IP CAT

Seventh International Workshop on
Information Processing in Cells and Tissues

29th-31st August 2007

Proceedings of the Seventh International Workshop on Information Processing in Cells and Tissues

Nigel Crook and Tjeerd olde Scheper
(editors)

August 2007

OXFORD
BROOKES
UNIVERSITY

7.2 Pomitaxis: Computing with a Bacterial-Inspired Algorithm

Dan Nicolau[1] and Philip Maini[2]

[1] Centre for Mathematical Biology, University of Oxford, United Kingdom

[2] Oxford Centre for Integrative Systems Biology, University of Oxford, United Kingdom

`nicolau@maths.ox.ac.uk`

We present a general-purpose optimisation algorithm inspired by "run-and-tumble", the biased random walk chemotactic swimming strategy used by the bacterium E coli to locate regions of high nutrient concentration The method uses particles (corresponding to bacteria) that swim through the variable space (corresponding to the attractant concentration profile). By constantly performing temporal comparisons, the particles drift towards the minimum or maximum of the function of interest. We illustrate the use of our method with three simple examples. We also present a discrete version of the algorithm. The new algorithm is expected to be useful in combinatorial optimisation problems involving many variables, where the functional landscape is apparently stochastic and has local minima, but preserves some derivative structure at the mesoscale.

Presentation Thursday 30th August 2007, 16.00

'Pomitaxis': Computing with a Bacterial-Inspired Algorithm

Dan V. Nicolau, Jr.[1], Philip K. Maini[1,2]

[1]Centre for Mathematical Biology, Mathematical Institute
University of Oxford, Oxford OX1 3LB, United Kingdom

[2]Oxford Centre for Integrative Systems Biology,
University of Oxford, Oxford OX13LB, United Kingdom

Abstract

We present a general-purpose optimisation algorithm inspired by "run-and-tumble", the biased random walk chemotactic swimming strategy used by the bacterium *E coli* to locate regions of high nutrient concentration The method uses particles (corresponding to bacteria) that swim through the variable space (corresponding to the attractant concentration profile). By constantly performing temporal comparisons, the particles drift towards the minimum or maximum of the function of interest. We illustrate the use of our method with three simple examples. We also present a discrete version of the algorithm. The new algorithm is expected to be useful in combinatorial optimisation problems involving many variables, where the functional landscape is apparently stochastic and has local minima, but preserves some derivative structure at the mesoscale.

Introduction

The correspondence between living systems and computers has been stressed in recent years (e.g. Bray, 1995, Nicolau and Nicolau, 2006). Biological processes can be thought of as processes of constrained optimization. Therefore, the mechanism or mechanisms used by a biological system to carry out a function is analogous to an algorithm or set of algorithms; the biological system is then an unconventional computer; and an instance of such a process taking place is analogous to a computational run. Of course, there are enormous differences between 'biological computing' and 'classical computing'. Biological processes are massively parallel, feature a large degree of stochasticity and noise (which, aside from being unavoidable, also plays a direct role in the computation) and, rather than being able to compute individual functions with high precision, deal instead with problems of a 'systems engineering' flavour, such as the control of very large systems, in the presence of non-linear constraints.

Because living systems are adapted to the environments in which they exist and therefore to the computational tasks required for survival, these natural computing paradigms are expected to be successful for dealing with problems similar to those confronting biosystems (Nicolau and Nicolau, 2006). An increasing number of algorithms are based on or inspired by biological strategies. These include neural networks (Basheer and Hajmeer, 2000), evolutionary computing (Eiben and Smith, 2003), DNA computing (Adleman, 1994), particle swarm optimization (Call et al., 2007), computing with bio-agents (Nicolau et al., 2007), ant optimization algorithms (Dorigo and Blum, 2005) and others. Increasingly these methods have been successfully applied to a spectrum of problems ranging from pattern identification and matching to aerodynamics engineering problems (Obayashi, 1997).

Chemotaxis, the process by which organisms direct their movements according to certain chemicals in their environment, is crucial for many biological functions. Bacteria such as *E. coli* use chemotaxis to find food (for example, glucose) by swimming towards the highest concentration of food molecules, or to flee from poisons (for example, phenol). In multicellular organisms, chemotaxis is critical to development as well as normal function (Wadhams and Armitage, 2004). By analogy with the process of finding the maximum of a function (represented by the attractant concentration profile in space), chemotaxis is a algorithm for optimisation. This computational facet of chemotaxis has already been noted by several authors (Bremermann, 1974, Muller et al., 2002). Recently, Vergassola et al. proposed a chemotaxis-inspired search method in the absence of gradients (Vergassola et al., 2007).

In this paper, we present a biocomputation approach that is based on the "run-and-tumble" chemotactic mechanism of the bacterium *E coli*. This method is essentially a general-purpose search algorithm that can be used to optimize a function or set of functions. The method bears some resemblance to Particle Swarm Optimisation (PSO) in that the potential solutions (the particles) move through the function space. However, unlike PSO or e.g. ant colony optimization, it does not use inter-particle communication and does not bias the trajectories based on the best solutions found over time, relying instead completely on the chemotactic drift property of bacteria to converge locally (not as a swarm) to solutions. We illustrate the potential of the method by applying it to three simple and representative optimisation problems. We also present a discrete version of the algorithm.

Methods

We begin by briefly describing the chemotactic swimming pattern of *Escherichia coli*, on which our algorithm is based. *E coli* is a common intestinal bacterium, cylindrical in shape and roughly 2 μm long and 1 μm wide. Each cell is equipped with approximately 6 flagella, each with a rotary motor at its base, embedded in the cell wall. The flagella are randomly distributed on the cell membrane. The rotary motor can turn clockwise and counter-clockwise at different times and is reversible. When all the motors turn in concert in a counter-clockwise direction, the flagella form a bundle that propels the cell forward in a "run". Runs are not perfectly rectilinear due to rotational Brownian motion that perturbs the cell direction by roughly $0.5\sqrt{t}$, where t is in seconds. If one or more of the motors reverse direction and turn clock-wise, the bundle becomes unstable and the cell turns in place ("tumbles") in a random fashion and with negligible displacement. This serves to reorient the cell; the orientation is not perfect and there is some persistence of direction after a tumble (the mean angle between the direction before and after a tumble is 63°) (Locsei, 2007). We omit this property in the present work, assuming that the reorientation is perfect.

E coli cells use the system of motors and flagella to execute chemotactic swimming towards regions of high nutrient concentration (or away from toxins) as follows. Because the cell is too small and the environment too stochastic to accurately measure a gradient across the cell body, a simple biochemical memory mechanism is used to perform temporal comparisons. As it is swimming, the bacterium monitors the concentration of chemoattractant (e.g. serine or aspartate) in the environment, comparing the average concentration measured over the last second with that measured over the previous 3 seconds. If the comparison indicates that the attractant concentration has increased, the cell is more likely to continue a straight-line run, while if it indicates that conditions have deteriorated, it is more likely to reorient by performing a tumble. In this way the bacterium performs a biased random walk, leading it (in a stochastic fashion) up a chemoattractant gradient. In the absence of any such gradient, both the run and tumble times are exponentially distributed with means of 1.0 and 0.1 seconds, respectively (Locsei, 2007).

We propose to use an analogous strategy to locate regions in a multi-dimensional space where a continuous (and respectively discrete) function takes a global maximum (or minimum) value. We define such a "bacterial optimiser" B as a set of n particles $(b_1, b_2, \ldots, b_n)$, each possessing an m-dimensional position vector function $p_i(t)$ and a velocity vector function $v_i(t)$ such that $p_i \in \Re^m$, $v_i \in \Re^m$, $i = 1, \ldots, n$ and $t \in \mathbf{N}$. This is the continuous version of the algorithm (a discrete version is described below). Let f: $\Re^m \rightarrow \Re$ be the objective function and let $\hat{x}_i = \max_{t>0} x_i(t)$ and $\hat{g} = \max_x f(x_i), i = 1, \ldots, n$. The algorithm proceeds as follows (the parameters are explained below):

(1) Initialise p_i and v_i for all i. A simple choice is $p_i = U[a_j, b_j]$, $j=1, \ldots, m$ and $v_i = r_1$ where r_1 is a $m \times 1$ vector each of whose entries $r_{1,j} = U[-1,1]$. a_j and b_j are the limits of the search domain in each dimension.

(2) $\hat{\mathbf{g}} = \min_x f(\mathbf{x}_i), i = 1, \ldots, n$

(3) While not converged:

```
For $1 \le i \le n$:
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \varpi \mathbf{v}_i + \beta \mathbf{r}_r$, where $\mathbf{r}_r$ is a $m \times 1$ vector,
    each of whose entries $\mathbf{r}_{r,j} = N(0,1)$

    If $f(x_i) \ge f(\hat{g})$:
        $\hat{g} \leftarrow x_i$

    $Pr(tumble) = T(A_i(t))$
```

```
                If $r \le Pr(tumble)$, where $r$ = U[0,1]:
                        $\boldsymbol{v}_i = \boldsymbol{r}_{tumble}$, where $\boldsymbol{r}_{tumble}$ is a $m\times 1$ vector
                        each of whose entries $\boldsymbol{r}_{tumble,j} = \in$ U[-1,1].
                End If

        End For

        $t \leftarrow t+1$

End While
```

Convergence can be decided either by setting an upper limit on the number of iterations $t_{\max}$ or by setting an acceptable value for $f(\hat{g})$.

The tumbling probability function T is calculated as follows:

$$T(t,i) = \begin{cases} p_w, A_i(t) < 0 \\ p_b, A_i(t) \ge 0 \end{cases}, \tag{1}$$

where

$$A_i(t) = \sum_{\tau=0}^{\min(w_r+w_d,t)} f(x_i(t-\tau))M(\tau) \tag{2}$$

and M: $\Re \to \Re$ is a memory comparison function, which can take any number of forms but which we define for simplicity here as:

$$M(\tau) = \begin{cases} 1/w_r, & 0 < \tau \le w_r \\ -1/w_d, & w_r < \tau \le w_d \\ 0, & \text{otherwise} \end{cases}. \tag{3}$$

The meaning of each of the parameters is as follows. ϖ is a speed factor for the particles in the functional search space, since the elements of $\boldsymbol{v}_i$ are bounded by -1 and 1. β is a strictly positive parameter that if greater than 0 ensures that the runs are not perfectly straight and simulates rotational Brownian motion during a run. p_w and p_b are probabilities of tumbling if conditions have improved ($A(t) \ge 0$) and deteriorated ($A(t) < 0$), respectively. In practice, the probability of tumbling must be larger if conditions have deteriorated, so we have $p_w > p_b$. w_r and w_d are the number of iterations (window lengths) over which the recent and distant past are averaged, respectively. A balance must be struck between accuracy (using longer window lengths) and fast response time to improving or deteriorating conditions (leading to the use of shorter window lengths). Because *E. coli* compares (roughly speaking) the last second of its life with the previous 3 during chemotactic swimming (Strong et al., 1998), this would suggest as simple rule that $w_d \approx 3w_r$ but in our algorithm this is likely to be sensitive to the properties of the function being optimized.

It is also possible to modify our algorithm so that it can be applied to discrete problems. The key change is to restrict the elements of the velocity vector $\boldsymbol{v}_i$ to positive values smaller than 1 and to treat these as probabilities of the entries in $\boldsymbol{x}_i$ changing. For example, for a problem in which the variables can only take binary values (0 or 1), an element of $\boldsymbol{v}_i$ equal to 0.1 means a 10% probability that the corresponding element of $\boldsymbol{x}_i$ will change state at the next iteration of the algorithm. Additionally, the speed ϖ shall be set to 1 in order for the probabilities to be guaranteed to be between 0 and 1. In problems where the variables can take a number of discrete values (for example where they can take any positive integer value), each element of $\boldsymbol{x}_i$ could, of course, be incremented by 1 or decremented by 1; therefore, the "direction" of the increment should be chosen at random. Finally, it may be desirable to use increments greater than 1 (this would correspond to using a larger speed in the continuous version). If this is done, then in order to avoid equal-sized increments at each point where a variable changes (and thus miss intermediate values), the size of the jump should be sampled from a suitable probability distribution with a mean of ξ where ξ is the average increment size.

One issue if using this discrete version of the algorithm is that a non-trivial fitness function may need to be used. In the continuous version, the fitness is evaluated simply as the value of the function at the point in m-dimensional space represented by the position vector. However, in the case of a discrete function, if the number of values that the function can assume is small or if these values are not consecutive (or both), this may not be appropriate. This is because the algorithm relies on "tumbles" being more likely when the fitness is relatively inferior and less likely when it is close to the desired value. Therefore using the value of the function as the fitness function may result in completely stochastic behaviour. We illustrate the issue using a simple (NP-complete) problem: Boolean satisfiability. Here it is required to determine, for a Boolean expression in n variables, what set of values for the variables (if any) makes the expression TRUE. Using a fitness function that simply takes the values 1 (for true) and 0 (for false) would not be advantageous in this case, reducing effectively to a random search through the function space (which would require exponential time proportional to 2^n). A more appropriate choice of fitness function may be:

$$f(\boldsymbol{x}) = \max_{\text{all } z} c(z) - c(\boldsymbol{x}) \tag{4}$$

where $c(\boldsymbol{x})$ is the number of clauses in the Boolean expression that evaluate to 0. In this way, the algorithm would favour position vectors $\boldsymbol{x}$ that result in a smaller number of such clauses evaluating to 0, thus in some sense being closer to finding a combination of variables that will cause the Boolean function to evaluate to TRUE. Of course, other possible fitness functions exist and in general, for discrete functions, it is to be expected that the choice of fitness function would vary with the problem.

Results

We implemented our algorithm using MATLAB and applied it to three different optimization problems. The first of these is trivial: finding the maximum of a two-dimensional Gaussian function. The second is finding the global minimum of a difficult two-dimensional function with many local minima. The third is concerned with how n particles should be distributed on a sphere so as to minimize the potential energy of the system.

Finding the maximum of a Gaussian function

In order to demonstrate the operation of our algorithm, we first applied it to the Gaussian function:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}} . \qquad (5)$$

The Gaussian is an attractive first choice for a two reasons. Firstly, because the fundamental solution of the diffusion equation is a Gaussian, we might expect naturally occurring attractant gradients to take this form and therefore, due to adaptation, we might expect bacterial chemotaxis to be efficient at finding the global maximum of such a function (corresponding, in vivo, to, for example, finding the point of maximum nutrient concentration in a local environment). Secondly, while being a non-trivial function, it possesses a continuous and smooth gradient that is everywhere non-zero.

Figure 1 shows a typical simulation of the algorithm. A colony of 100 particles is initially distributed at random points chosen from $x_{init} \in [-3, 3]$ and $y_{init} \in [-3, 3]$. We also randomly choose $x_0 \in [-3, 3]$ and $y_0 \in [-3, 3]$. The initial velocity vectors are also chosen at random such that directions are uniformly distributed in $[-\pi, \pi]$ and the magnitude of each direction vector is 0.02. We also set $\sigma = 1$ for simplicity. The figure shows the first 500 consecutive iterations, with the maximum value found converging to the true maximum at (x_0, y_0).

Optimising a difficult two-dimensional function

We next applied our algorithm to Problem 4 of the 100-digit challenge (Trefethen, 2002). This problem asks for the minimum of the function

$$f = e^{\sin 50x} + \sin(60e^y) + \sin(70\sin x) + \sin(\sin(80y)) - \sin(10(x+y)) + \tfrac{1}{4}(x^2+y^2) . \qquad (6)$$

It is made difficult by the presence of many local minima that are very close to the global minimum – the latter is approximately $f_{min} \approx -3.30686864747523728$ and occurs at $(x, y) \approx (-0.0244030796943785, 2.10612427155358)$. Figure 2A illustrates the difficulty, with a graph of the function showing the behaviour near the global minimum.

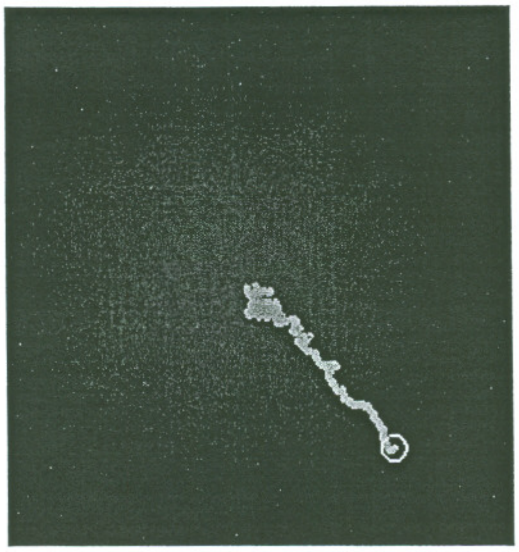

Figure 1. Finding the global maximum in a Gaussian gradient field. 500 consecutive iterates of the colony centre (starting at the lower right) are shown in green; higher values of the attractant are shown as shades of red. The circle indicates the point at which the 250 particles are initialized.

With the bacterial algorithm, setting up the problem consists of placing a number (250 in our computations) of particles at random in the two-dimensional function space near the minimum ($-5 \leq x \leq 5$, $-5 \leq y \leq 5$ are appropriate intervals) and randomising their directions. Again, the behaviour of the algorithm is good – Figure 2B shows the percentage difference of the algorithm's best estimate from f_{min} over the course of a computational run. The 2000 iterations require, for 250 particles, only 2-3 seconds of computer time (on a 1 GHz desktop machine running MATLAB) to find the minimum with 8-digit accuracy. In the simple implementation presented here, the memory function and other parameters such as the particle velocity, directional persistence etc. have not been optimized; this time would be reduced by some (unknown) factor if these steps were taken.

Finding the minimum-energy configuration of particles on a sphere

Lastly, we applied our algorithm to a difficult n-body configuration problem: how to distribute n particles on a sphere so as to minimise the potential energy

$$E = \sum_{i \neq j} \frac{1}{d(i, j)} \tag{7}$$

where $d(i, j)$ is the (great-circle) distance on the sphere between particles i and j. This problem is computationally difficult because, similarly to the n-body problem and to protein folding, the potential energy space to be searched grows very rapidly with the number of particles – at each iteration of a search algorithm, all pairwise distances must be re-evaluated. Additionally, because a small difference in even a single distance can make a large difference to the sum, a brute force search will fail due to the fine required partition of the search space. Approximations to optimal configurations for this problem are known (Hardin, Sloane & Smith, 1996) for various numbers of particles.

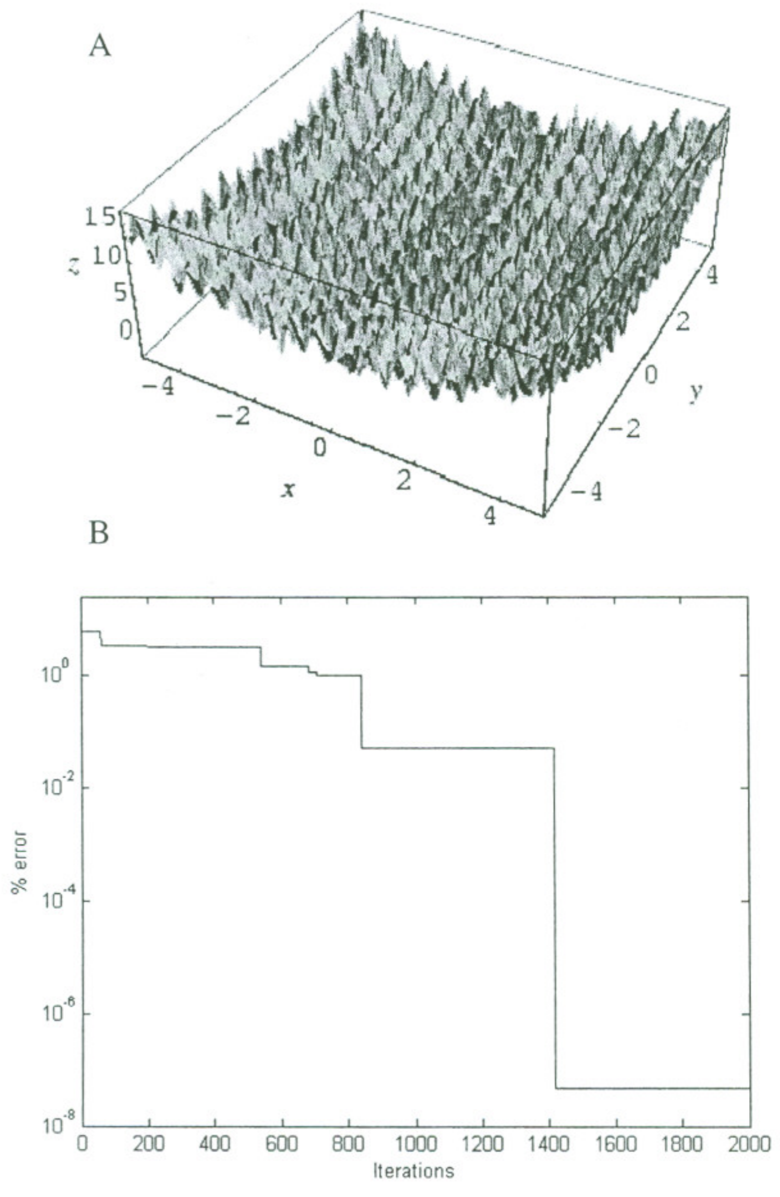

Figure 2. A. A graph of the function in Eq. (6), showing the complex behaviour near the global minimum. B. Finding the minimum of this function using taxis. The vertical axis shows the percentage difference between the best estimate, $f(\hat{\mathbf{g}})$ and the (known) global minimum.

To apply the taxis algorithm to the problem, first we convert the coordinates of the particles to spherical coordinates (latitude and longitude). For two particles *i* and *j*, let ϕ_i and ϕ_j be the latitudes and λ_i and λ_j the longitudes. Then the great-circle distance on a sphere of unit radius is:

$$d(i, j) = \arccos\{\sin\phi_i \sin\phi_j + \cos\phi_i \cos\phi_j \cos(\lambda_i - \lambda_j)\}. \tag{8}$$

Each bacterium in the computation represents one possible solution, i.e. one arrangement of particles. The *n*-dimensional location vector of each computational agent *i* is then $\boldsymbol{p}_i = \{(\phi_1, \lambda_1);(\phi_2, \lambda_2)...(\phi_n, \lambda_n)\}$ and the velocity vector is $\boldsymbol{v}_i = \{\theta_1, \theta_2, ..., \theta_n\}$, where θ_j is the bearing of the j^{th} particle, with $\theta_j \in [-\pi, \pi]$. The location and velocity vectors are initially chosen at random for each computational agent (we used 20 in our simulations). Note that with this definition, a tumble corresponds to all the particles in one potential solution reorienting.

At each step of the calculation, the optimum arrangement among the *k* agents (the one with the smallest potential energy) is recorded and represents the best arrangement found up to that point. Using larger values of *k* increases the probability of rapid convergence and decreases the probability of the entire system becoming stuck in local minima, but increases the running time in proportion to *k*. Table 1 presents the results of this algorithm (left column) compared with the values given by Hardin, Sloane and Smith (1996). Remarkably, 'taxis' seems to find more optimal arrangements than those previously known. Figure 3 shows the convergence of the system to these values for different numbers of particles (in all results shown, the computations were stopped after 50,000 iterations).

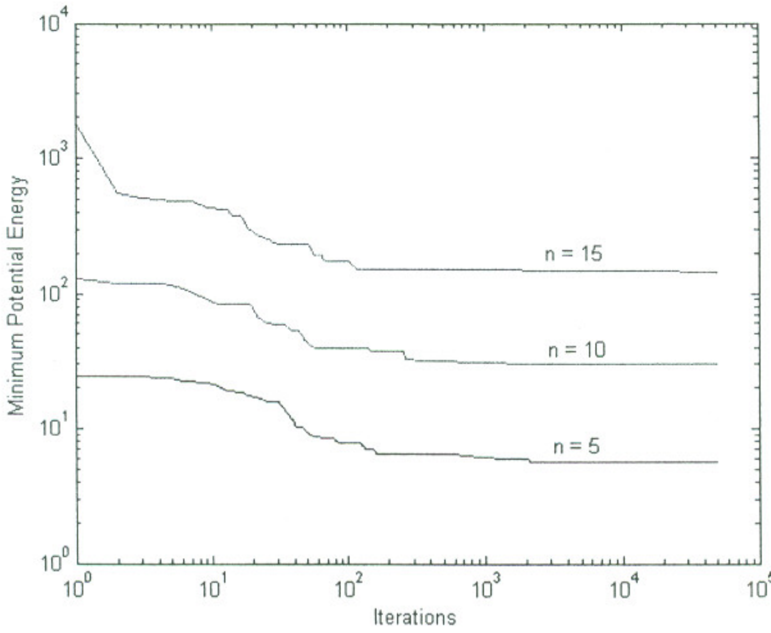

Figure 3. The convergence of a system of agents to the lowest potential-energy arrangement of n particles on a sphere.

Number of particles	Lowest Energy:	
	Hardin, Sloane & Smith, 1996	'Taxis' Algorithm
5	6.4746915	6.3395412
10	32.7169495	30.2804821
15	80.6702441	77.9961307
20	150.8815683	142.582206
25	243.8127603	238.313036
30	359.6039459	351.460535

Table 1. Lowest-energy particle arrangements found with 'taxis' compared with previously published values.

Discussion

We have so far presented three simple examples of problems that can be tackled with our method. Clearly, the performance of the algorithm, as is the case with any optimization algorithm, would depend strongly on the nature of the problem under consideration. Why might we, in general, expect computing with taxis to perform well at optimizing certain difficult functions? We can speculate on an answer to this question. Because bacteria are the oldest motile organisms and because the environments in which they live are complex at different scales of space and time, it might be expected that they be very efficient at solving optimization problems, including through chemotaxis. Recent work (Nicolau et al., 2007) suggests that run-and-tumble is evolutionarily optimal and that, remarkably, this simple algorithm can for example (as a conservative estimate) locate on average more than 92% of the total available nutrient in a Gaussian field. Furthermore, other natural algorithms and biocomputation methods such as neural networks, evolutionary computing, DNA computing and (most similar to taxis computing), particle swarm optimization have been successful. Therefore, there are general reasons to be optimistic about the potential of taxis computing for global optimisation.

The question can also be asked in the opposite direction: for what types of functions would the method be expected to perform well? Functions that are difficult to optimize because of the presence of many local extrema are good candidates because they resemble in some sense the natural environments of bacteria. If we think of the presence of many such extrema as an "apparent stochasticity" in the function (from the point of view of a particle walking the functional landscape) then we can draw an analogy between noise in biological environment and the presence of many local minima on this landscape. In other words, local fluctuations in the derivative of a function are analogous to noise in a natural environment, in this sense.

On the other hand, run-and-tumble relies on the presence of gradients to produce a drift towards favourable environments. If the functional landscape is either extremely stochastic or discontinuous, no gradient will be reliably detected and, in the limit, the method reduces to a diffusion-like random local search at a number of random points (equal to the number of bacteria in the system) on the landscape. This may not always be disadvantageous – for example, one can imagine funnel-like landscapes (similar to the postulated energy landscapes of folding protein) that possess smooth gradients on the whole but become very stochastic near the global minimum. In these cases a combination of gradient-induced global drift and noise-induced random local search may perform well. Nonetheless, taking these ideas together, we expect the type of function on which taxis computing will perform well relative to other methods to possess local gradients on scales larger than the characteristic velocity of the moving particles.

As mentioned, taxis computing bears some resemblance to particle swarm optimization. Both exhibit some attributes of evolutionary computing: each particle represents a potential solution, these solutions are initially randomly chosen, and the algorithm proceeds by evolving the solutions from iteration to iteration, with each iteration being based on the last. Of course, both methods are based on the concept of a set of particles moving through the problem space.

Two essential differences are that (a) in PSO the particles share information about the best solutions found up to each point in the computational run and (b) in PSO the velocity of each particle in the swarm is changing smoothly while in the model we propose here, the direction of each particle is constant during a run and is randomized by a tumble. The first of these is particularly essential because it means the swarm as a whole may become trapped in local minima. In PSO, at each step the velocity of particle *i* is re-evaluated according to the equation (Call et al., 2007):

$$\mathbf{V_{id}} = w\mathbf{V_{id}} + c_1 r_1 (b_p - x_{id}) + c_2 r_2 (b_i - x_{id}), \tag{9}$$

where $\boldsymbol{V_{id}}$ is the velocity of the particle, $\boldsymbol{x_{id}}$ is the position of the particle, $\boldsymbol{b_p}$ is the position of the best solution seen by the swarm as a whole and $\boldsymbol{b_i}$ is the position of the best solution found by the particle. r_1 and r_2 are random numbers and c_1, c_2 and w are positive real numbers representing the "weights" of the three terms. Because the particles (a) cooperate amongst themselves and (b) remember and factor in their best solution to date, a sufficiently good local minimum, once found, may trap the particle and in some cases the whole swarm. This cannot happen in taxis computing because these features are not present; instead, each particle relies on the structure of the local environment combined with random reorientations to explore the search space. Although it is possible (though not equally likely as in PSO) that an individual bacterium may become trapped in a local minimum for a time, this cannot happen at the level of the colony. Furthermore, because of the stochastic nature of run-and-tumble, its escape probability from this region will be non-zero and hence the residence time will be finite. Of course, the cooperation property of PSO is often valuable because the swarm as a whole can converge towards a favourable region of the problem space, which can then be searched more efficiently; nonetheless, the reinforcement of solutions already found at both swarm level and individual level means the algorithm has a higher probability of missing the global minimum of functions with properties similar to those described above.

Investigations of the performance of the discrete version of the algorithm will form the subject of future work. One can speculate, however, on the prospects of this method. On the one hand, because chemotaxis relies on the presence of gradients – in the context of computing, a direct and well-behaved relationship between position in *n*-space and fitness, we do not expect the method to perform as well or as consistently for discrete functions, for which there is little or no such correlation. For example, the difficulty in solving Boolean satisfiability stems from the property that a change in the state of one single variable (possibly among hundreds or thousands of such variables) will be the difference between the expression evaluating as TRUE or FALSE. On the other hand, a discrete version of PSO (Yang et al., 2004) has been successfully used to solve various discrete problems such as the capacitated vehicle routing problem (CVRP) (Ai-ling et al., 2006). Although in the worst case, taxis computing for discrete problems may reduce to a stochastic random search through the functional landscape (if tumble probability is uncorrelated with changes in the fitness function, or if these changes are very rare), in many cases the method may work well. This is expected to be the case, for example, when the position vector (i.e. the independent variables) is simply restricted to integer entries, as might happen for an integer optimization problem.

In the numerical results presented here, we have used a memory function of the form in Eq. 3 with $w_d \approx 3w_r$, because *E coli* compare roughly the last second of their lives with the previous 3 (Strong et al., 1998), and for simplicity. However, the form of the memory function used by a live bacterium is believed to be more complicated (Segall et al., 1986). Even more importantly, the optimal form of the memory function in the context of biocomputation is likely to be (a) different and (b) sensitive to problem or class of problems under consideration. Therefore, future work will explore different memory functions and their performances for different problems. A promising avenue is to "evolve" the memory function *in silico* for a particular class of problems. For example, in recent work we evolved the memory function of a chemotactic bacterium-like organism on a computer, in the presence of a Gaussian attractant distribution, finding that the evolved function resembles the biphasic shape believed to be at work in the chemotactic mechanism of *E coli*. Presumably, when exposed to different functional landscapes, a "species" of digital organisms equipped with the ability to evolve the memory function will adapt to the function in question, developing an optimal or near-optimal response.

It was mentioned above that in PSO, the swarm converges to the best solution found to date and that this strategy, while running the risk of missing the global optimum, means that local searches near the best solution found are more efficient – because they are carried out by more particles. In an attempt to introduce this feature into our model by mimicking the natural behaviour of bacteria, one possible variation on the algorithm presented above would also allow the agents to divide (produce offspring) when in a favourable region of the functional landscape. This would maintain the advantage of not swarming to a local minimum while increasing the efficiency of local searches (and, if the agents can also die, reducing the proportion of computational time dedicated to unpromising regions). Finally, future work will also focus on comparing this method with other methods, both of a natural computing flavour and also more classical methods such as steepest descent, random search etc.

Acknowledgements

D.V.N. would like the acknowledge financial support from the Clarendon Fund, the UK Overseas Research Student Award Scheme and the Devorguilla Scholarship from Balliol College, University of Oxford. The authors would like to thank Kevin Burrage for suggestions that have improved the manuscript.

References

1. Adleman, L.M., (1994), "Molecular Computation Of Solutions To Combinatorial Problems", Science 266(11): 1021–1024.
2. Ai-ling, C., Gen-ke, Y., Zhi-ming, W., (2006), Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem, J Zhejiang Univ Science A, 7(4): 607-614.
1. Basheer, I.A., Hajmeer, M., (2000), "Artificial neural networks: fundamentals, computing, design, and application", J Microbiol Meth 43 (1): 3-31
2. Bray, D., (1995), "Protein molecules as computational elements in living cells", Nature 376: 307 – 312.
3. H. J. Bremermann, (1974), "Chemotaxis and optimization", J Franklin Inst, 297: 397–404.
4. Call, S.T., Zubarev, D.Y., Boldyrev, A.I., (2007), "Global Minimum Structure Searches via Particle Swarm Optimization", J Comput Chem 28: 1177-1186.
5. Dorigo M., Blum C., (2005), "Ant colony optimization theory: A survey", Theor Comp Sci 344(2-3): 243-278.
6. Eiben, A.E., Smith, J.E., (2003), "Introduction to Evolutionary Computing", New York: Springer.
7. Hardin, R.H., Sloane, N.J.A., Smith, W. D., (1997), "Spherical coverings" (in preparation), Available at http://www.research.att.com/njas/coverings/index.html.
8. Locsei, J.T., (2007), "Persistence of direction increases the drift velocity of run and tumble chemotaxis", J Math Biol, in press
9. Muller, S., Marchetto, J., Airaghi, S., Koumoutsakos, P., (2002), "Optimization based on Bacterial Chemotaxis", IEEE Trans Evol Comput 6(1): 16-29.
10. Nicolau, D.V., Armitage, J.P., Maini, P.K., (2007), "In Silico Evolution of Chemotactic Swimming", in submission.
11. Nicolau, D.V., Nicolau, D.V., (2006), "Biocomputation", in the Wiley Encyclopaedia of Biomedical Engineering, New York: Wiley.
12. Obayashi, S., (1997), "Pareto genetic algorithm for aerodynamic design using the Navier-Stokes equations," in Genetic Algorithms in Engineering and Computer Science, New York: Wiley.
13. Segall, J.E., Block, S.M., Berg, H.C., (1986), "Temporal comparisons in bacterial chemotaxis", Proc Natl Acad Sci U S A. 83(23): 8987-8991.

14. Strang, G., (2005), “The SIAM 100-digit challenge - A study in high-accuracy”, Science 307(5709): 521-522.
15. Strong, S. P., Freedman, B., Bialek, W. & Koberle, R., (1998), “Adaptation and optimal chemotactic strategy for *E. coli*”, Phys Rev E 57: 4604-4617.
16. Trefethen, N., (2002), "A Hundred-Dollar, Hundred-Digit Challenge.", SIAM News 35(1).
17. Vergassola, M., Villermaux, E., Shraiman, B.I., (2007), “'Infotaxis' as a strategy for searching without gradients”, Nature 445: 406-409.
18. Wadhams, G.H., Armitage, J.P., (2004), “Making sense of it all: bacterial chemotaxis”, Nat Rev Mol Cell Biol 5:1024-1037.

The Information Processing in Cells and Tissues (IPCAT) workshop series brings together a multidisciplinary group of scientists working in fields related to modelling information processing in biosystems. The workshops are concerned with the nature of biological information and the ways in which it is processed in both biological and artificial cells and tissues.

A key motivation is to provide common ground for a dialogue between scientists from a range of disciplines, without emphasis on any particular views of biological information processing or approaches to modelling.

IPCAT2007 has build on the previous biennial IPCAT workshops (initiated in 1995) to highlight the most recent developments in research in information processing in cells and tissues. The workshop has featured the presentation of refereed original papers grouped around emergent themes of common interest.

- Automata and cellular automata
- Enzyme and gene networks
- Evolving, adapting, and neural hardware
- Evolutionary algorithms
- Information processing in bio-developmental systems
- Information processing in neural and non-neural biosystems
- Machine learning
- Modelling of metabolic pathways and responses
- Novel bio-information processing systems
- Self-organising, self-repairing, and self-replicating systems
- Simulation of genetic and ecological systems

ISBN 978-80-87139-68-4

9 788087 139684 >