

## Automated neuron model optimization techniques: a review

W. Van Geit · E. De Schutter · P. Achard

Received: 14 February 2008 / Accepted: 5 September 2008  
© Springer-Verlag 2008

**Abstract** The increase in complexity of computational neuron models makes the hand tuning of model parameters more difficult than ever. Fortunately, the parallel increase in computer power allows scientists to automate this tuning. Optimization algorithms need two essential components. The first one is a function that measures the difference between the output of the model with a given set of parameter and the data. This error function or fitness function makes the ranking of different parameter sets possible. The second component is a search algorithm that explores the parameter space to find the best parameter set in a minimal amount of time. In this review we distinguish three types of error functions: feature-based ones, point-by-point comparison of voltage traces and multi-objective functions. We then detail several popular search algorithms, including brute-force methods, simulated annealing, genetic algorithms, evolution strategies, differential evolution and particle-swarm optimization. Last, we shortly describe Neurofitter, a free software package that combines a phase-plane trajectory density fitness function with several search algorithms.

**Keywords** Optimization · Neuron · Model · Parameters · Automated tuning · Error function · Fitness function

### 1 Introduction

Tuning the parameters of neuron models has always been a challenge. However, because of the sharp increase in the computational power that is available to neuroscientists, which allows for the exploration of large sets of parameter values, and the increased complexity of neuron models, which makes the hand tuning of models more and more difficult, automated parameter search methods have become increasingly important. Automated tuning of neuronal model parameters is therefore an active topic and different algorithms have been published (Baldi et al. 1998; Bhalla and Bower 1993; Bush et al. 2005; Druckmann et al. 2007; Gerken et al. 2006; Keren et al. 2005; Lewicki 1998; Prinz et al. 2003; Tabak et al. 2000; Van Geit et al. 2007; Vanier and Bower 1999; Davison et al. 2000; Holmes et al. 2006; Huys et al. 2006; Pettinen et al. 2006; Reid et al. 2007; Weaver and Wearne 2006; Haufier et al. 2007; Tobin and Calabrese 2006).

The construction of a tuning algorithm for neuron models can be subdivided into two stages more or less independent from each other. The first is the choice of an error function (also called fitness function or cost function) that quantifies how well the model output compares with the experimental data. The second step is to pick an optimization algorithm to find minima of the error function. The choice is large but not straightforward since each of these algorithms is designed to handle a specific type of problems.

The two choices are independent because an optimization algorithm can usually be used with any error function and vice versa. There are however some exceptions to this rule. For example, multi-objective problems (Cohon 1985)

---

W. Van Geit · E. De Schutter (✉)  
Computational Neuroscience Unit, Okinawa Institute of Science  
and Technology, 7542 Onna, Onna-Son, Okinawa 904-0411, Japan  
e-mail: erik@oist.jp

W. Van Geit · E. De Schutter  
Theoretical Neurobiology, University of Antwerp,  
Universiteitsplein 1, Wilrijk, Antwerp 2610, Belgium  
e-mail: werner@tnb.ua.ac.be

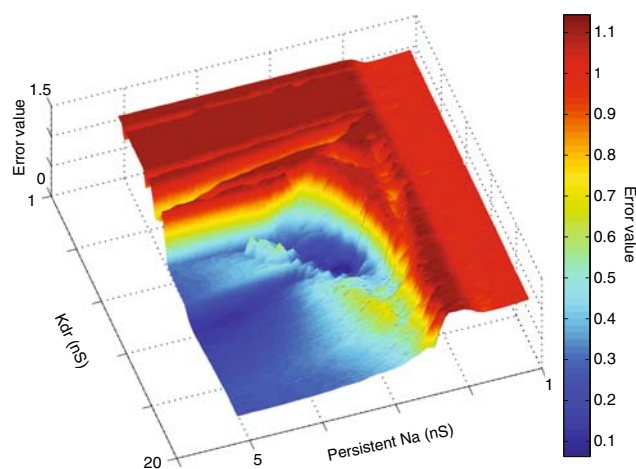
P. Achard  
Volen Center for Complex System, Brandeis University,  
415 South Street, Waltham, MA 02454, USA  
e-mail: achard@brandeis.edu

influence both the way error functions are used and which optimization algorithm can be selected. This subdivision is also not operant with some other methods, like the one described in Huys et al. (2006).

In this paper we will review the options available to scientists who want to use automated methods to fit neuron models. We will first describe the most common error functions that are used to compare electrophysiological traces of model neurons with data. We will then survey the different classes of algorithms existing on the market today. It is of course impossible to describe each algorithm and its many variations but our goal is to give a good overview of this field to allow the reader to find the way in the literature. Last, we will discuss Neurofitter (Van Geit et al. 2007), a software tool that allows the user to link different general optimization methods to the phase–plane trajectory density (PPTD) error function.

## 2 Error functions

Several methods exist to compare the output of models with experimental data and each of them has its own advantages and disadvantages. In this section we describe some of the error functions that have been recently used in the literature. These functions describe either the fitness of the solution, in which case a higher fitness value corresponds to a better solution, or the error made and then one wants to minimize the function. We have chosen to use the terminology of the latter approach in the text.



**Fig. 1** Error function landscape. The error function value of a model is plotted as a function of its two free parameters (persistent sodium current conductance and delayed rectifier potassium current conductance). The model used is of a bursting pre-Bötzing complex neuron (Butera et al. 1999). The maximal conductances of the two ion channels are varied along the X and Y-axes. The Z-axis shows the value obtain by calculating the error measure using the PPTD method. The target output was obtained with parameter values 2.8 and 11.2 nS for  $g_{NaP}$  and  $g_{KDr}$ , respectively. The method used to obtain the data in this plot is described in more detail in Van Geit et al. (2007)

The choice of an error function (See Fig. 1 for an example) is very important since it will influence the final choice of a parameter set (a solution) and the performance of the search algorithm used. Several criteria should be taken into account when creating an error function, among those we would like to emphasize:

*Relevance:* The function should reflect fundamental properties of the data that the model has to reproduce.

*Speed:* The function should be fast to calculate, since typically a large amount of error evaluations are performed during the search.

*Smoothness:* The solution space should be as simple as possible, so that the search algorithm can rapidly converge to a global optimum. This in general means as little local minima as possible.

### 2.1 Feature-based

A first approach to define an error function, which has already proven successful, is to use features of the data as points of comparison with the models. Examples of such features include average spike height and spike width, resting membrane potential, after-hyperpolarizing potential, firing rate, bursting rate, etc.

One advantage is that these criteria can be averaged over different data sets, so that a model can be fitted to the statistics of a trace instead of to the properties of a single recording (Druckmann et al. 2007). This can be useful since the response of a cell to a certain stimulus can vary significantly from trial to trial, even when repeated after a short interval. So it makes sense to emphasize more the average behavior of a cell to reduce the effect of noise.

Another advantage is the focus of the method on criteria that the scientist judge relevant. If one designs a model in order to reproduce, for example, a given firing rate with a given variability with no interest at all in the height of the spikes or the sub-threshold potential waveform, this type of error function is definitively to be advised.

All these measures need however, clear prior definitions, which are very user-specific and sometimes difficult to automate. Indeed, during the exploration of the full parameter space a model might have very erratic behaviors and the measure in use should be robust. Measuring the spike height for example relies on spike detection and can be quite complicated (Lewicki 1998). Spikes have to be distinguished from sub-threshold variations, plateaus, single-spike bursts, spikelets or noise and their definition has to be stable for different resting potentials. It is therefore impossible to come to a unique definition of a spike that will be effective in any cell, in any location (dendrite, soma, axon, etc.) and in any experimental condition (patch-clamp, extracellular recording, voltage dye imaging, etc.).

### 2.2 Point-to-point comparison

Another approach is to compare electrophysiological traces in a more direct manner. The easiest method is to calculate the root mean square of the difference between the data and the model voltage traces:

$$\text{rms} = \sqrt{\frac{1}{N} \sum_{i=0}^N (V_{\text{data}}[i] - V_{\text{model}}[i])^2} \quad (1)$$

where  $V_{\text{data}}[i]$  is the data voltage at the  $i$ th time step and  $V_{\text{model}}[i]$  the model one.  $N$  is the number of time steps in the voltage recording.

This approach faces the problem that it is very sensitive to time shifts between the traces. This is illustrated in Fig. 2, where a spike train is compared to itself with a small time shift and can cause an error value that is larger than when it is compared to a flat trace. An obvious solution to this problem would be to shift the traces before comparing them, so that the peaks of the spikes coincide. While this is easy for spike trains that are regular, it creates serious problems in more complex cases. For example, comparing spikes that belong to bursts when the number of spikes per burst is not identical makes the realignment impossible.

The PPTD method (LeMasson 2001) belongs to this same category but presents fewer problems with phase shifts. We will describe it more extensively in the last section of this article.

Handling high levels of noise can be more difficult with point-to-point comparison than with feature-based error functions. With low noise levels and cell-to-cell variation, the above equation can be normalized to the noise level.

### 2.3 Multi-objective

Different criteria can be combined to determine how good a model is. Sometimes these criteria do not influence each other and can be optimized almost separately, but most of the

time there exists some conflicts between them. The simplest solution is to combine the error functions of the different criteria by as a weighted sum:

$$f(\bar{x}) = w_1 f_1(\bar{x}) + w_2 f_2(\bar{x}) + \dots + w_n f_n(\bar{x})$$

with  $\bar{x}$  the vector of parameters,  $w_i$  the weights and  $f_i$  the error functions.

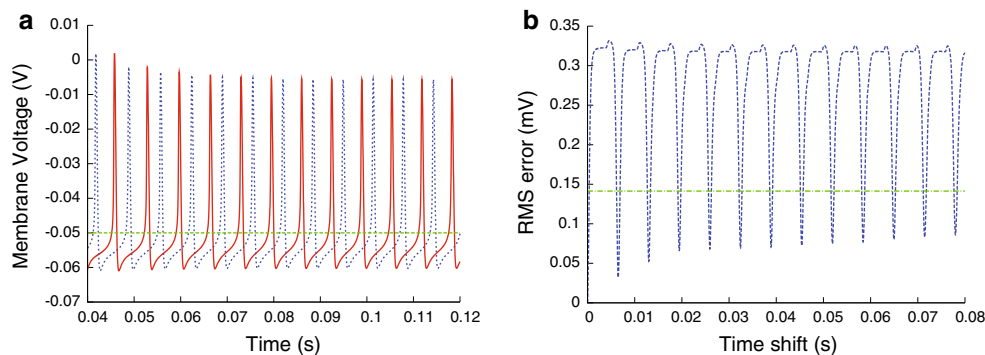
Another solution is to consider each criterion equally important and to find an equilibrium between them. The different criteria are in this case called *objectives* (Handl et al. 2007).

The best way to deal with multi-objective problems is to use the concept of dominance. A solution X is said to *dominate* another solution Y, if one of X's objectives is better than the corresponding one of solution Y, and all the other objectives are at least as good as Y's. For any pair of solution X and Y it is possible that "X dominates Y," "Y dominates X" and "neither X nor Y dominate each other." A solution that is not dominated by any other solution will be part of the Pareto front. The Pareto front is therefore formed of all the dominating solutions (see Fig. 3). Any solution in the Pareto front is considered as good as the other one and the user has the final word to select one or several of them.

## 3 Search techniques

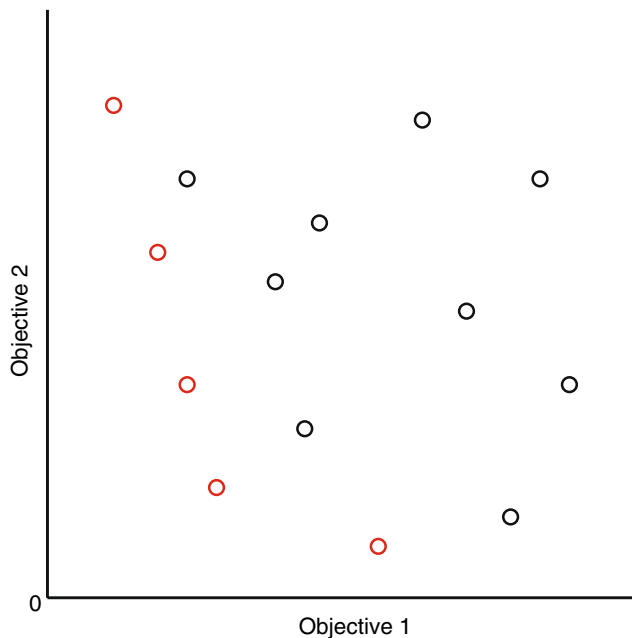
### 3.1 General considerations

After determining the most appropriate error function, one has to decide which algorithm to apply in order to find the optimal solution of the problem. These algorithms can be classified as local or global, deterministic or stochastic, discrete or continuous, single or multi-objective, etc.



**Fig. 2** Root mean square error measure. **a** Two spike trains that show a phase difference (*red* and *blue*). A trace with constant voltage is also shown (*green*). **b** The  $Y$ -axis shows the root mean square error between

the traces of **a** when one of them is shifted in time ( $X$ -axis). The difference between the spike trains (*blue*) is higher for certain time shifts, than the difference between a spike train and the flat trace (*green*)



**Fig. 3** Pareto front. Every *circle* represents a possible solution of a two-dimensional multi-objective optimization problem, where lower objective values are better. The *red circles* form the Pareto front that consists of all the solutions that are not dominated

### 3.1.1 Global versus local algorithms

The choice of a search algorithm for a given optimization problem depends on several factors, of which one of the most important is the shape of the solution space given an error function (Price et al. 2005b). An important property of this space is the density of local minima, i.e. points where the *local* gradient is zero. The goal of an optimization algorithm is however, to find the *global* minimum, the point with the smallest error values in the solution space. When the error function does not contain any (or very few) local minima, a method that stops at the first minimum, such as local gradient or simplex algorithms, is very appropriate. Unfortunately, it is difficult to evaluate the number of local minima before running the search algorithm and many biologically relevant optimization problems have a lot of local minima (Achard and De Schutter 2006; Banga et al. 2003). So in general it is better to use *global* optimization techniques, which contain mechanisms to prevent them from getting stuck in a local minimum.

### 3.1.2 Global minima

While theoretically speaking many optimization problems may have only 1 unique global minimum, in practical (biological) systems it might be useful to consider the possibility of multiple global “optima.” In the case of neuron model parameter optimization it can be readily assumed that

because of noise in the experimental data, and because of the incompleteness of the neuron model, it is impossible to reach the global minimum, i.e. to find a set of parameters that “perfectly” replicate the experimental data. Therefore the criteria can be relaxed, and an “optimal” model can be defined as one that shows a negligible difference in behavior compared to the experimental data. In very simple neuron models with very few free parameters, chances are high that the “optimal” models are all centered around one point in solution space, with only minute differences between the parameter values. However, research (Achard and De Schutter 2006; Bhalla and Bower 1993; Goldman et al. 2001; Prinz et al. 2003; Foster et al. 1993; Golowasch et al. 2002; Swensen and Bean 2005; Olypher and Calabrese 2007) has shown that this does not always have to be the case. It is possible to have both real neurons and neuron models with very similar behavior (which could all be considered optimal), but with very different values for their parameter values.

In the rest of the text, a terminology considering “one global optimum” will be used, but the remarks of this section will remain valid.

### 3.1.3 The exploration/exploitation balance

Since in theory the global minimum to a general optimization problem can be located anywhere in the solution space, the convergence of global search algorithms is only guaranteed within infinite time. In practice of course, the goal of these algorithms is to find a global solution with a reasonable probability in a finite amount of time. This creates a trade-off between two concepts: *exploration* and *exploitation* (Eiben and Schippers 1998).

The exploration should be maximized, which means that the algorithm should visit as many regions of the solution space as possible when searching for the global solution. But, on the other hand, the algorithm must also exploit as much information as possible from the solution space already visited during early stages of the search. Any algorithm has to find a balance between these two concepts. At one extreme, methods like brute force search explore the solution space in a completely fixed manner maximizing the exploration. On the other side, the gradient descent methods exploit the local gradient information maximally but contain no exploratory component.

### 3.1.4 Stochastic versus deterministic algorithms

The difference between stochastic and deterministic methods lies in the use of random numbers. The future state of stochastic algorithms will not be completely defined by their previous state, provided that the pseudo-random number generator has a long enough period. The introduction of

randomness is one of the easiest ways to increase the exploration properties of the algorithms.

The structure of many global stochastic search methods (including evolutionary algorithms, particle swarm optimization, etc.) is quite similar. A loop is executed in which each step consists of the choice of a number of points in the parameter space, i.e. of parameter sets. The error function is evaluated in all of these points and a new list of points is drawn from a probability distribution that can depend on the results of the previous steps in the loop. From this list of points subsequently a selection is made that will become the starting points for the next round in the loop.

### 3.2 Brute force search

The simplest search method is the brute force search. It consists of a scanning of the parameter space without taking into account any information previously retrieved during the search. This approach is useful if the number of parameter is limited since the number of error evaluation needed grows as a power of the number of parameters of the problem. Methods described in the literature use this technique (Prinz et al. 2003).

The scan can be performed deterministically or randomly. In the first case, the error function is evaluated on a number of equidistant points in every dimension of the parameter space creating a mesh. This is useful to get a better understanding of the parameter space of the model, since it allows visualizing the different parameter zones in which the model has a certain behavior (like spiking, bursting, silent, etc.) (Taylor et al. 2006).

In the stochastic approach, the so-called ‘blind random search’, the points in the parameter space where the error function is evaluated are chosen completely randomly. This has as advantage that the spread of the points contains no artificial bias introduced by the shape of the mesh. The random spread search is also a very useful baseline to benchmark the ‘intelligence’ of other search algorithms against.

### 3.3 Local random search

A small improvement in exploitation of the ‘blind random search’ can be achieved with a fairly simple adaptation. Instead of choosing every new point completely randomly, these points are defined as  $y_k = x_k + r_k$  where  $x_k$  are the points of the previous generation, and  $r_k$  is an independent random vector. The point  $y_k$  replaces the solution  $x_k$  if it has a better error value. While this method seems very similar to the blind random search, it has as advantage that it exploits some information about the previous loop and so can be considered more ‘intelligent’ than the blind search.

### 3.4 Gradient descent

The gradient descent methods search for local minima by using the fact that, at these locations, the error function  $f$  has a null gradient  $\nabla f$  (Broyden 1967). Minima are hence found by following the gradient downhill.

One of the simplest forms of gradient descent is the steepest descent method. As the name already points out, the method always moves from point  $a$  to point  $b$  following the equation  $b = a - \gamma \nabla f(a)$  until  $\nabla f = 0$ . If  $f(b) < f(a)$  then the point  $b$  replaces the point  $a$ . If not, then  $\gamma$  as to be reduced. Choosing the appropriate initial value for  $\gamma$  is very important. If it is too small, the algorithm takes a huge amount of time to reach the minimum whereas if it is too large, the search points may always overpass the minimum.

The Newton method is a bit more advanced in the sense that it searches the zero of  $\nabla f$  by also making use of the second derivative for 1-dimensional problems or, as a generalization, the Hessian in higher dimensions. The next point of the search in this case is calculated by  $b = a - \gamma [Hf(a)]^{-1} \nabla f(a)$ . It requires the evaluation of the Hessian matrix  $Hf$  in the point  $a$ , which could be quite computationally expensive, but this is compensated by the fact that the method converges faster. Quasi-Newton methods, which improve the speed by approximating the Hessian matrix, are more efficient than the steepest method in most cases.

### 3.5 Simulated annealing

Just as other optimization methods, simulated annealing was inspired by natural phenomena. Annealing is a process used in material sciences to improve the properties of a material by heating it and then cooling it very slowly. During the hot period, atoms are very mobile and can therefore move away from their initial positions. The slow temperature decrease allows atoms to crystallize as smoothly as possible, i.e. to find low internal energy configurations. In a sense this is also what one wants to do while performing a search method. In the beginning, a larger degree of freedom is desired so that exploration of the solution space can take place. After a while a cooling process should start in which it is getting harder and harder to escape from optimal solutions.

The method (Kirkpatrick et al. 1983) defines a temperature  $T$  of the search and a candidate solution  $x$ . During each step of the algorithm there exists a probability of replacing the candidate solution with a neighboring point  $y$ . This point can be chosen in a problem-specific way. It is often generated by adding to the original point a vector whose coordinates are chosen from a Gaussian distribution.

The probability of replacement is generally calculated with a Boltzmann–Gibbs distribution:

$$p_{xy} = \begin{cases} 1 & \text{if } f(x_k) < f(y_k) \\ e^{-\frac{f(x_k) - f(y_k)}{cT}} & \text{otherwise} \end{cases} \quad (2)$$

with  $c$  a positive constant,  $T$  the temperature and  $f$  the error function.

Since this equation gives a non-null probability to the replacement of the candidate solution with a solution with a larger error value, it is possible to escape from local minima.

This probability, however, depends on the temperature. In the beginning of the search the chances of jumping uphill are higher than in later stages. How the temperature evolves during the search method can be defined in several ways. There can be exponential cooling  $T_{k+1} = \rho \cdot T_k$ , other possibilities include linear cooling or logarithmic cooling (Nourani and Andresen 1998).

The temperature-changing scheme can be itself a parameter that is optimized, in which case one speaks of *adaptive simulated annealing*.

### 3.6 Evolutionary search algorithms

Evolutionary algorithms are stochastic algorithms that are based on natural evolution (Eiben and Smith 2003). In the same way as nature uses the principle of “survival of the fittest” to improve the overall quality of the individuals in a population over a long time, evolutionary search algorithms thread the points in the parameter space as individuals that have a certain fitness (or error value) and hence a certain probability to survive and to generate siblings.

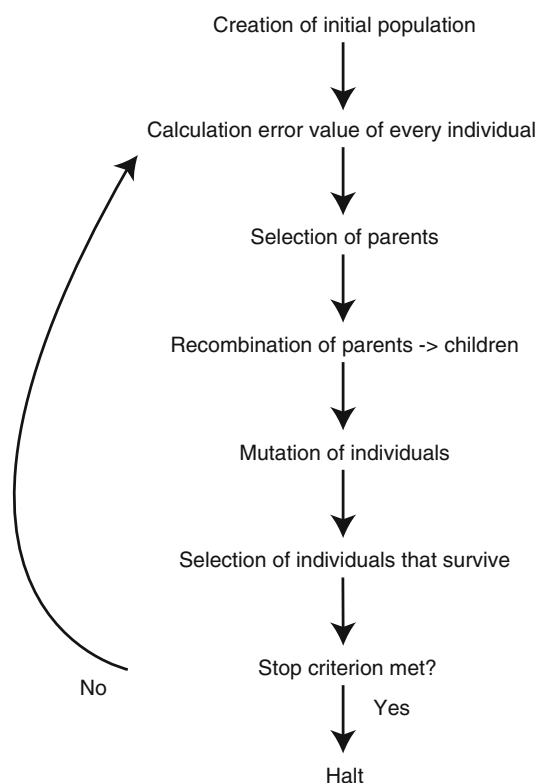
Different subsets of evolutionary algorithms exist. Among them are the, probably the best well-known, genetic algorithms, but also the evolution strategies and differential evolution.

All evolutionary algorithms follow the same general organization (see Fig. 4). First an initial population of random individuals is created. The fitness value of each individual is then calculated. Some individuals are selected, according to their fitness, to become parents and reproduce themselves and be mutated. The reproduction consists of mixing the parameters of several parents. The mutation is a random modification of the parameters of a single individual. The new individuals generated by reproduction and mutation are then evaluated and compared to the original population. A selection of the best individuals will form a new population that can evolve, until some stopping criterion is met.

The exploratory part of the algorithm is driven by the evolution (mutation and reproduction) while the exploitation part is due to the selection process.

#### 3.6.1 Genetic algorithms

The genetic algorithm became widely known thanks to the work of Holland (Holland 1975). The most general



**Fig. 4** EA. The general structure of an evolutionary algorithm

representation of individuals in genetic algorithms is a finite string of bits. In a problem where for example the parameters are vectors with integer coordinates, the individuals can be represented as a concatenated string of the bit representation of their coordinates. Mutation takes place by flipping one or more bits of the parameters and reproduction by combining the bit strings of parents.

The mutation and reproduction operators used by genetic algorithms are based upon the bit representation of the solutions, which is problematic in cases when the parameters contain floating-point numbers. Since the binary representation of floating point numbers can be quite complicated, flipping bits, or pasting the strings together to form new children, is less useful. This is why for these kind of problems, the evolutionary strategies approach, introduced by Rechenberg and Schwefel (Rechenberg 1973), is preferred, since it is specifically designed for floating-point optimization problems.

#### 3.6.2 Evolution strategies

In evolution strategies (ES), the mutated individuals are generated by adding a vector  $r_i$ , with coordinates drawn from a Gaussian distribution, to an individual  $x_i$  from the population. This makes the mutation operator more *natural* than the bit flipping. The standard deviation of the normal distribution

used in the mutation operator determines the mutation rate. The higher this rate, the more the algorithm will explore the solution space, at the cost of a lower exploitation. A high mutation rate is desirable in the beginning of the search and a low one in the end. The mutation rate can be changed with a heuristic rule during the algorithm, but in ES a common technique used is to make the mutation rate self-adapting, meaning that it becomes a parameter of the search itself. This way the balance between exploration and exploitation is optimized itself during the search.

The reproduction or recombination operator in ES can be constructed in different ways. The number of parents can be greater than two, the more parents are chosen the less “local” the search will be, since the chances are higher to combine features from individuals that are very far away in the solution space. The way the parents are combined into a child can be done according to different rules. When using the *discrete* method, every parameter value of the child corresponds to the corresponding value of exactly 1 parent. For example the parents  $(x_1, x_2, x_3, x_4, \dots)$  and  $(y_1, y_2, y_3, y_4, \dots)$  could generate the child  $(x_1, y_2, y_3, x_4, \dots)$ . When *intermediate* recombination is used a child is of the form  $((x_1 + y_1)/2, (x_2 + y_2)/2, (x_3 + y_3)/2, (x_4 + y_4)/2, \dots)$ .

The selection procedure comes also in different flavors. In case of a *plus* strategy, the survivors will be chosen from both the parents and the children. A *comma* strategy, however, only selects from the children. The methods used to perform the selection process include *roulette* and *tournament* selection. All these methods give a higher chance of survival to an individual depending on its error value. More information about the different kinds of operators in use within the field of Evolution Strategies can be found in Eiben and Smith (2003).

### 3.6.3 Differential evolution

Differential evolution (Price et al. 2005a) is another method that belongs to the family of the evolutionary algorithms. One of the problems of the evolutionary strategies and genetic algorithms is that they can become very complicated to implement because of all their types of operators, and all the relevant settings. The advantage of the differential evolution is its simplicity and limited amount of settings, which was one of the goals in its design, while it stays very versatile and efficient.

The algorithm starts with a random population of points in the parameter space. New generations are created by taking for every member  $p$  of the generation, three completely random individuals  $x, y, z$  from the same generation. A child  $c$  is created as  $c = x + r(y - z)$ , with  $r$  a positive real number. The child competes with its parent  $p$ , and replaces it if its error value is smaller. This procedure is repeated as many times as there are individuals in the population, and after

that a new generation is started. Interestingly, in this method, the mutation size by which the individual  $x$  is perturbed is proportional to the difference between two other individuals  $(y - z)$ . This means that it only depends on the previously generated points. Hence the amount of exploration is automatically regulated. If all the best points are in the same vicinity, the mutation size will be small, increasing exploitation. If the best solutions found are far away from each other, the mutation size will remain large, allowing more exploration to take place.

### 3.6.4 Population diversity

A classical problem that can appear during a search performed by evolutionary algorithms is the lack of diversity in the population. This seriously limits the rate of exploration, since many individuals will cluster around certain points in the solution space.

A solution that has been proposed is the concept of fitness sharing (Goldberg and Richardson 1987). The idea is to adapt the error function to prevent the clustering of the individuals. To measure the amount of clustering a distance function  $d$  is defined in the parameter space. If the error function is given by  $E(i)$ , and the sharing function by  $S(d(i, j))$  then the new error function is defined as

$$E'(i) = \frac{E(i)}{\sum S(d(i, j))} \quad (3)$$

with  $j$  in the sum iterating over all individuals in the population. The sharing function is monotonically increasing function in  $d$ , which means that the new error function  $E'(i)$  assigns a larger error value to a parameter set if there are more other individuals nearby. This way the algorithm gains from pushing clustered individuals away from each other, thus increasing its exploration and the diversity of the found solutions.

Another solution is to define multiple subpopulations that are almost blind to each other (Potter and De Jong 1994). By reducing the probability of reproduction between subpopulations, one cluster of individuals is unlikely to attract individuals from other subpopulations.

### 3.6.5 Multi-objective evolutionary algorithm

Adapting evolutionary algorithms to allow for multi-objective optimization is relatively straightforward. As we have already described earlier in the text, a multi-objective has not one error function, but several that all have to be optimized simultaneously, and a way to handle this uses the concept of domination. For example the non-dominated sorting genetic algorithm (NSGA) (Srinivas and Deb 1994), subdivides the population in fronts of individuals that are dominated by an equal amount of other individual. So, the

Pareto front is one of these fronts, since these individuals are dominated by 0 others. The ordering that this method introduces, is then used in the selection operator of the GA, with the Pareto front individuals having the highest probability of survival. To increase diversity, fitness sharing is used in this algorithm. Other ways to implement multi-objective optimization in evolutionary algorithms exist, like the Niche Pareto Genetic Algorithm (NPGA) (Horn et al. 1994) or the Multi-objective Genetic Algorithm (MOGA) (Fonseca and Fleming 1993).

### 3.7 Particle swarm optimization

In the particle swarm optimization (PSO) algorithm a swarm of particles, each representing a possible solution, are “flying” around in the parameters space as do flocks of birds or fishes (Kennedy and Eberhart 1995).

At the start of the algorithm all the particles in the swarm are spread randomly across the parameter space and have a random velocity. Particles are connected to each other in a certain topology. Neighboring particles in the topology share information about the optimal solution they have found during the search as described by Eq. 4. Different topologies exist like star-like structures, or sets of interconnected local neighborhoods or purely random ones.

After the initialization the particles start moving in the parameter space, the position of particle  $k$  is given by

$$\begin{aligned}x_{k+1} &= x_k + v_k \\v_{k+1} &= w \cdot v_k + r_1 c_1 \cdot (x_{\text{neighborbest}} - x_k) \\ &\quad + r_2 c_2 \cdot (x_{\text{localbest}} - x_k)\end{aligned}\quad (4)$$

with  $v$  the velocity,  $w$  the *inertia* factor,  $c_1$  and  $c_2$  positive constants and  $r_1$  and  $r_2$  random numbers,  $x_{\text{localbest}}$  points to the best solution found by the particles itself so far, and  $x_{\text{neighborbest}}$  by its neighborhood.

As one can see there is exploration in the algorithm as a result of inertia, but also exploitation as information from the past of the search is used in the decision to choose new points.

### 3.8 Hybrid

As a result of the no-free-lunch theorem (Wolpert and Macready 1997) no search algorithm is perfect for every kind of situation. Some methods combine different search methods to take into account different needs during different period. For example we have discussed extensively the fact that some algorithm are better in exploration than exploitation and vice versa. Since typically in the beginning of the algorithm the exploration of the solution space is very important, one can choose for a more globally oriented search method during this stage (Goldberg and Voessner 1999).

Once a region (or multiple regions) has been found close to a possible global optimum, a switch can be made to local search methods to exploit the gradient information contained in this region more efficiently.

## 4 Neurofitter and the PPTD method

### 4.1 Phase-plane trajectory density

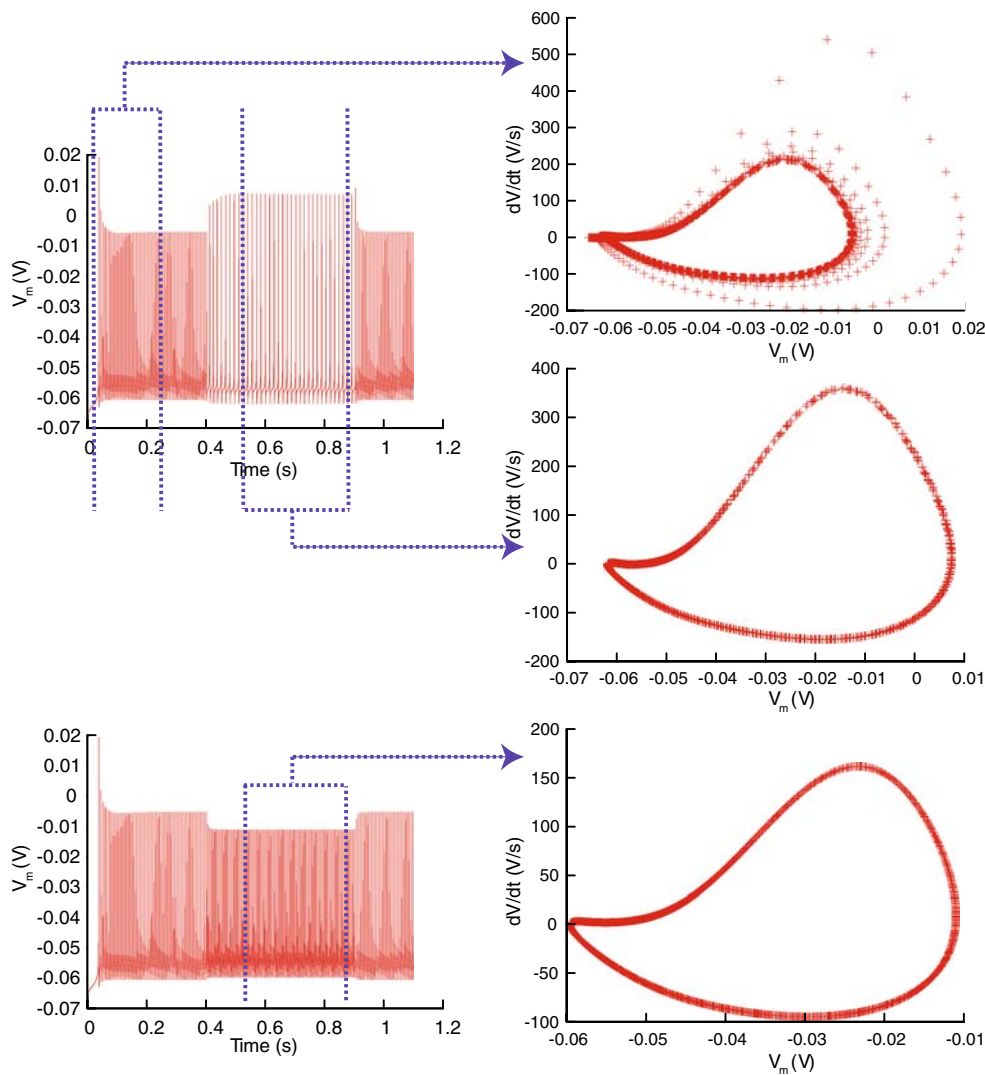
Neurofitter is a software tool specifically developed to help neuroscientists to tune neuron models. It provides a link between external optimization methods and an error function based on the PPTD method. This technique (LeMasson 2001) has been shown to be much less sensitive than other point-to-point comparison methods to time shifts (Achard and De Schutter 2006). It achieves this by not comparing the voltage traces in the time domain, but by calculating the difference between two-dimensional histograms of the voltage and its time derivative (see Fig. 5).

In voltage traces that contain spikes, the phase plot shows a circular shape that depends on the exact shape of the spikes in the time-dependent trace. So when the shapes of two  $V - dV/dt$  histograms agree, it means that the spikes have a similar shape, an important criterion used to compare models with experimental data. It also means that the resting membrane potential and the shape of sub-threshold activity are similar. Time shifts between traces are not a problem anymore since information about the exact timing of the spikes is discarded. This last property, however, can also be considered as a disadvantage, since in some cases the exact timing of the spikes is important. This problem can be solved by specifying time ranges for which separate phase-plane trajectory plots are generated. This allows the user to specify at which times in the trace the model has to show a certain behavior, so that for example tonic-firing can be separated from bursts. By specifying time ranges of just 1 or a couple of spikes, the user is also able to “zoom” in on certain parts of the trace, and to require the model to fire spikes at very specific times.

The error value is computed as the square difference between the histograms of both the model output and the experimental data:

$$E = \sqrt{\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \left( \frac{\text{data}_{ij}}{N_{\text{data}}} - \frac{\text{model}_{ij}}{N_{\text{model}}} \right)^2} \quad (5)$$

with  $\text{data}_{ij}$  and  $\text{model}_{ij}$  the number of points in the bins of the data and model  $V - dV/dt$  histograms, respectively,  $N_{\text{data}}$  and  $N_{\text{model}}$  the total number of points in the traces, and  $N_x$  and  $N_y$  the number of bins of the two-dimensional histogram. The size of the bins has to be small enough to



**Fig. 5** Phase–plane trajectory density plot. Two voltages traces recorded from a cerebellar Purkinje cell model subjected to  $-0.17$  and  $0.18$  nA step current injection, respectively (in time interval between

$0.4$  and  $0.9$  s), are transformed into PPTD plots. The *dashed lines* show to which time range each PPTD plot corresponds. Note that the axis ranges are different for the three *right panels*

capture important details of the phase trace but large enough to be insensitive to noise or sampling approximations.

#### 4.1.1 Weights

The introduction of time ranges, and the possibility to combine traces from different recordings to fit the model (see Fig. 5), introduces the problem that different error values, generated by comparing several histograms have to be added together to get a global value. The method allows the user to specify weights for all of the different histograms.

$$E = \sum_{i=1}^{N_{\text{recordingsites}}} \sum_{j=1}^{N_{\text{protocols}}} \sum_{k=1}^{N_{\text{timeranges}}} w_{ijk} \cdot e_{ijk}$$

with  $i$  ranging over the different recording sites,  $j$  over the different stimulation protocols and  $k$  over the relevant time ranges,  $e_{ijk}$  the error values of the different traces calculated according to Eqs. (1) or (2) and  $w_{ijk}$  the weights of the different traces in the global error function  $E$ .

#### 4.2 Neurofitter

To make it easy for the user to combine the phase–plane trajectory method and good optimization algorithms we have created a software tool called Neurofitter (Van Geit et al. 2007). The core task performed by Neurofitter is the computation of a PPTD error function.

The only requirement imposed by the method that Neurofitter uses to the experimental data, is that it consists of

time-dependent traces. The software has, until now, always been tested with voltage traces, but this is not a requisite per se, in essence for example current recordings during voltage clamp experiments can be also used. One should however be careful by combining traces recorded from different cells, since it is not guaranteed that one cell model will be able to be fitted to all of these at once.

The user has to provide Neurofitter with the model that has to be fitted to the experimental data. The model can be written in Neuron (Hines and Carnevale 1997) or Genesis (Bower and Beeman 1998), or any other neuron model simulation language, as long as the interpreter can be started with a shell command. The parameters with which the model has to be evaluated are passed to the interpreter by way of a file, which should be read by the model code. After execution Neurofitter will read back the trace containing the output, which should have the same sampling rate as the experimental data.

The search algorithms implemented in Neurofitter are Brute force search, evolutionary strategies, particle swarm optimization and nonlinear optimization for mixed variables and derivatives (Audet and Orban 2004). These algorithms are well suited to be parallelized. This is very useful since a typical search requests the evaluation of a large number of parameter values. When models are complex, the exact class of models for which Neurofitter is very useful, this can require of a large amount of computational power. The user can choose at which level to parallelize the search so that it can be adapted to the available computing resources.

## 5 Conclusions

We would like to emphasize that the list of techniques described in this text is not exhaustive. Our goal was to provide a description of some of the methods used in the field, and to give a more thorough description of the technique that we use in the Neurofitter package, namely the PPTD method. A neuroscientist that wants to use any of the methods available to tune neuron models should keep in mind some rules. It would be ideal if a ready-made solution was available that would solve all tuning problems. But, according to the NFL theorems, it will not be possible to find a general search method that has the best performance for all situations, so it might be worthwhile to spend some time experimenting with different techniques, and maybe most importantly, with the settings of the search methods. As an extra problem, it might be a difficult task for the user to discern which techniques are better, especially in the case of stochastic methods. If one stochastic search algorithm outperforms another in a certain case, the test procedure has to be repeated with a lot of different random seeds, and even then making a small change to for example the error function might flip the result.

Finding the right error function that you need, might be the hardest issue. While some of the methods that are available try to automate this process as much as possible, there are always some parameters that have to be set by the user, like for example the weights of different sub-problems in the error function.

We wish to stress that automated parameter tuning cannot magically turn a bad model into a good one. When fitting real experimental data, it is important to have the parts of the model that are not fitted by the search, like possibly the kinetics of the ion channels, already within acceptable margins. It is also necessary to have in the model all the fundamental components needed to obtain the desired output. If for example a slow current is visible in the experimental data, and the model does not contain such a current, it will be impossible for the search method to find a good fit of the traces. Since there is not a limit on which parts of the model can be automatically fitted, it is better that if one is not confident about certain aspects, like the parameters for the ion channel kinetics, to also include these in the fitted parameters of the model.

We believe that the development of automated search methods for neuron model tuning is a field that will receive more and more attention in the future. The combination of both an increase in parameters of neuron models that have to be tuned, and the larger computational power that is becoming available to neuroscientists, is likely to give a boost to this field of science.

## References

- Achard P, De Schutter E (2006) Complex parameter landscape for a complex neuron model. *PLoS Comput Biol* 2:e94
- Audet C, Orban D (2004) Finding optimal algorithmic parameters using a mesh adaptive direct search. *Cahiers du GERAD G-2004-xx*
- Baldi P, Vanier MC, Bower JM (1998) On the use of Bayesian methods for evaluating compartmental neural models. *J Comput Neurosci* 5:285–314
- Banga JR, Moles CG, Alonso AA (2003) Global optimization of bio-processes using stochastic and hybrid methods. In: Floudas CA, Pardalos PM (eds) *Frontiers in global optimization. Nonconvex optimization and its applications*. Kluwer, Dordrecht, pp 45–70
- Bhalla US, Bower JM (1993) Exploring parameter space in detailed single neuron models: simulations of the mitral and granule cells of the olfactory bulb. *J Neurophysiol* 69:1948–1965
- Bower JM, Beeman D (1998) *The book of GENESIS exploring realistic neural models with the GENERAL NEURAL SIMULATION SYSTEM*, 2nd edn. Springer, New York
- Broyden CG (1967) Quasi-Newton methods and their application to function minimisation. *Math Comput* 21:368–381
- Bush K, Knight J, Anderson C (2005) Optimizing conductance parameters of cortical neural models via electrotonic partitions. *Neural Netw* 18:488–496
- Butera RJ, Rinzel J, Smith JC (1999) Models of respiratory rhythm generation in the pre-Bötzinger complex. I. Bursting pacemaker neurons. *J Neurophysiol* 82:382–397

- Cohon JL (1985) Multicriteria programming: brief review and application. In: Gero JS (eds) Design optimization. Academic Press, New York
- Davison AP, Feng J, Brown D (2000) A reduced compartmental model of the mitral cell for use in network models of the olfactory bulb. *Brain Res Bull* 51:393–399
- Druckmann S, Banitt Y, Gideon A, Schurmann F, Markram H, Segev I (2007) A novel multiple objective optimization framework for automated constraining of conductance-based neuron models by noisy experimental data. *Front Neurosci* 1:7–18
- Eiben AE, Schippers CA (1998) On evolutionary exploration and exploitation. *Fundam Inf* 35:35–50
- Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, Heidelberg
- Fonseca CM, Fleming PJ (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. Genetic algorithms: proceedings of the fifth international conference, pp 416–423
- Foster WR, Ungar LH, Schwaber JS (1993) Significance of conductances in Hodgkin–Huxley models. *J Neurophysiol* 70:2502–2518
- Gerken WC, Purvis LK, Butera RJ (2006) Genetic algorithm for optimization and specification of a neuron model. *Neurocomputing* 69:1039–1042
- Goldberg DE, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. Proceedings of the second international conference on genetic algorithms on genetic algorithms and their application, pp 41–49
- Goldberg DE, Voessner S (1999) Optimizing global-local search hybrids. *Proc Gen Evol Comput Conf* 1:220–228
- Goldman MS, Golowasch J, Marder E, Abbott LF (2001) Global structure, robustness, and modulation of neuronal models. *J Neurosci* 21:5229–5238
- Golowasch J, Goldman MS, Abbott LF, Marder E (2002) Failure of averaging in the construction of a conductance-based neuron model. *J Neurophysiol* 87:1129–1131
- Handl J, Kell DB, Knowles J (2007) Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Trans Comput Biol Bioinform* 4:279–292
- Haufler D, Morin F, Lacaille JC, Skinner FK (2007) Parameter estimation in single-compartment neuron models using a synchronization-based method. *Neurocomputing* 70:1605–1610
- Hines ML, Carnevale NT (1997) The NEURON simulation environment. *Neural Comput* 9:1179–1209
- Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press, Ann Arbor
- Holmes W, Ambros-Ingerson J, Grover L (2006) Fitting experimental data to models that use morphological data from public databases. *J Comput Neurosci* 20:349–365
- Horn J, Nafpliotis N, Goldberg DE (1994) A Niche Pareto genetic algorithm for multiobjective optimization. *Proc First IEEE Conf Evol Comput IEEE World Cong Comput Intell* 1:82–87
- Huys QJ, Ahrens MB, Paninski L (2006) Efficient estimation of detailed single-neuron models. *J Neurophysiol* 96:872–890
- Kennedy J, Eberhart RC (1995) Particle swarm optimization. Proceedings of the IEEE international joint conference on neural networks, pp 1942–1948
- Keren N, Peled N, Korngreen A (2005) Constraining compartmental models using multiple voltage recordings and genetic algorithms. *J Neurophysiol* 94:3730–3742
- Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. *Science* 220(4598):671–680
- LeMasson M (2001) Introduction to equation solving and parameter fitting. In: Computational neuroscience: realistic modeling for experimentalists. CRC Press, London, pp 1–23
- Lewicki MS (1998) A review of methods for spike sorting: the detection and classification of neural action potentials. *Network* 9:R53–R78
- Nourani Y, Andresen B (1998) A comparison of simulated annealing cooling strategies. *J Phys Math General* 31:8373–8385
- Olypher AV, Calabrese RL (2007) Using constraints on neuronal activity to reveal compensatory changes in neuronal parameters. *J Neurophysiol* 98:3749–3758
- Pettinen A, Yli-Harja O, Linne M (2006) Comparison of automated parameter estimation methods for neuronal signaling networks. *Neurocomputing* 69:1371–1374
- Potter MA, De Jong K (1994) A cooperative coevolutionary approach to function optimization. *Parallel Problem Solving Nature (PPSN) III*:249–257
- Price K, Storn RM, Lampinen JA (2005a) Differential evolution: a practical approach to global optimization (Natural computing series). Springer, New York
- Price K, Storn RM, Lampinen JA (2005b) The motivation for differential evolution. In: Differential evolution: a practical approach to global optimization (Natural Computing Series). Springer, Berlin, pp 1–36
- Prinz AA, Billimoria CP, Marder E (2003) Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons. *J Neurophysiol* 90:3998–4015
- Rechenberg I (1973) Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog, Stuttgart
- Reid MS, Brown EA, DeWeerth SP (2007) A parameter-space search algorithm tested on a Hodgkin–Huxley model. *Biol Cybern* 96:625–634
- Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. *Evol Comput* 2:221–248
- Swensen AM, Bean BP (2005) Robustness of burst firing in dissociated purkinje neurons with acute or long-term reductions in sodium conductance. *J Neurosci* 25:3509–3520
- Tabak J, Murphey CR, Moore LE (2000) Parameter estimation methods for single neuron models. *J Comput Neurosci* 9:215–236
- Taylor AL, Hickey TJ, Prinz AA, Marder E (2006) Structure and visualization of high-dimensional conductance spaces. *J Neurophysiol* 96:891–905
- Tobin AE, Calabrese RL (2006) Endogenous and half-center bursting in morphologically inspired models of leech heart interneurons. *J Neurophysiol* 96:2089–2106
- Van Geit W, Achard P, De Schutter E (2007) Neurofitter: a parameter tuning package for a wide range of electrophysiological neuron models. *Front Neuroinformatics* 1:1
- Vanier MC, Bower JM (1999) A comparative survey of automated parameter-search methods for compartmental neural models. *J Comput Neurosci* 7:149–171
- Weaver CM, Weame SL (2006) The role of action potential shape and parameter constraints in optimization of compartment models. *Neurocomputing* 69:1053–1057
- Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. *IEEE Trans Evol Comput* 1:67–82