Bacterial Foraging Optimization Algorithm[*]

Jason Brownlee
jasonb@CleverAlgorithms.com
The Clever Algorithms Project
http://www.CleverAlgorithms.com

November 21, 2010
Technical Report: CA-TR-20101121-1

Abstract

The Clever Algorithms project aims to describe a large number of Artificial Intelligence algorithms in a complete, consistent, and centralized manner, to improve their general accessibility. The project makes use of a standardized algorithm description template that uses well-defined topics that motivate the collection of specific and useful information about each algorithm described. This report describes the Bacterial Foraging Optimization Algorithm using the standardized algorithm description template.

Keywords: `Clever, Algorithms, Description, Optimization, Bacterial, Foraging, Optimization, Algorithm`

1 Introduction

The Clever Algorithms project aims to describe a large number of algorithms from the fields of Computational Intelligence, Biologically Inspired Computation, and Metaheuristics in a complete, consistent and centralized manner [1]. The project requires all algorithms to be described using a standardized template that includes a fixed number of sections, each of which is motivated by the presentation of specific information about the technique [2]. This report describes the Bacterial Foraging Optimization Algorithm using the standardized algorithm description template.

2 Name

Bacterial Foraging Optimization Algorithm, BFOA, Bacterial Foraging Optimization, BFO

3 Taxonomy

The Bacterial Foraging Optimization Algorithm belongs to the field of Bacteria Optimization Algorithms and Swarm Optimization, and more broadly to the fields of Computational Intelligence and Metaheuristics. It is related to other Bacteria Optimization Algorithms such as the Bacteria Chemotaxis Algorithm [5], and other Swarm Intelligence algorithms such as Ant Colony Optimization and Particle Swarm Optimization. There have been many extensions of the approach that attempt to hybridize the algorithm with other Computational Intelligence algorithms and Metaheuristics such as Particle Swarm Optimization, Genetic Algorithm, and Tabu Search.

4 Inspiration

The Bacterial Foraging Optimization Algorithm is inspired by the group foraging behavior of bacteria such as E.coli and M.xanthus. Specifically, the BFOA is inspired by the chemotaxis behavior of bacteria that will perceive chemical gradients in the environment (such as nutrients) and will move toward or away from specific signals.

5 Metaphor

Bacteria perceive the direction to food based on the gradients of chemicals in their environment. Similarly, bacteria secrete attracting and repelling chemicals into the environment and can perceive each other in a similar way . Using locomotion mechanisms (such as flagella) bacteria can move around in their environment, sometimes moving chaotically (tumbling and spinning), and other times moving in a directed manner that may referred to as swimming. Bacterial cells are treated like agents in an environment, using their perception of food and other cells as motivation to move, and stochastic tumbling and swimming like movement to re-locate. Depending on the cell-cell interactions, cells may swarm a food source, may aggressively repel or ignore each other.

6 Strategy

The information processing strategy of the algorithm is to allow cells to stochastically and collectively swarm toward optima. This is achieved through a series of three processes on a population of simulated cells: 1) 'Chemotaxis' where the cost of cells is derated by the proximity to other cells and cells move along the manipulated cost surface one at a time (the majority of the work of the algorithm), 2) 'Reproduction' where only those cells that performed well over their lifetime may contribute to the next generation, and 3) 'Elimination-dispersal' where cells are discarded and new random samples are inserted with a low probability.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the Bacterial Foraging Optimization Algorithm for minimizing a cost function. A bacteria cost is derated by its interaction with other cells. This interaction is calculated as follows:

$$
\begin{aligned}
interaction(cell_k) = & \sum_{i=1}^{S} \Big[-d_{attract} \times exp\Big(-w_{attract} \times \sum_{m=1}^{P} (cell_m^k - other_m^i)^2 \Big) \Big] + \\
& \sum_{i=1}^{S} \Big[h_{repellant} \times exp\Big(-w_{repellant} \times \sum_{m=1}^{P} cell_m^k - other_m^i)^2 \Big) \Big]
\end{aligned}
$$

where $cell_k$ is a given cell, $d_{attract}$ and $w_{attract}$ are attraction coefficients, $h_{repellant}$ and $w_{repellant}$ are repulsion coefficients, S is the number of cells in the population, P is the number of dimensions on a given cells position vector.

The remaining parameters of the algorithm are as follows $Cells_{num}$ is the number of cells maintained in the population, N_{ed} is the number of elimination-dispersal steps, N_{re} is the number of reproduction steps, N_c is the number of chemotaxis steps, N_s is the number of swim steps for a given cell, $Step_{size}$ is a random direction vector with the same number of dimensions as the problem space, and each value $\in [-1, 1]$, and P_{ed} is the probability of a cell being subjected to elimination and dispersal.

8 Heuristics

- The algorithm was designed for application to continuous function optimization problem domains.
- Given the loops in the algorithm, it can be configured numerous ways to elicit different search behavior. It is common to have a large number of chemotaxis iterations, and small numbers of the other iterations.
- The default coefficients for swarming behavior (cell-cell interactions) are as follows $d_{attract} = 0.1$, $w_{attract} = 0.2$, $h_{repellant} = d_{attract}$, and $w_{repellant} = 10$.
- The step size is commonly a small fraction of the search space, such as 0.1.
- During reproduction, typically half the population with a low health metric are discarded, and two copies of each member from the first (high-health) half of the population are retained.
- The probability of elimination and dispersal (p_{ed}) is commonly set quite large, such as 0.25.

9 Code Listing

Listing 1 provides an example of the Bacterial Foraging Optimization Algorithm implemented in the Ruby Programming Language. The demonstration problem is an instance of a continuous function optimization that seeks $min f(x)$ where $f = \sum_{i=1}^{n} x_i^2$, $-5.0 \leq x_i \leq 5.0$ and $n = 2$. The optimal solution for this basin function is $(v_0, \ldots, v_{n-1}) = 0.0$. The algorithm is an implementation based on the description on the seminal work [6]. The parameters for cell-cell interactions (attraction and repulsion) were taken from the paper, and the various loop parameters were taken from the ‘Swarming Effects’ example.

```ruby
def objective_function(vector)
  return vector.inject(0.0) {|sum, x| sum + (x ** 2.0)}
end

def random_vector(problem_size, search_space)
  return Array.new(problem_size) do |i|
    search_space[i][0] + ((search_space[i][1] - search_space[i][0]) * rand())
  end
end

def generate_random_direction(problem_size)
  bounds = Array.new(problem_size){[-1.0,1.0]}
  return random_vector(problem_size, bounds)
end

def compute_cell_interaction(cell, cells, d, w)
  sum = 0.0
  cells.each do |other|
    diff = 0.0
    other[:vector].each_with_index do |other_value,i|
      diff += (cell[:vector][i] - other_value)**2.0
```

Algorithm 1: Pseudo Code for the Bacterial Foraging Optimization Algorithm.

```
Input: $Problem_{size}$, $Cells_{num}$, $N_{ed}$, $N_{re}$, $N_c$, $N_s$, $Step_{size}$, $d_{attract}$, $w_{attract}$, $h_{repellant}$,
       $w_{repellant}$, $P_{ed}$
Output: $Cell_{best}$
Population ← InitializePopulation($Cells_{num}$, $Problem_{size}$);
for $l = 0$ to $N_{ed}$ do
    for $k = 0$ to $N_{re}$ do
        for $j = 0$ to $N_c$ do
            foreach Cell ∈ Population do
                $Cell_{fitness}$ ← Cost(Cell) + Interaction(Cell, Population, $d_{attract}$, $w_{attract}$,
                $h_{repellant}$, $w_{repellant}$);
                $Cell_{health}$ ← $Cell_{fitness}$;
                $Cell\prime$ ← 0;
                for $i = 0$ to $N_s$ do
                    RandomStepDirection ← CreateStep($Problem_{size}$);
                    $Cell\prime$ ← TakeStep(RandomStepDirection, $Step_{size}$);
                    $Cell\prime_{fitness}$ ← Cost($Cell\prime$) + Interaction($Cell\prime$, Population, $d_{attract}$,
                    $w_{attract}$, $h_{repellant}$, $w_{repellant}$);
                    if $Cell\prime_{fitness} > Cell_{fitness}$ then
                        $i \leftarrow N_s$;
                    else
                        Cell ← $Cell\prime$;
                        $Cell_{health}$ ← $Cell_{health}$ + $Cell\prime_{fitness}$;
                    end
                end
                if Cost(Cell) ≤ Cost($Cell_{best}$) then
                    $Cell_{best}$ ← Cell;
                end
            end
        end
        SortByCellHealth(Population);
        Selected ← SelectByCellHealth(Population, $\frac{Cells_{num}}{2}$);
        Population ← Selected;
        Population ← Selected;
    end
    foreach Cell ∈ Population do
        if Rand() ≤ $P_{ed}$ then
            Cell ← CreateCellAtRandomLocation();
        end
    end
end
return $Cell_{best}$;
```

```ruby
    end
    sum += d * Math.exp(w * diff)
  end
  return sum
end

def compute_attract_repel(cell, cells, d_attr, w_attr, h_rep, w_rep)
  attract = compute_cell_interaction(cell, cells, -d_attr, -w_attr)
  repel = compute_cell_interaction(cell, cells, h_rep, -w_rep)
  return attract + repel
end

def evaluate(cell, cells, d_attr, w_attr, h_rep, w_rep)
  cell[:cost] = objective_function(cell[:vector])
  cell[:interaction] = compute_attract_repel(cell, cells, d_attr, w_attr, h_rep, w_rep)
  cell[:fitness] = cell[:cost] + cell[:interaction]
end

def tumble_cell(problem_size, cell, step_size)
  step = generate_random_direction(problem_size)
  vector = Array.new(problem_size) do |i|
    cell[:vector][i] + step_size * step[i]
  end
  return {:vector=>vector}
end

def chemotaxis(cells, problem_size, search_space, chem_steps, swim_length, step_size, d_attr,
    w_attr, h_rep, w_rep)
  best = nil
  chem_steps.times do |j|
    moved_cells = []
    cells.each_with_index do |cell, i|
      sum_nutrients = 0.0
      evaluate(cell, cells, d_attr, w_attr, h_rep, w_rep)
      best = cell if best.nil? or cell[:cost] < best[:cost]
      sum_nutrients += cell[:fitness]
      swim_length.times do |m|
        new_cell = tumble_cell(problem_size, cell, step_size)
        evaluate(new_cell, cells, d_attr, w_attr, h_rep, w_rep)
        best = cell if cell[:cost] < best[:cost]
        break if new_cell[:fitness] > cell[:fitness]
        cell = new_cell
        sum_nutrients += cell[:fitness]
      end
      cell[:sum_nutrients] = sum_nutrients
      moved_cells << cell
    end
    puts " >chemotaxis=#{j}, fitness=#{best[:fitness]}, cost=#{best[:cost]}"
    cells = moved_cells
  end
  return [best, cells]
end

def search(problem_size, search_space, pop_size, elim_disp_steps, repro_steps, chem_steps,
    swim_length, step_size, d_attr, w_attr, h_rep, w_rep, p_eliminate)
  cells = Array.new(pop_size) { {:vector=>random_vector(problem_size, search_space)} }
  best = nil
  elim_disp_steps.times do |l|
    repro_steps.times do |k|
      c_best, cells = chemotaxis(cells, problem_size, search_space, chem_steps, swim_length,
          step_size, d_attr, w_attr, h_rep, w_rep)
      best = c_best if best.nil? or c_best[:cost] < best[:cost]
      cells.sort{|x,y| x[:sum_nutrients]<=>y[:sum_nutrients]}
      cells = cells[0...(pop_size/2)] + cells[0...(pop_size/2)]
    end
    cells.each do |cell|
      if rand() <= p_eliminate
        cell[:vector] = random_vector(problem_size, search_space)
      end
    end
  end
  return best
end

if __FILE__ == $0
  problem_size = 2
  search_space = Array.new(problem_size) {|i| [-5, 5]}
  pop_size = 50
  step_size = 0.1 # Ci
  elim_disp_steps = 1 # Ned
  repro_steps = 4 # Nre
  chem_steps = 70 # Nc
  swim_length = 4 # Ns
  p_eliminate = 0.25 # Ped
  d_attr = 0.1
  w_attr = 0.2
  h_rep = d_attr
  w_rep = 10

  best = search(problem_size, search_space, pop_size, elim_disp_steps, repro_steps, chem_steps,
      swim_length, step_size, d_attr, w_attr, h_rep, w_rep, p_eliminate)
  puts "done! Solution: c=#{best[:cost]}, v=#{best[:vector].inspect}"
end
```

Listing 1: Bacterial Foraging Optimization Algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

Early work by Liu and Passino considered models of chemotaxis as optimization for both E.coli and M.xanthus which were applied to continuous function optimization [4]. This work was consolidated by Passino who presented the Bacterial Foraging Optimization Algorithm that included a detailed presentation of the algorithm, heuristics for configuration, and demonstration applications and behavior dynamics [6].

10.2 Learn More

A detailed summary of social foraging and the BFOA is provided in the book by Passino [8]. Passino provides a follow-up review of the background models of chemotaxis as optimization and describes the equations of the Bacterial Foraging Optimization Algorithm in detail in a Journal article [7]. Das, et al. present the algorithm and its inspiration, and go on to provide an in depth analysis the dynamics of chemotaxis using simplified mathematical models [3].

11 Conclusions

This report described the Bacterial Foraging Optimization Algorithm using the standardized algorithm description template.

12 Contribute

Found a typo in the content or a bug in the source code? Are you an expert in this technique and know some facts that could improve the algorithm description for all? Do you want to get that warm feeling from contributing to an open source project? Do you want to see your name as an acknowledgment in print?

Two pillars of this effort are i) that the best domain experts are people outside of the project, and ii) that this work is subjected to continuous improvement. Please help to make this work less wrong by emailing the author 'Jason Brownlee' at `jasonb@CleverAlgorithms.com` or visit the project website at `http://www.CleverAlgorithms.com`.

References

[1] Jason Brownlee. The clever algorithms project: Overview. Technical Report CA-TR-20100105-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[2] Jason Brownlee. A template for standardized algorithm descriptions. Technical Report CA-TR-20100107-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[3] Swagatam Das, Arijit Biswas, Sambarta Dasgupta, and Ajith Abraham. *Foundations of Computational Intelligence Volume 3: Global Optimization*, chapter Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications, pages 23–55. Springer, 2009.

[4] 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.

[5] Sibylle D. Müller, Jarno Marchetto, Stefano Airaghi, and Petros Koumoutsakos. Optimization based on bacterial chemotaxis. *IEEE Transactions on Evolutionary Computation*, 6(1):16–29, 2002.

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

[7] K. M. Passino. Bacterial foraging optimization. *International Journal of Swarm Intelligence Research*, 1(1):1–16, 2010.

[8] Kevin M. Passino. *Biomimicry for Optimization, Control, and Automation*, chapter Part V: Foraging. Springer, 2005.