+01	+	7.935e+01 ± 1.65e+01	+	7.945e+01 ± 2.81e+01	+	**4.975e+01 ± 1.46e+01**
f_{22}	-9.539e+02 ± 1.64e+01	+	-9.613e+02 ± 1.61e+01	+	-8.543e+02 ± 9.64e+00	+	**-9.889e+02 ± 8.18e+00**
f_{23}	1.659e+01 ± 6.73e+00	+	**9.141e+00 ± 4.27e-01**	-	2.298e+01 ± 3.40e+00	+	9.995e+00 ± 2.82e-01
f_{24}	2.861e+03 ± 2.59e+02	+	7.729e+02 ± 8.20e+01	+	8.291e+02 ± 8.52e+01	+	**5.534e+02 ± 3.95e+01**

For each algorithm and each test problem, 30 independent runs have been performed. The budget of each single run has been fixed equal to $5000 \cdot n$ fitness evaluations, where n is the dimensionality of the problem. All the experiments were executed using the optimization platform Kimeme, see [1]. Tables 1-3 show the obtained numerical results. Average final fitness values are computed for each algorithm and each problem over the 30 runs available. In each table, the best results are highlighted in bold face. In order to strengthen the statistical significance of the results, the Wilcoxon Rank-Sum test has also been applied according to the description given in [22], where the confidence level has been fixed to 0.95. In each table, the results of the Wilcoxon test for cBFO against the other algorithms considered in this study are displayed. A "+" indicates the case in which cBFO statistically outperforms, for the corresponding test problem, the algorithm indicated in column; a "=" indicates that a pairwise comparison leads to success of the Wilcoxon Rank-Sum test, i.e., the two algorithms have the same performance; a "−" indicates that cBFO is outperformed.

Numerical results show that cBFO has overall a respectful performance despite its limited memory requirement. In particular, cBFO outperforms, on a regular basis, ABFO (which, in turn, outperforms cBFO only in one case out of 72). This fact is an extremely interesting finding which, according to our interpretation, is related to two different counterbalancing effects. The first one is related to the population modeling of compact algorithms: the sampling mechanism indeed seems to introduce a beneficial randomness, see [15], which endows the original BFO framework with extra search moves that allow the exploration of different regions of the search space. The second effect is related to the inherent exploitative pressure which characterizes a compact algorithm, and which allows, especially in high-dimensional cases, a better exploitation of the most promising search directions.

As for the other memory saving algorithms considered in this study, a clear trend emerges. In 10-dimensional problems, cBFO is outperformed by cDE,

especially on separable functions (6 "+" and 13 "−"), while it slightly outperforms ISPO (11 "+" and 6 "−"). In 20-dimensional problems, both ISPO and cBFO display a better performance than in 10 dimensions. In particular, cBFO has a similar performance with respect to ISPO (12 "+" and 10 "−"), while it outperforms cDE (13 "+" and 3 "-"). This trend is confirmed in 40-dimensional problems, where cBFO and ISPO have a similar performance (13 "+" and 11 "−") and cDE is clearly outperformed by cBFO (21 "+" and 1 "−"). In conclusion, cBFO seems to robustly handle various landscapes and offer a good performance in several cases, especially in high-dimensional cases.

3.1 Holm-Bonferroni Procedure

In order to draw some statistically significant conclusions regarding the performance of cBFO, the Holm-Bonferroni procedure, see [11,12], for the four algorithms under study and the 72 problems under consideration has been performed. The Holm-Bonferroni procedure consists of the following. Considering the results in the tables above, the four algorithms under analysis have been ranked on the basis of their average performance calculated over the 72 test problems. More specifically, a score R_i for $i = 1, \cdots, N_A$ (where N_A is the number of algorithms under analysis, $N_A = 4$ in our case) has been assigned in the following way: for each problem, a score of 4 is assigned to the algorithm displaying the best performance, 3 is assigned to the second best, and so on. The algorithm displaying the worst performance scores 1. For each algorithm, a mean score has been calculated averaging the sum of the scores of each problem. On the basis of these scores the algorithms have been sorted. Within the calculated R_i values, cBFO has been taken as a reference algorithm. Indicating with R_0 the rank of cBFO, and with R_j for $j = 1, \cdots, N_A - 1$ the rank of one of the remaining three algorithms, the values z_j, for $j = 1, \cdots, N_A - 1$, have been calculated as $z_j = (R_j - R_0)/\sqrt{\frac{N_A(N_A+1)}{6N_{TP}}}$, where N_{TP} is the number of test problems in consideration ($N_{TP} = 72$ in our case). By means of the z_j values, the corresponding cumulative normal distribution values p_j have been calculated. These p_j values have then been compared with the corresponding $\delta/(N_A - j)$ where δ is the significance level of null-hypothesis rejection, set to 0.05 in our case. Table 4 displays ranks, z_j values, p_j values, and corresponding $\delta/(N_A - j)$. The rank of cBFO is shown in parenthesis. The values of z_j and p_j are expressed in terms of z_{N_A-j} and p_{N_A-j} for $j = 1, \cdots, N_A - 1$. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected" i.e., cBFO statistically outperforms the algorithm under consideration, or "Accepted" if the distribution of values can be considered the same (there is no out-performance). Numerical results in Table 4 show that cBFO has the best rank among the algorithms considered in this study. However, the rank difference is large enough to claim that cBFO "globally" outperforms only ABFO and ISPO, while the null hypothesis is accepted when cBFO is compared to cDE, meaning that the performance of cDE and cBFO is indistinguishable on the selected benchmarks. This result is, in our opinion, remarkable, since it

Table 4. Holm-Bonferroni Procedure

$N_A - j$	Algorithm	$z_{N_A} - j$	$p_{N_A} - j$	$\delta/(N_A - j)$	Hypothesis	Rank
3	ABFO	-6.65e+00	1.48e-11	1.67e-02	Rejected	1.75
2	ISPO	-3.42e+00	3.12e-04	2.50e-02	Rejected	2.4444
1	cDE	-2.39e+00	8.46e-03	5.00e-02	Accepted	2.6667
						(3.1806)

indicates not only that cBFO clearly outperforms its population-based counterpart, but also that it represents a good alternative to a robust and versatile optimizer like cDE. Most importantly, these results confirm our previous finding, see [17], that a properly designed memory saving algorithm can successfully tackle complex problems, with different dimensionality, even better than overwhelmingly complicated population based algorithms. In this light, we think that a proper algorithmic design will allow, in the future, the integration of Computational Intelligence methods within cheap devices notwithstanding the limited hardware.

4 Conclusion

This paper introduces a novel compact optimization algorithm, namely compact Bacterial Foraging Optimization (cBFO). Like its population-based counterpart, this heuristic employs the metaphor of the chemotaxis mechanism which occurs in bacterial foraging. An extensive set of test problems has been considered for algorithmic testing. Numerical results show that, despite an extremely limited memory footprint, cBFO clearly outperforms one of the most recent implementations of population-based BFO which employs adaptation. In addition, cBFO is competitive with another compact algorithm employing a different logic, i.e. the compact Differential Evolution. Further studies will investigate the introduction of adaptive and self-adaptive schemes in the cBFO framework here proposed.

References

1. Cyber Dyne Srl Home Page, `http://cyberdynesoft.it/`
2. Ahn, C.W., Ramakrishna, R.S.: Elitism based compact genetic algorithms. IEEE Transactions on Evolutionary Computation 7(4), 367–385 (2003)
3. Biswas, A., Dasgupta, S., Das, S., Abraham, A.: Synergy of PSO and Bacterial Foraging Optimization: A Comparative Study on Numerical Benchmarks. In: Corchado, E., et al. (eds.) Innovations in Hybrid Intelligent Systems. ASC, vol. 44, pp. 255–263. Springer, Heidelberg (2007)
4. Chen, H., Zhu, Y., Hu, K.: Self-adaptation in Bacterial Foraging Optimization algorithm. In: Proc. 3rd International Conference on Intelligent System and Knowledge Engineering, ISKE 2008, vol. 1, pp. 1026–1031 (November 2008)
5. Chen, H., Zhu, Y., Hu, K.: Cooperative bacterial foraging optimization. Discrete Dynamics in Nature and Society 2009 (2009)
6. Chen, H., Zhu, Y., Hu, K.: Adaptive bacterial foraging optimization. Abstract and Applied Analysis 2011 (2011)

7. Dang, J., Brabazon, A., O'Neill, M., Edelman, D.: Option Model Calibration Using a Bacterial Foraging Optimization Algorithm. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 113–122. Springer, Heidelberg (2008)
8. Das, S., Biswas, A., Dasgupta, S., Abraham, A.: Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications. Foundations of Computational Intelligence (3), 23–55 (2009)
9. Dasgupta, S., Das, S., Abraham, A., Biswas, A.: Adaptive computational chemotaxis in bacterial foraging optimization: an analysis. Trans. Evol. Comp. 13(4), 919–941 (2009)
10. Dasgupta, S., Das, S., Biswas, A., Abraham, A.: Automatic circle detection on digital images with an adaptive bacterial foraging algorithm. Soft Comput. 14(11), 1151–1164 (2010)
11. García, S., Fernández, A., Luengo, J., Herrera, F.: A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Computing 13(10), 959–977 (2008)
12. Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6(2), 65–70 (1979)
13. Kim, D.H., Abraham, A., Cho, J.H.: A hybrid genetic algorithm and bacterial foraging approach for global optimization. Information Sciences 177(18), 3918–3937 (2007)
14. Mininno, E., Cupertino, F., Naso, D.: Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions on Evolutionary Computation 12(2), 203–219 (2008)
15. Mininno, E., Neri, F., Cupertino, F., Naso, D.: Compact differential evolution. IEEE Transactions on Evolutionary Computation 15(1), 32–54 (2011)
16. Hansen, N.: Auger, A., Finck, S., Ros, R., et al.: Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions. Tech. Rep. RR-6829, INRIA (2010)
17. Neri, F., Iacca, G., Mininno, E.: Disturbed exploitation compact differential evolution for limited memory optimization problems. Information Sciences 181(12), 2469–2487 (2011)
18. Passino, K.M.: Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Systems Magazine 22(3), 52–67 (2002)
19. Shao, Y., Chen, H.: The optimization of cooperative bacterial foraging. In: World Congress on Software Engineering, vol. 2, pp. 519–523 (2009)
20. Tripathy, M., Mishra, S., Venayagamoorthy, G.: Bacteria foraging: A new tool for simultaneous robust design of upfc controllers. In: IJCNN 2006: International Joint Conference on Neural Networks, pp. 2274–2280 (2006)
21. Wang, X.S., Cheng, Y.H., Hao, M.L.: Estimation of distribution algorithm based on bacterial foraging and its application in predictive control. Dianzi Xuebao (Acta Electronica Sinica) 38(2), 333–339 (2010)
22. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1(6), 80–83 (1945)
23. Zhou, J., Ji, Z., Shen, L.: Simplified intelligence single particle optimization based neural network for digit recognition. In: Proceedings of the Chinese Conference on Pattern Recognition (2008)