Optimization by Bayesian adaptive locally linear stochastic descent

Cliff C. Kerr[1,2,3,4]*, **Tomasz G. Smolinski**[5], **Salvador Dura-Bernal**[4], **David P. Wilson**[1]

[1] Kirby Institute for Infection and Immunity in Society, University of New South Wales, Sydney, NSW, Australia
[2] Complex Systems Group, School of Physics, University of Sydney, Sydney, NSW, Australia
[3] Centre of Excellence for Integrative Brain Function, University of Sydney, Sydney, NSW, Australia
[4] Department of Physiology and Pharmacology, SUNY Downstate Medical Center, Brooklyn, NY, USA
[5] Department of Computer and Information Sciences, Delaware State University, Dover, DE

Abstract

When standard optimization methods fail to find a satisfactory solution for a parameter fitting problem, a tempting recourse is to adjust parameters manually. While tedious, this approach can be surprisingly powerful in terms of achieving optimal or near-optimal solutions. This paper outlines an optimization algorithm, Bayesian Adaptive Locally Linear Stochastic Descent (BALLSD), that has been designed to replicate the essential aspects of manual parameter fitting in an automated way. Specifically, BALLSD uses simple Bayesian principles to form probabilistic assumptions about (a) which parameters have the greatest effect on the objective function, and (b) optimal step sizes for each parameter. We show that for a certain class of optimization problems (namely, those with a moderate to large number of scalar parameter dimensions, especially if some dimensions are more important than others), BALLSD is capable of minimizing the objective function with far fewer function evaluations than classic optimization methods, such as the Nelder-Mead nonlinear simplex, Levenberg-Marquardt gradient descent, simulated annealing, and genetic algorithms.

1 Introduction

Consider a human $\mathcal{H}$ who is attempting to minimize a nonlinear objective function, $E = f(\mathbf{x})$, by manually adjusting parameters in the vector $\mathbf{x}$. $\mathcal{H}$ typically begins with a uniform prior regarding which parameters to vary, and chooses step sizes that are a fixed fraction (*e.g.*, 10%) of the initial parameter values. $\mathcal{H}$ will then pseudorandomly choose one or more parameters to adjust. Every time a parameter x_i is found to reduce E, the probability that $\mathcal{H}$ will select x_i in the future increases; conversely, if changes in x_i are not found to improve E, the probability that $\mathcal{H}$ will select x_i decreases (formally, $\mathcal{H}$ forms "hunches" about which parameters are "good"). $\mathcal{H}$ also adaptively adjusts the step size based on the information $\mathcal{H}$ obtains about the curvature of parameter space with respect to each dimension (*e.g.*, if $\Delta E/\Delta x_i \approx$ const. over multiple iterations, $\mathcal{H}$ will increase the step size). Despite its drawbacks, the adaptive nature of manual parameter fitting makes it a remarkably powerful method.

Thus, despite the smörgåsbord of available automated optimization algorithms, manual fitting of parameters remains a familiar bane of researchers (*e.g.*, Castillo et al. (2005); Brom et al. (2012)), especially in cases where evaluations of the objective function are computationally intensive, such as climate models (Wilby, 2005), neuronal network models (Prinz et al., 2003; Baker et al., 2011; Kerr et al., 2013), or detailed epidemiological models (Kwon et al., 2012). However, it is difficult to estimate how commonly manual parameter fitting is performed, since authors often do not explicitly mention its use (*e.g.*, Song et al. (2013)).

In many types of optimization problems, it is more important to need only a small number of function evaluations to find a reasonable local minimum than it is to find the global minimum (Goodner et al., 2012). Indeed, the latter may be ill-defined given the large uncertainties that are often present when models of complex systems are fitted to empirical data, as in the citations listed above.

With the increasing availability of high-performance computers and clusters (Hilbert and López, 2011), easily parallelizable optimization methods such as evolutionary algorithms (where different individuals can be run on different cores) and Monte Carlo methods (where different initializations can be run on different cores) have a notable advantage for certain types of problems. The common theme in these algorithms is the ability to use a different random seed for each parallel instance. However, as the size of parameter space increases, the advantage of this approach is diluted: whereas a 3- or even 5-dimensional parameter space may be reasonably densely sampled by a Monte Carlo initialization, a 20- or 100-dimensional

* Corresponding author:
Address: 450 Clarkson Ave., Brooklyn, NY 11203, USA
Phone: +1 (347) 721-7328
Email: cliff@thekerrlab.com

space cannot. This is because parameter space grows exponentially with an increasing number of dimensions, whereas parallelization increases sampling rates linearly.

In high-dimensional parameter spaces, it is unlikely that all parameters contribute equally to the objective function. Identifying those that contribute more, thereby allowing computational resources to be focused on them, has the potential to significantly reduce the total number of function evaluations required. Despite humans' limited capacity to implement Bayesian-optimal strategies (Charness et al., 2007; Steyvers et al., 2009), we speculate that this adaptive approach to both parameter selection and step size is the key reason why manual parameter fitting can be highly effective.

In this paper, we present an algorithm, Bayesian Adaptive Locally Linear Stochastic Descent (BALLSD), that was inspired by manual parameter fitting. This approach is most applicable to optimization problems with more than approximately 5 dimensions – *i.e.*, large enough so that performing function evaluations across all dimensions is inefficient.

2 Methods

Consider an objective function $E = f(\mathbf{x})$, where E is the scalar error (or other quantity) to be minimized (or maximized) and $\mathbf{x} = [x_1, x_2, ..., x_n]$ is an n-element vector of parameters. There are $2n$ possible directions j to step in: an increase or decrease in the value of each parameter. Associated with each parameter x_i are (a) two initial step sizes: $s_j = s_i^+$ or s_i^-, which define the step size in the directions of increasing or decreasing x_i, respectively (*i.e.*, $s_i^+ > 0$ and $s_i^- < 0$); and (b) two initial probabilities: $p_j = p_i^+$ or p_i^-, which define the likelihood of selecting direction j (for a uniform prior, $p_j = 1/2n$ – satisfying the requirement that $\sum \mathbf{p} = \sum_{j=1}^{2n} p_j = 1$). Thus, the vectors $\mathbf{s}$ and $\mathbf{p}$ have length $2n$.

At each step k, the algorithm maps a random variable $\alpha \in (0, 1)$ onto $\mathbf{p}$, thereby choosing a direction $j \in (1 \dots 2n)$ and a corresponding parameter $i = \lceil j/2 \rceil \in (1 \dots n)$, where $\lceil \cdot \rceil$ denotes the ceiling operator. The algorithm then evaluates

$$E_k^{\pm} = f(\mathbf{x} + \delta(i)), \tag{1}$$

where $\delta(i)$ is an n-element vector such that $\delta_i = s_j$ and 0 otherwise. Then:

1) If $E_k^{\pm} < E_{k-1}$:
   a) The new parameter value is adopted: $x_i \rightarrow x_i + s_j$;
   b) The error is updated: $E_k \rightarrow E_k^{\pm}$;
   c) s_j is increased: $s_j \rightarrow s_j \cdot s_{inc}$ ($s_{inc} > 1$);
   d) p_j is increased: $p_j \rightarrow p_j \cdot p_{inc}$ ($p_{inc} > 1$), and $\mathbf{p}$ is renormalized such that $\sum \mathbf{p} = 1$.
2) Otherwise:
   a) The parameter vector $\mathbf{x}$ and error E are not changed;
   b) s_j is decreased: $s_j \rightarrow s_j / s_{dec}$ ($s_{dec} > 1$);
   c) p_j is decreased: $p_j \rightarrow p_j / p_{dec}$ ($p_{dec} > 1$), and $\mathbf{p}$ is renormalized as above.

The algorithm thus has four meta-parameters: s_{inc}, s_{dec}, p_{inc}, and p_{dec}. In practice, a reasonable choice is $s_{inc} = s_{dec} = p_{inc} = p_{dec} = 2$, although in our experience any value from approximately 1.2 to 3 appears to work reasonably well for the test cases used here; in general, the smoother and more linear the objective function is, the larger the learning rates should be. In addition to these meta-parameters, four initial value vectors need to be specified: the initial parameter vector $\mathbf{x_0}$, step sizes $\mathbf{s}$ (which in general can be initialized as a fixed fraction of the corresponding initial parameter value, unless it is zero), and probabilities $\mathbf{p}$ (where typically $p_j = 1/2n$ suffices for an n-parameter problem).

By modifying $\mathbf{s}$ and $\mathbf{p}$ after each iteration, the algorithm learns which directions are most effective to step in and by how much, using a loosely Bayesian approach (in the sense that it estimates the posterior distributions of $\mathbf{s}$ and $\mathbf{p}$ by modifying the priors according to accumulated evidence). This, combined with the stochastic choice of which parameters to modify on each iteration, resembles the way in which humans (imperfectly) perform Bayesian decision-making in situations such as N-armed bandit problems (Steyvers et al., 2009).

3 Results

3.1 Comparison to other optimization methods

Here we compare BALLSD to four standard optimization methods: the Nelder-Mead nonlinear simplex algorithm (Nelder and Mead, 1965), Levenberg-Marquardt gradient descent (Marquardt, 1963), simulated annealing (Kirkpatrick et al., 1983), and a genetic algorithm (Bethke, 1978). All methods were implemented in MATLAB 2012b (The Mathworks), via the Optimization Toolbox functions "fminsearch", "lsqnonlin", "simulannealbnd", and "ga", respectively.

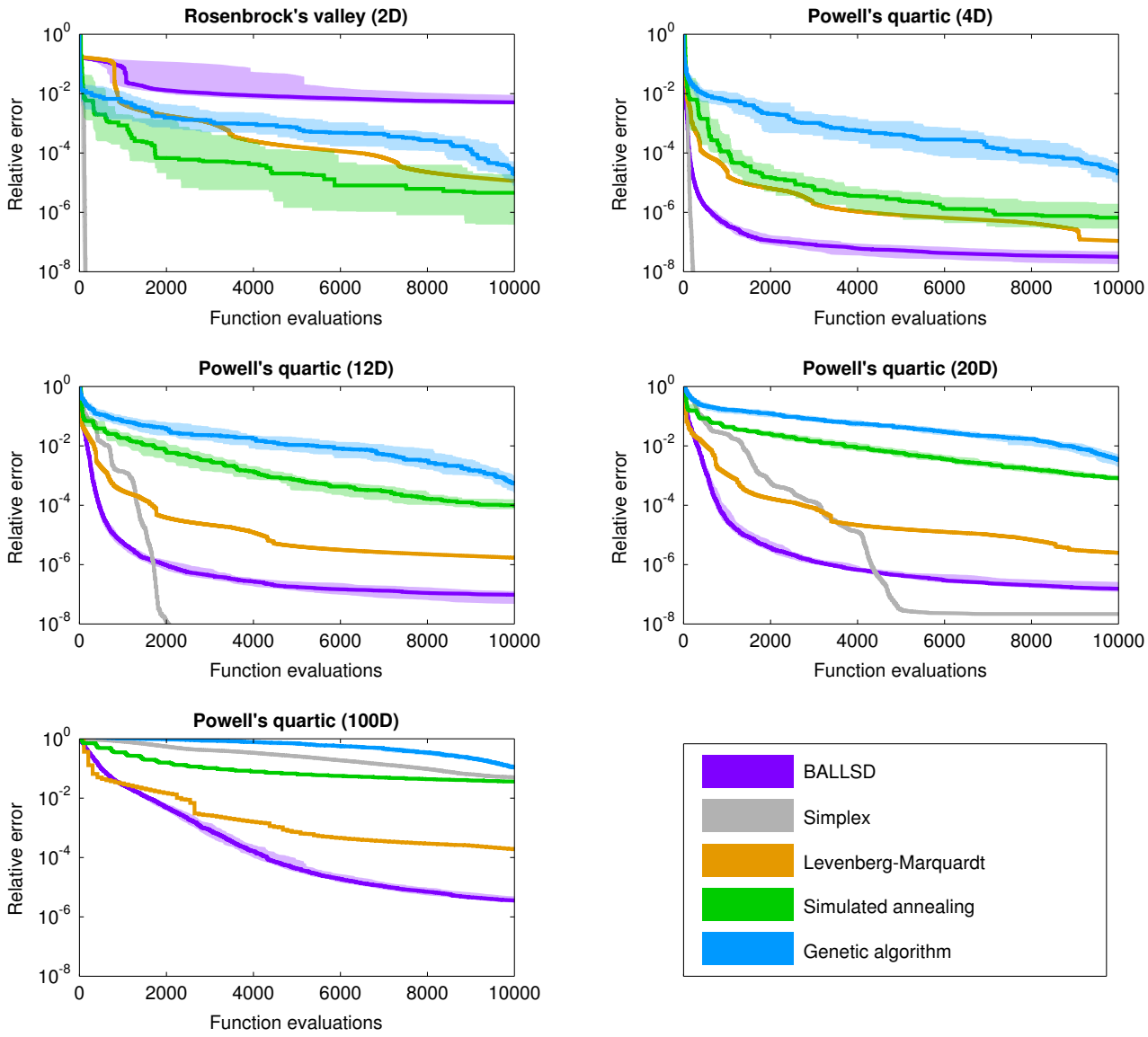

Figure 1: Performance of BALLSD compared to four standard nonlinear optimization algorithms: Nelder-Mead nonlinear simplex, Levenberg-Marquardt gradient descent, simulated annealing, and a genetic algorithm. While standard methods – especially the simplex method – are most efficient for low-dimensional problems (*e.g.*, Rosenbrock's valley), in many cases BALLSD is the most efficient algorithm for high-dimensional parameter spaces (*e.g.*, the 100-dimensional version of Powell's quartic function). For the stochastic methods (BALLSD, simulated annealing, and the genetic algorithm), the shaded regions show the interquartile range for 40 different random seeds.

For BALLSD, we used meta-parameters $s_{inc} = p_{inc} = s_{dec} = p_{dec} = 2$, initial step sizes s_j of 20% of the parameter values in $\mathbf{x_0}$ (which are given below; the step size for any parameter with an initial value of 0 is the mean of the other step sizes), and initial probabilities p_j of $1/2n$ for an n-dimensional problem. MATLAB's default meta-parameters were used for the other four algorithms, except that the initial temperature of the simulated annealing algorithm was set to be equal to $10 \cdot \langle|\mathbf{x_0}|\rangle$ following manual exploration of meta-parameter space, since the default choice of 100 did not generalize well across problems of different scales. Indeed, one of the major disadvantages of this type of algorithm is its sensitivity to the values of its meta-parameters (Ben-Ameur, 2004).

To test this suite of algorithms, we used original and modified versions of two classic optimization problems given in Nelder and Mead (1965):

1) Rosenbrock's parabolic valley (two-dimensional):

$$E = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \tag{2}$$

with the starting point at $\mathbf{x} = (-1.2, 1)$. The optimum is at $\mathbf{x} = (1, 1)$.

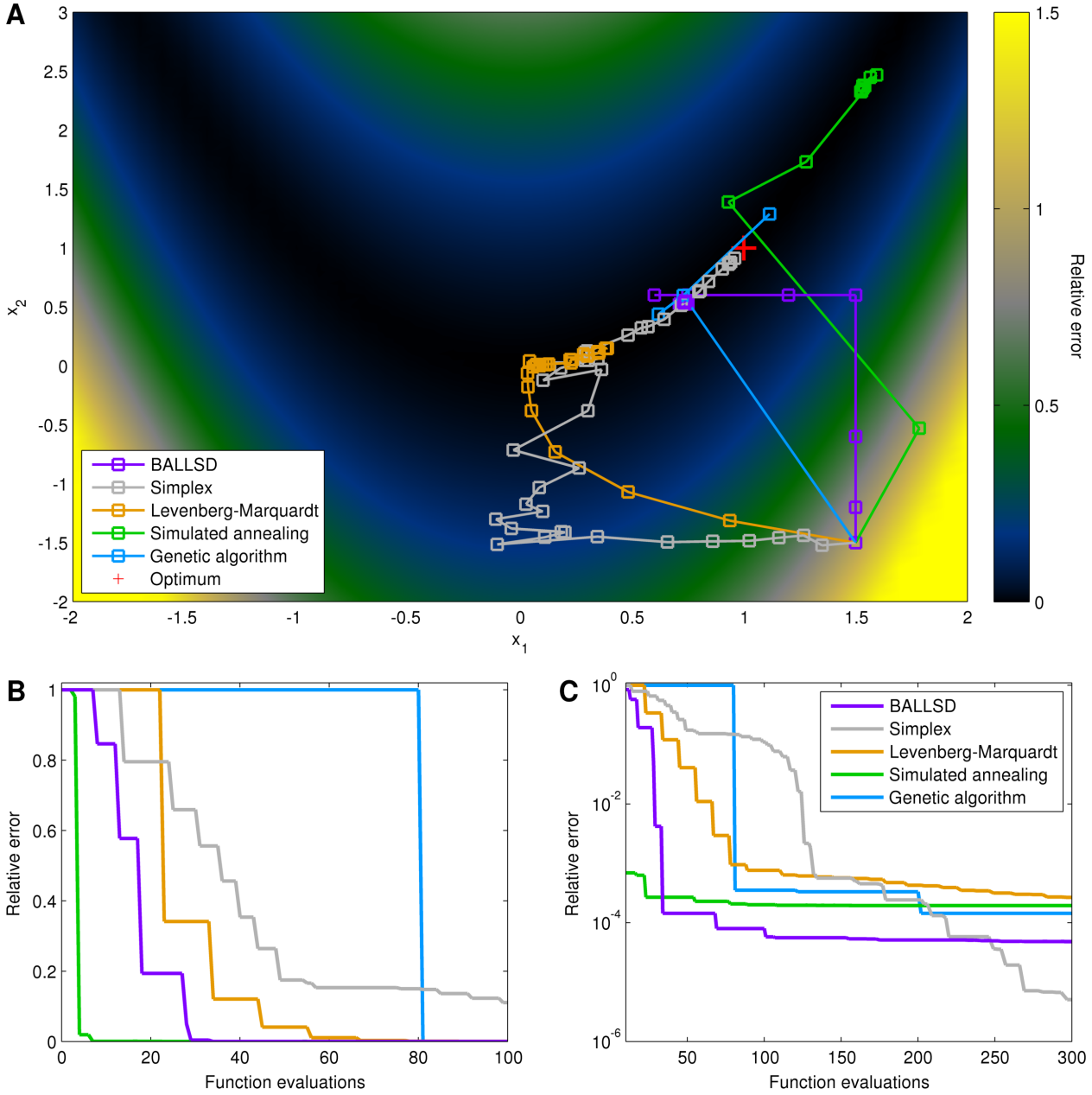

FIGURE 2: Optimization of the 10-dimensional version of Rosenbrock's valley starting from the point $(1.5, -1.5)$. **(A)** Trajectories of each optimization method up to 300 function evaluations; each iteration is shown with a square, but note that multiple function evaluations may occur at each iteration. Note the locally linear steps of BALLSD that rapidly adapt in size. **(B)** Relative error of each method for the first 100 function evaluations, showing the initial stage of the algorithms. **(C)** Relative error for the first 300 function evaluations, showing the asymptotic stage of the algorithms.

2) A modified 10-dimensional version of Rosenbrock's valley, with the functional form as given in Eq. 2, but with a 10-element parameter vector $\mathbf{x}$; the remaining 8 parameters do not contribute to the objective function. The starting point is at $\mathbf{x} = (1.5, -1.5, 0, 0 \ldots 0)$. The optimum is at $\mathbf{x} = (0, 0 \ldots 0)$.
3) A modified Powell's quartic function (N-dimensional):

$$E = \sum \left((\mathbf{x_1} + 10\mathbf{x_2})^2 + 5(\mathbf{x_3} - \mathbf{x_4})^2 + (\mathbf{x_2} - 2\mathbf{x_3})^4 + 10(\mathbf{x_1} - \mathbf{x_4})^4 \right), \tag{3}$$

where $\mathbf{x}_q$ is a vector of length $N/4$. The starting point is at $\mathbf{x_{1_0}} = (\overline{3})$, $\mathbf{x_{2_0}} = (\overline{-1})$, $\mathbf{x_{3_0}} = (\overline{0})$, and $\mathbf{x_{4_0}} = (\overline{1})$, where each component is repeated $N/4$ times (*e.g.*, if $N = 8$, $\mathbf{x_0} = (3, 3, -1, -, 1, 0, 0, 1, 1)$). The optimum is at $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}) = (0, 0, 0, \ldots 0)$. Here, we used 4, 12, 20, and 100-dimensional versions of Powell's function.

As shown in Fig. 1, for the two-dimensional optimization problem, the nonlinear simplex method is most efficient, with all other algorithms requiring considerably more function evaluations to obtain the same error. Notably, BALLSD was

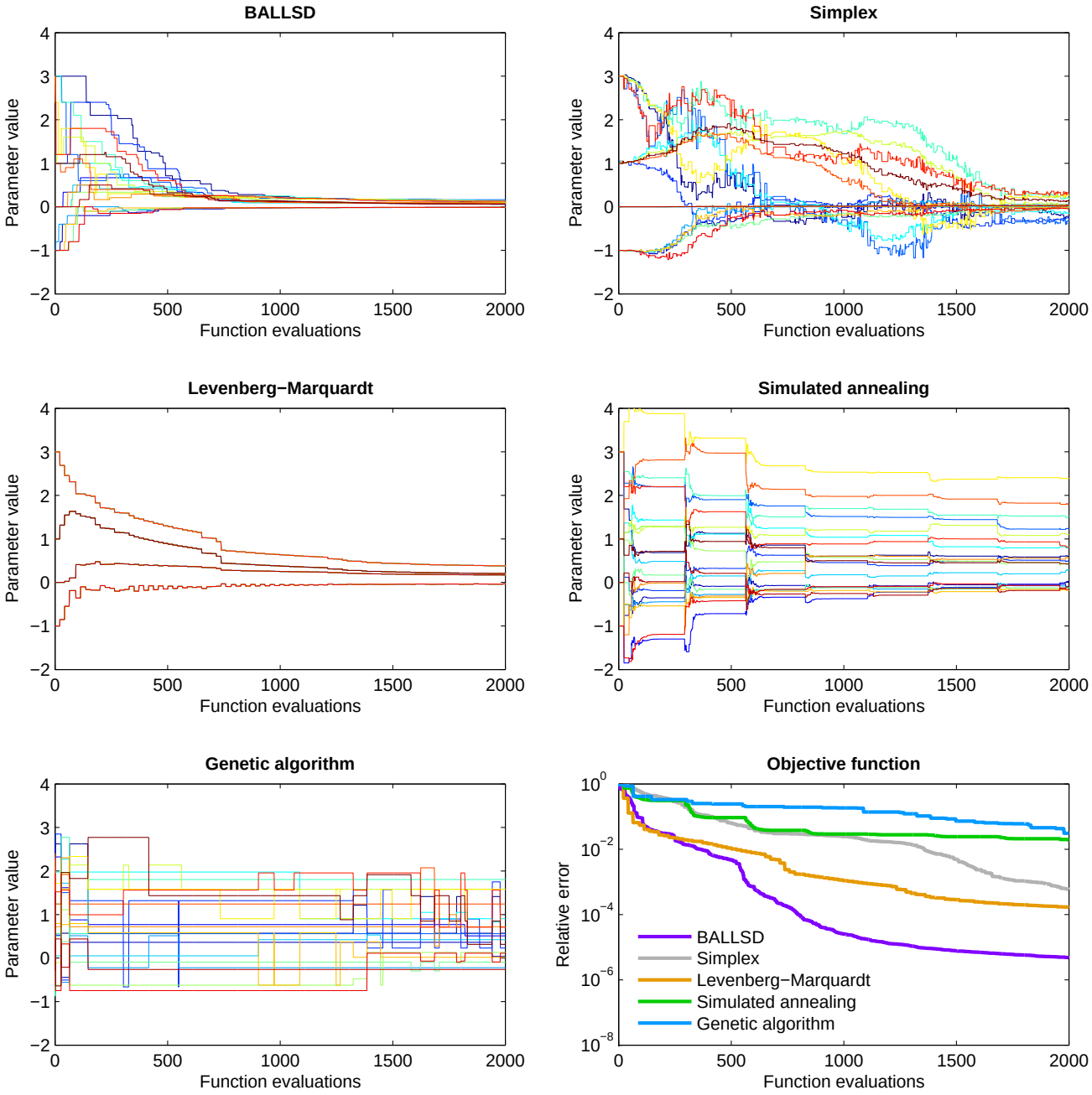

Figure 3: Parameter update strategies for each algorithm applied to a 20-dimensional Powell's quartic function. Each line is a separate parameter; the optimum is at $(0, 0, 0, \ldots 0)$. The error relative to the starting point is shown in the bottom right panel. For small numbers of iterations (the adaptive phase of BALLSD), the Levenberg-Marquardt method reduces error most quickly; for larger numbers of iterations, BALLSD achieves 1–4 orders of magnitude smaller error for a given number of iterations than the other methods. (Note: since the genetic algorithm does not use a single initial point, individuals were instead initialized using a uniform random distribution in the range $[-1, 3]$. The Levenberg-Marquardt algorithm operates on the 20-dimension Powell's function identically to the 4-dimensional version, with the exception that each iteration requires 5 times as many function evaluations.)

especially *inefficient*, since its assumption of local linearity is violated by the shallow, curved valley. However, with the modified version of the problem described above, BALLSD is the most efficient algorithm over most of the first several hundred function evaluations, as shown in Fig. 2 for a single random seed. For small numbers of iterations (<30), for this particular seed, simulated annealing was by far the most efficient algorithm, reducing the error by a remarkable 98% after just 4 function evaluations. However, this algorithm became mired near the point $(1.5, 2.4)$, far from the true minimum of $(1, 1)$, and did not significantly reduce the error beyond the first 20 function evaluations. After 50 function evaluations, BALLSD had reduced the error by 99.9%, compared to 99.7% for simulated annealing, 96% for the Levenberg-Marquardt

method, 82% for the nonlinear simplex method, and 0% for the genetic algorithm. Similarly, BALLSD reduced the error by 99.99% after 70 function evaluations; in comparison, the next best algorithm (the simplex method) required 220 function evaluations to reach the same error level. During the descent into the shallow curved valley (comprising ∼99.9% of the total error), the most efficient algorithms were BALLSD and simulated annealing; within the valley (the remaining ∼0.1% of the total error), the simplex algorithm was by far the most efficient. Hence, this example illustrates that in optimization problems where some parameters are significantly more important than others, BALLSD has significant advantages. In contrast, for problems in which all parameters have equal importance, as in the original Rosenbrock's valley problem, other algorithms have superior performance.

For the 4-dimensional Powell's quartic function, the nonlinear simplex method was again the most efficient, followed by BALLSD. For the 12- and 20-dimensional version, BALLSD was most efficient for 60–1700 and 250–4400 function evaluations respectively (corresponding to roughly 99.9999% of the total error at the upper limit in each case), after which the simplex method was most efficient. For the 100-dimensional version, the Levenberg-Marquardt method was most efficient for the first 1000 function evaluations (corresponding to 97% of the total error), but BALLSD was the most efficient algorithm for larger numbers of function evaluations.

The five optimization methods discussed here employ very different parameter update strategies, as shown strikingly in Fig. 3. The approach used in BALLSD is most similar to the Levenberg-Marquardt method, with the exception that the rate of convergence of the former increases over time (due to its adaptive step size), whereas for the latter it decreases. In the example shown here (a 20-dimensional Powell's quartic function), the Levenberg-Marquardt method has the lowest error for 250 or fewer iterations; for large numbers of iterations, BALLSD has by far the lowest error – indeed, for 2000 or more iterations, it has nearly 2 orders of magnitude less error than the Levenberg-Marquardt method, and 4 orders of magnitude less error than nonlinear simplex, simulated annealing, and genetic algorithms. The superior performance of BALLSD compared to the other methods is surprising since, unlike in Fig. 2, in this problem all parameters are of roughly equal importance, so the adaptive probability $\mathbf{p}$ is unlikely to significantly contribute to the efficiency of the optimization. Thus, even in cases where BALLSD's only advantage is its adaptive step size, it is still capable of outperforming traditional algorithms.

3.2 Optimizing HIV resource allocations

In contrast to the foregoing theoretical discussion of error minimization for analytical functions, here we describe the practical application that BALLSD was designed for, a topic that has recently garnered considerable interest: finding the allocation of resources across different HIV prevention programs that minimizes new infections (Anderson et al., 2014). Full details of the HIV model are presented in Supplementary Methods S1. In brief, the model describes HIV transmission and progression in a number of interacting subpopulations (14 in this case), including female sex workers, men who have sex with men, and general males and females in different age groups. The model incorporates roughly 200 parameters, which describe the sexual behavior, injecting behavior, HIV testing and treatment rates, and sexual and injecting partnerships of each population, as well as basic clinical parameters such as HIV transmissibility and disease progression rates. In addition to empirical estimates of these parameters, the model is calibrated to match surveillance data on HIV prevalence, diagnoses, and numbers of people on treatment. In this example, the model was based on and calibrated to behavioral and surveillance data from Swaziland. Since the model is relatively computationally intensive, requiring 1–2 s per function evaluation on a standard laptop, large numbers ($>10^2$) of evaluations are wearisome.

To optimize the allocation of Swaziland's HIV budget, we assumed that spending on particular HIV programs produces changes in corresponding behavioral parameters or testing and treatment rates (for example, programs targeting female sex workers increase their probability of condom use). In this case, the objective we chose to minimize was the number of new infections over the period 2015–2020, subject to the constraint that total funding was held constant at current (2014) levels. Allocations across 9 different HIV prevention, testing, and treatment programs were considered.

As shown in Fig. 4A, under current conditions, the model predicts a median of approximately 2500 new infections per year in Swaziland. However, if funding is optimally allocated, as shown in Fig. 4B (which consists largely of shifting funds from programs for orphans and vulnerable children towards treatment and male circumcision programs), this can be reduced to approximately 1260 new infections per year. BALLSD found this allocation after 65 function evaluations. The next-best algorithm, the Levenberg-Marquardt method, found a nearly identical allocation after 830 function evaluations. None of the other methods reached this level of optimization within 2000 function evaluations; by that point, the genetic algorithm had achieved 99.3% of the reduction in new infections found by BALLSD and the Levenberg-Marquardt method, the nonlinear simplex algorithm 95%, and the simulated annealing algorithm 90%.

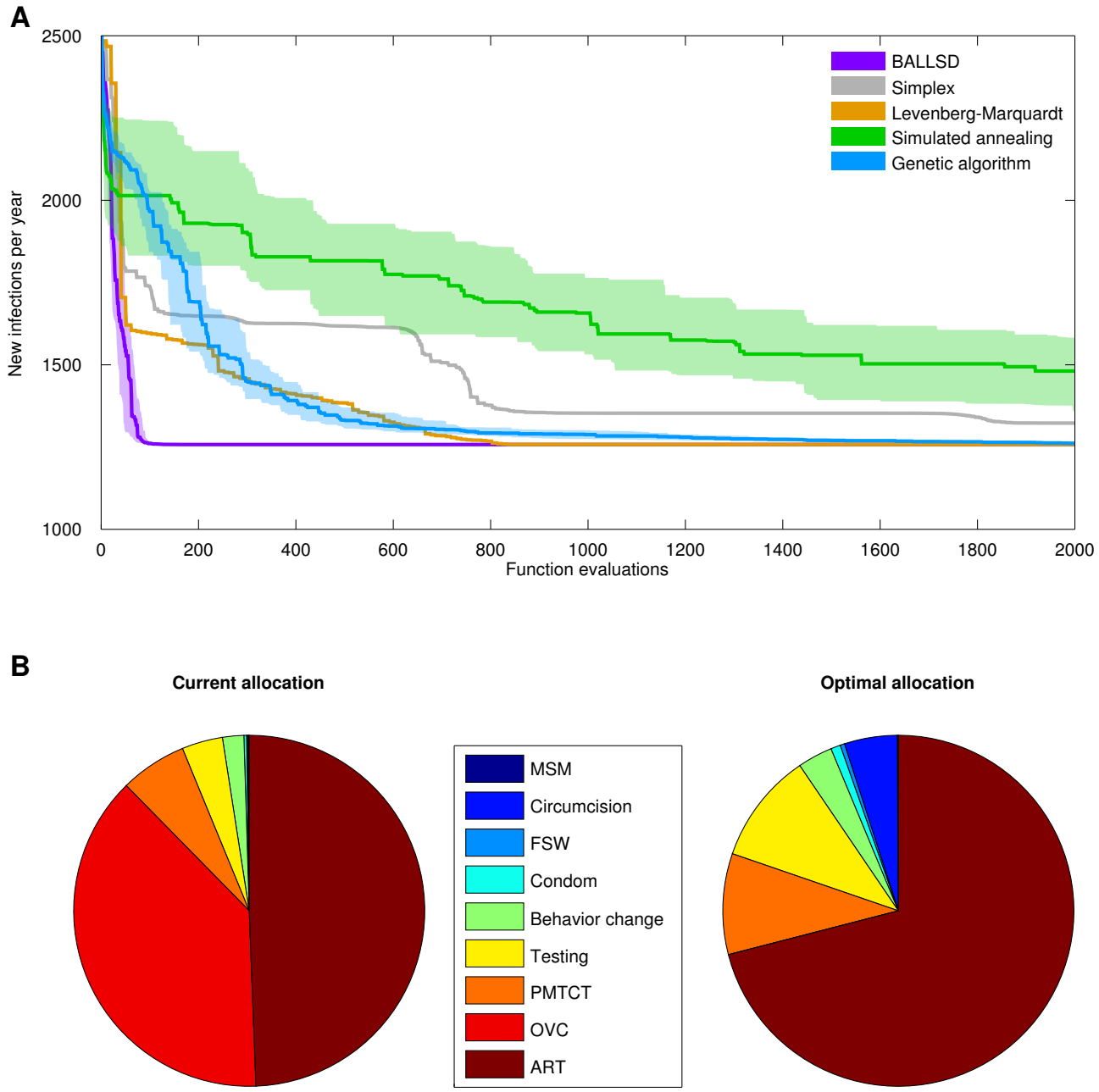

Figure 4: Comparison of optimization methods for a real-world example of HIV resource allocation. **(A)** The minimum number of new infections calculated by each method for the first 2000 function evaluations. As above, the shaded regions show the interquartile ranges over 40 different random seeds. **(B)** Resource allocations in Swaziland for each HIV program, showing the current budget allocation (left) and the allocation that minimizes new infections (right). MSM = men who have sex with men; FSW = female sex workers; PMTCT = prevention of mother-to-child transmission; OVC = orphans and vulnerable children; ART = antiretroviral therapy.

4 Discussion

4.1 Extensions of the algorithm

This section describes several modifications to the basic algorithm that may make it more suitable for a broader range of optimization problems.

First, the assumptions of uniform priors $\mathbf{p}$ and uniform initial step sizes $\mathbf{s}$ can easily be relaxed, allowing assumptions about the scale or relative importance of parameters to be incorporated. However, due to the adaptive nature of the algorithm, even silly initial choices of $\mathbf{p}$ and $\mathbf{s}$ will be corrected, as long as all p_j and s_j are nonzero.

Second, rather than updating the probability p_j by a fixed amount after each successful iteration, the change in p_j (Δp_j) could be proportional to the change in the error function E (ΔE), with a larger ΔE resulting in larger Δp_j. Since the expected change in E at step k is proportional to both $E_k - \min(E)$ and the ratio of the step size to the characteristic scale of each parameter, and since in general neither of these quantities are known, the constant of proportionality between Δp_j and ΔE cannot typically be estimated *a priori*. One can partially circumvent this problem by comparing the current ΔE to its previous values; however, more weight would need to be given to more recent values, since ΔE tends to decrease as the algorithm converges on a solution.

Third, it is possible to relax the assumption of local linearity by varying multiple parameters on a single iteration. However, assuming a separate probability is stored for each parameter combination, this reduces the learning rate; for an n-parameter problem, modifying a single parameter at each iteration results in a learning rate of $1/2n$ on average for each parameter; in the limit where all possible combinations of parameters are considered, the learning rate would be $1/2^{2n}$. While manageable for small numbers of parameters (*e.g.*, $\leq$4), this quickly becomes intractable as the number of parameters grows. Conversely, if multiple parameters are modified simultaneously, the probabilities of all modified parameters could be updated simultaneously; this approach is likely to be most effective in very high-dimensional systems where the function E is nearly flat with respect to many of the dimensions, in which case varying parameters one by one may be time-consuming. The superior performance of simulated annealing compared to BALLSD for small numbers of function evaluations in the 10-parameter Rosenbrock's valley problem shown in Fig. 2 is likely due to this effect.

Fourth, to circumvent the problem of local minima, the method may be used with either Monte Carlo initialization (Metropolis and Ulam, 1949) or a Metropolis-Hastings approach (Metropolis et al., 1953). In the former, the BALLSD algorithm would be repeated multiple times (typically, $10^2 - 10^3$) with pseudorandom choices of $\mathbf{x_0}$. In the latter, instead of always performing step 2 of the algorithm if the new iteration does not reduce error, step 1 would be performed with nonzero acceptance ratio ρ, where ρ is a function of the change in error; *e.g.*, $\rho \propto E_k^{\pm}/E_{k-1}$. Although the parameter set resulting from each iteration can be kept, as in a standard Metropolis-Hastings algorithm, the value of doing so is limited since the asymptotic distribution of parameter sets is not guaranteed to reach a stationary distribution, due to the adaptive method for choosing which parameters to vary. Instead, it would suffice to keep two parameter sets, the current one and the best one. As a simpler alternative to implementing a Metropolis-Hastings approach, rather than always reducing the step size if the new iteration does not reduce the error, step size could have a nonzero probability of increasing, potentially allowing the algorithm to escape local minima.

Finally, the BALLSD algorithm is only loosely Bayesian. While a more formal Bayesian approach may be desirable in certain situations, in general it is difficult to determine whether new information should be used to update the existing distribution, or whether the system is in a sufficiently dissimilar part of the parameter space that information from much earlier iterations is no longer relevant. Nonetheless, for certain problems, additional capacity for adaptation may be beneficial. For example, the basic implementation of BALLSD described above performs poorly in the classic version of Rosenbrock's valley (Fig. 1); for this particular problem, an algorithm that was capable of learning nonlinear parameter combinations would be far more efficient.

4.2 Conclusions

This paper has presented a simple optimization method inspired by the process of manual parameter fitting that is capable of outperforming traditional algorithms for certain classes of problems. The algorithm is most effective for problems with moderate to large dimensionality (>5 dimensions), which corresponds to the case in which there are enough parameters that different parameters are likely to have very different overall contributions to the objective function. Indeed, the relative uniformity of parameters in the test functions used here (in terms of both scale and effectiveness) does not necessarily reflect certain real-world situations in which some – or even most – of the objective function's parameters may have little influence on its value. In such situations, BALLSD is especially effective, as it is able to adapt to those parameters (and those scales) that produce the greatest improvements in the objective function. An example of this is provided in Fig. 4, where BALLSD finds what appears to be the globally optimal solution more than 10 times faster than any other algorithm.

This study has two main limitations. First, MATLAB's default values of the meta-parameters were used for the other algorithms (except the initial temperature of the simulated annealing, as noted above). Meta-parameter tuning would likely increase the performance of these algorithms more than it would for BALLSD, since these algorithms are not adaptive – but conversely, an advantage of BALLSD is that it typically does not require any meta-parameter tuning, so in that sense the comparison is fair. Second, we chose the four algorithms to compare against BALLSD based on their popularity, as evidenced by their inclusion in MATLAB's Optimization Toolbox. However, many other optimization algorithms exist, some of which have also been shown to significantly outperform these more traditional methods (*e.g.*, Rios and Sahinidis (2013)).

As noted above, BALLSD has already been used successfully in the real-world application of optimizing the allocation of HIV budgets. For this problem, standard optimization methods, including the four compared against BALLSD in this paper, were found to require an excessively large number of function evaluations to obtain acceptable solutions. This led the authors to resort to manual parameter fitting until BALLSD was developed. It is our hope that this algorithm may be able to free other researchers from similar unpleasantries. Finally, we note that Python and Matlab implementations of BALLSD, as well as the suite of tests used in this paper, are available for download at http://thekerrlab.com/ballsd.

Acknowledgements

C.C.K. was supported by the Australian Research Council (ARC) Discovery Early Career Researcher Award DE140101375. C.C.K. and S.D.B. were supported by the Defense Advanced Research Projects Agency (DARPA) Contract N66001-10-C-2008. C.C.K. and D.P.W. were supported by World Bank Assignment 1045478. T.G.S. was supported by National Institutes of Health grants NCRR 5P20RR016472-12 and NIGMS 8P20GM103446-12, and the National Science Foundation grants EPSCoR-0814251 and HRD-1242067. The authors wish to thank W. W. Lytton, D. Pokrajac, J. Francis, R. M. Stuart, and Z. McGrath for their helpful comments.

Author contributions

C.C.K. came up with the original algorithm and wrote the manuscript; T.G.S. revised and corrected the manuscript and provided refinements to the method; S.D.B. checked to see if C.C.K.'s code actually worked, and, since it did not, fixed it and added further functionality; D.P.W. led the conceptual development of the HIV resource allocation model that spurred development of this algorithm and was used here to demonstrate its use. All authors read and approved the manuscript.

References

Anderson SJ, Cherutich P, Kilonzo N, Cremin I, Fecht D, Kimanga D, Harper M, Masha RL, Ngongo PB, Maina W et al. (2014) Maximising the effect of combination HIV prevention through prioritisation of the people and places in greatest need: a modelling study. *The Lancet* 384:249–256.

Baker JL, Perez-Rosello T, Migliore M, Barrionuevo G, Ascoli GA (2011) A computer model of unitary responses from associational/commissural and perforant path synapses in hippocampal CA3 pyramidal cells. *Journal of Computational Neuroscience* 31:137–158.

Ben-Ameur W (2004) Computing the initial temperature of simulated annealing. *Computational Optimization and Applications* 29:369–385.

Bethke AD (1978) *Genetic algorithms as function optimizers.* University of Michigan.

Brom C, Vyhnánek J, Lukavský J, Waller D, Kadlec R (2012) A computational model of the allocentric and egocentric spatial memory by means of virtual agents, or how simple virtual agents can help to build complex computational models. *Cognitive Systems Research* 17–18:1–24.

Castillo P, Lozano R, Dzul A (2005) Stabilization of a mini rotorcraft with four rotors. *IEEE Control Systems Magazine* 25:45–55.

Charness G, Karni E, Levin D (2007) Individual and group decision making under risk: An experimental study of Bayesian updating and violations of first-order stochastic dominance. *Journal of Risk and Uncertainty* 35:129–148.

Goodner J, Tsianos GA, Li Y, Loeb GE (2012) BioSearch: A physiologically plausible learning model for the sensorimotor system. In *Proceedings of the Society for Neuroscience Annual Meeting* 275.22/LL11.

Hilbert M, López P (2011) The world's technological capacity to store, communicate, and compute information. *Science* 332:60–65.

Kerr CC, Van Albada SJ, Neymotin SA, Chadderdon GL, Robinson P, Lytton WW (2013) Cortical information flow in Parkinson's disease: a composite network/field model. *Frontiers in Computational Neuroscience* 7:1–14.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simmulated annealing. *Science* 220:671–680.

Kwon JA, Anderson J, Kerr CC, Thein HH, Zhang L, Iversen J, Dore GJ, Kaldor JM, Law MG, Maher L, Wilson DP (2012) Estimating the cost-effectiveness of needle-syringe programs in Australia. *AIDS* 26:2201–2210.

Marquardt DW (1963) An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics* 11:431–441.

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. *Journal of Chemical Physics* 21:1087–1092.

Metropolis N, Ulam S (1949) The Monte Carlo method. *Journal of the American Statistical Association* 44:335–341.

Nelder JA, Mead R (1965) A simplex method for function minimization. *Computer Journal* 7:308–313.

Prinz AA, Billimoria CP, Marder E (2003) Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons. *Journal of Neurophysiology* 90:3998–4015.

Rios LM, Sahinidis NV (2013) Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization* 56:1247–1293.

Song W, Kerr CC, Lytton WW, Francis JT (2013) Cortical plasticity induced by spike-triggered microstimulation in primate somatosensory cortex. *PLOS ONE* 8:e57453.

Steyvers M, Lee MD, Wagenmakers EJ (2009) A Bayesian analysis of human decision-making on bandit problems. *Journal of Mathematical Psychology* 53:168–179.

Wilby RL (2005) Uncertainty in water resource model parameters used for climate change impact assessment. *Hydrological Processes* 19:3201–3219.