Acta Crystallographica Section D
**Biological
Crystallography**
ISSN 0907-4449

Differential evolution for protein crystallographic optimizations

Duncan E. McRee

ActiveSight and Molecular Images,
4045 Sorrento Valley Boulevard, San Diego,
CA 92121, USA

Correspondence e-mail:
dmcree@active-sight.com

Received 26 January 2004
Accepted 8 October 2004

Genetic algorithms are powerful optimizers that have been underutilized in protein crystallography, given that many crystallographic problems have characteristics that would benefit from these algorithms: non-linearity, interdependent parameters and a complex function landscape. These functions have been implemented for real-space optimizations in a new fitting program, *MIfit*, for real-space refinement of protein models and heavy-atom searches. Some programming tips and examples will be presented here to aid others who might want to use genetic algorithms in their own work.

1. Introduction

Genetic algorithms (GA) include a broad class of optimizers that share several characteristics. A vector of the variables (analogous to genes in natural evolution) of the problem to be optimized is constructed and a randomized population of these vectors is created. The vectors are then scored by the means of a fitness function and the fitter vectors selected to form the next generation. The next generation is formed by mutation and crossover of the parent vectors and if the daughter thus formed is more fit than the parent then the parent is replaced. The whole process is repeated for several generations. After a number of generations the population as a whole becomes more fit (that is, it scores higher in the fitness function) and in a successful outcome individuals appear that will solve the problem. Generally, the best scoring individuals from the last generation are returned as solutions. Genetic algorithms can be differentiated from each other by the means by which they perform the mutation and crossover steps and by the manner in which the variables or ‘genes’ are encoded. In experimenting with genetic algorithms, we have found that the choice of algorithm and the tuning of the algorithm parameters make large differences in the speed and success in solving the problem.

Several previous examples exist of the use of genetic algorithms in protein crystallography. Perhaps the best known and most widely used has been the program *EPMR* written by Kissinger *et al.* (1999). *EPMR* is a molecular-replacement program that optimizes the best values of three translational and three rotational parameters simultaneously using a GA. Our success with this program was responsible for our looking further into using GA for our problems. After much experimentation, we found that a particular GA method termed differential evolution (DE) was the most successful and flexible optimizer for our problems, which included rigid-body fitting in real-space and finding the optimum side-chain rotamers. We have implemented these in the program *MIfit*, which is available from Molecular Images (San Diego, CA, USA; http://www.molimage.com).

2. Differential evolution

Here, we will discuss issues relevant to using DE in common crystallographic applications; a detailed description of DE, other example uses and code samples can be found in the existing literature (Price, 1999; Storm, 1999; Lampinen & Zelinka, 1999; Chisholm, 1999). In DE, the genes are floating-point numbers and this has advantages in scaling and dynamic range over the more traditional bits or integers used in some other GAs. A common difficulty in GA optimization is the determination of the proper size of the mutation step, the amount by which a variable (or gene) is changed between generations. If the steps are too large the solution may be missed and if they are too small it will take many generations to converge. In DE, the size of the mutation step is determined by taking differences between two individuals in the population [and multiplying it by a mutation factor, *F* (Price, 1999), which is used as an overall control variable]. This method has the advantage of providing an adaptive scaling of the mutation as the population matures. In the early stages, the population is diverse and the mutation sizes can be large. If a variable contributes to the score produced by the fitness function, its values within the population will converge towards the best value and the variance in that variable will decrease. If the variable contributes very little or nothing to the score, its range of values will increase with time as mutations are made. This can either continue until values of the variable are found that make significant contributions, or when the maximum generation limit is reached the variable will have a large variance in the population with no dominant values. This information can be very valuable in assessing the final solution reached at the end of a run.

3. Restraints and constraints

Interestingly, the difference method of scaling means that a variable can be constrained to a value by constructing an initial population with no variance in that variable; that is, where all of the members of the population have the same value. In this case, the differences between individuals will always be zero and the mutation size will thus also be zero. Similarly, a variable can have integer values by seeding the population with only whole numbers; all of the differences between those variables will also be whole numbers. Many variables are bounded and can have values only in a specified range. In this case, a function can be supplied to impose these bounds between generations. Restraints, where a variable is restrained to be near a value with a given distribution, are very common in crystallographic refinement of models. For example, bonds and angles are restrained to target values but allowed to vary, a scenario which is more difficult to set up in DE. One strategy is to include this expected variance as part of the fitness function. The best score would be obtained by an individual that has the best fit of positional values to the data and the best geometry. Weighting could also be incorporated into the fitness function by setting an overall weight for geometric restraints *versus* positional values. Another strategy is to impose a direct selection process in between generations where the outliers would be pruned and replaced with new individuals. This has been likened to a predator–prey relationship in nature. A combination of both may produce the best results. In practice, this pruning cannot be too aggressive or the population will never be able to evolve. It will essentially be driven to extinction before it can get started! A good strategy is to add a pruning step every dozen generations or so.

4. Random numbers and probability distributions

It is essential to use a random-number generator suited to the problem. This issue arises because the standard random-number generators included in C and Fortran are very fast but not necessarily well suited to DE. Better random-number generators are available in many code libraries. Two types of random-number generators are useful in different contexts: equally distributed and Gaussian distributed. Equally distributed suffices for many problems, but in many cases a Gaussian distribution in a variable is more desirable and may lead to faster convergence by forming a few outliers that may prove to be closer to the final value when the starting point is far from the final answer. The random-number generator could make use of prior probability information if it is known. For instance, with phase distributions, a range of phases that match the phase probability distribution would be used. Generating correct distributions is key to making the problem well conditioned.

5. Scoring

The fitness function is key to making a GA work. In some cases the scoring may be straightforward, comprising a simple evaluation of the current parameters of an equation. In other cases, the function could be quite complex and essentially form a program within the program. For instance, in one implementation of DE the best parameters for a checkers-playing program are evaluated *via* a pairwise tournament that is held between all of the members of the population and the fitness score is the number of times an individual wins (Chisholm, 1999). In nature, fitness is very involved and for humans includes an incredibly complex range of social interactions and tool-making abilities that are eventually included in an individual's ability to reproduce. In nature, the environment an individual finds itself in is highly variable and is impactful in assessing the fitness of an individual, whereas in the computer the environment is the same for every individual.

6. Crossover

Crossovers are a very powerful component of GA optimizers which distinguish GA methods from other optimization techniques involving random fluctuations such as Monte Carlo or simulated annealing. In some GA algorithms, only crossover is used with mutations held to 0. In this case, all of the variation needed to solve the problem should exist in the initial population. The crossover functions used can vary from a simple swapping of variables to swapping of contiguous groups in an attempt to imitate gene linkage in biological recombination processes. In DE, a simple swapping of values between individuals is normally used. The effect of crossovers is to allow for 'hill-hopping' in complex functions that have a number of local maxima. A simple illustration of this shown in Fig. 1. The overall rate of crossovers are controlled by rolling a random number and comparing it with the crossover frequency, CR, which can be set to a value between 0 (no crossovers) and 1 (always crossover) (Price, 1999).

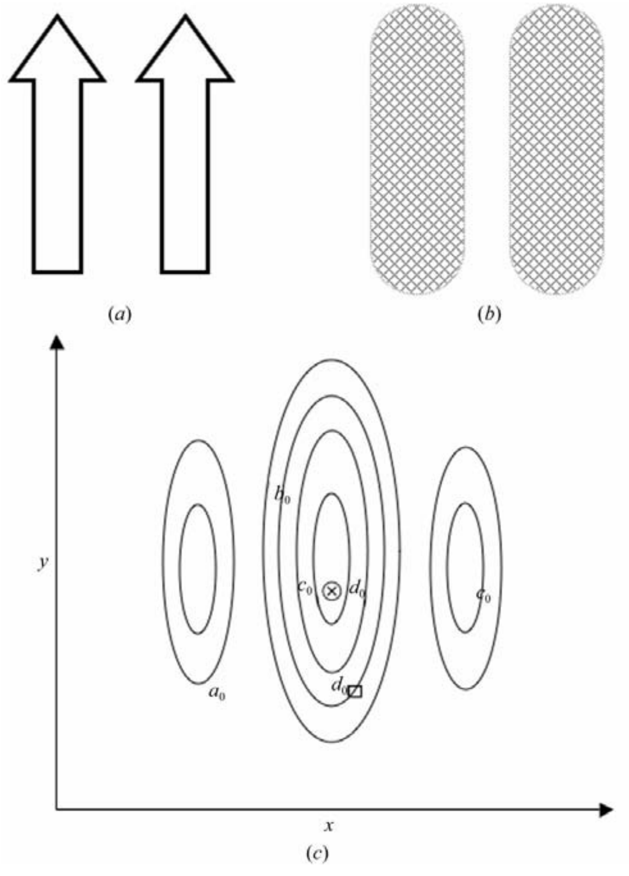

Figure 1
A hypothetical example of a problem with several local maxima. In this problem, a two-stranded ribbon schematic (*a*) is being fit to a medium-resolution density map (*b*), an example of a two-dimensional search over the variables *x* and *y*. (*c*) shows the function landscape corresponding to the correlation between the model and the map over all values of the variables *x* and *y*. The function has three maxima: the middle correct one and two side maxima where only one strand of the two overlaps at about half the height of the center one. To illustrate how crossovers between two vectors in a GA can jump between local maxima or 'hill-hop', imagine that we start a GA run with four vectors, a_0, b_0, c_0 and d_0, each with two genes *x* and *y*, assigned random values over the entire search space as shown in (*c*). If we make a daughter of *c* by crossing over between *c* and *d* such that we take the *y* value from c_0 and the *x* value from d_0, then the daughter that is produced will be very near the correct maxima and will replace the parent, 'winning' over the other vectors.

Table 1
Statistics showing the progress of a typical rigid-body refinement of the protein 7-Fe ferrodoxin into a 13 Å map.

The model consists of 1793 atoms in 108 residues and 165 waters. The DE parameters were six variables (t_x, t_y, t_z, φ_x, φ_y, φ_z), a population size of 120 (20 times the number of variables), a mutation factor *F* of 0.60 and a crossover frequency CR of 0.80. Every fifth generation is listed. The table lists the best and worst scores from the scoring function (see text) and the variance in the six variables as the refinement progresses. $\sigma\varphi_x$, $\sigma\varphi_y$ and $\sigma\varphi_z$ are given in degrees and σt_x, σt_y and σt_z are in ångstroms. The score is the average amount of density at the atom centers of the moving fragment as described in the text. The variances are generated initially at random with an arbitrary target width of 1 Å and 0.25°. As the algorithm searches the first 15 generations, the variances give larger and larger step sizes as the solution is sought until a good solution is found in the 15th generation. The variances then decrease as subsequent generations optimize the solution. Figures showing the model and density corresponding to the start and finish are shown in Fig. 2. The entire run took 2 s on an AMD Athlon XP-M 1600+ 14 GHz processor running *MIfit* under Windows XP. Final values were t_x = −2.09, t_y = 4.87, t_z = 8.08 Å, φ_x = −0.06, φ_y = 0.72, φ_z = −0.05°.

Generation	Best score	Worst score	$\sigma\varphi_x$	$\sigma\varphi_y$	$\sigma\varphi_z$	σt_x	σt_y	σt_z
1	−1.33	−2.78	0.90	0.57	0.76	0.25	0.20	0.23
5	0.27	−2.29	1.67	1.22	1.36	0.28	0.26	0.25
10	2.08	−1.48	1.95	1.61	1.63	0.27	0.27	0.26
15	2.42	−0.92	1.63	2.04	1.51	0.27	0.27	0.28
20	2.48	0.91	1.23	1.43	1.30	0.22	0.21	0.25
25	2.58	1.43	0.90	1.25	1.04	0.18	0.15	0.20
30	2.58	1.98	0.67	0.83	0.76	0.16	0.13	0.15
35	2.61	2.22	0.56	0.63	0.52	0.11	0.10	0.12
40	2.61	2.38	0.41	0.54	0.49	0.08	0.08	0.09

7. Programming example

This example is for optimizing a molecular fragment into an electron-density map. The search has six parameters, three translations in *x*, *y* and *z* and three rotations about the axes of *x*, *y* and *z*. The goal of the optimization is to find the values of these parameters that give the highest correlation with the electron density. The scoring function is the sum of the density found at the atom centers weighted by the atomic weight of the atom. If the resolution is low, densities above 2σ are truncated to prevent overweighting overlapped densities (*i.e.* the center of a ring such as in a phenylalanine residue). Negative densities are heavily penalized by multiplying them by a factor of five. The map density is interpolated by means of a spline function that has been shown to return values within 0.1% of the true value (Lampinen & Zelinka, 1999). A more accurate method for scoring would be to correlate the observed and calculated density. However, constructing the calculated density is a lengthy procedure involving looping over all the atoms over a sphere of density points and would slow the function considerably. Since this function is meant to be interactive in real time, the approximation has been found to be a better compromise between speed and accuracy.

To set the parameters for mutation-step size, the program first calculates the scoring function for the starting solution. If it is high, meaning that the model is a fair match to the density, then the mutation-step size is set to be smaller under the premise that the user only needs to optimize an already good solution, otherwise the mutation-step size is broadened so that a larger volume is searched for the solution. The translational variance target is set to 0.25 Å and the rotation target to 0.75°.

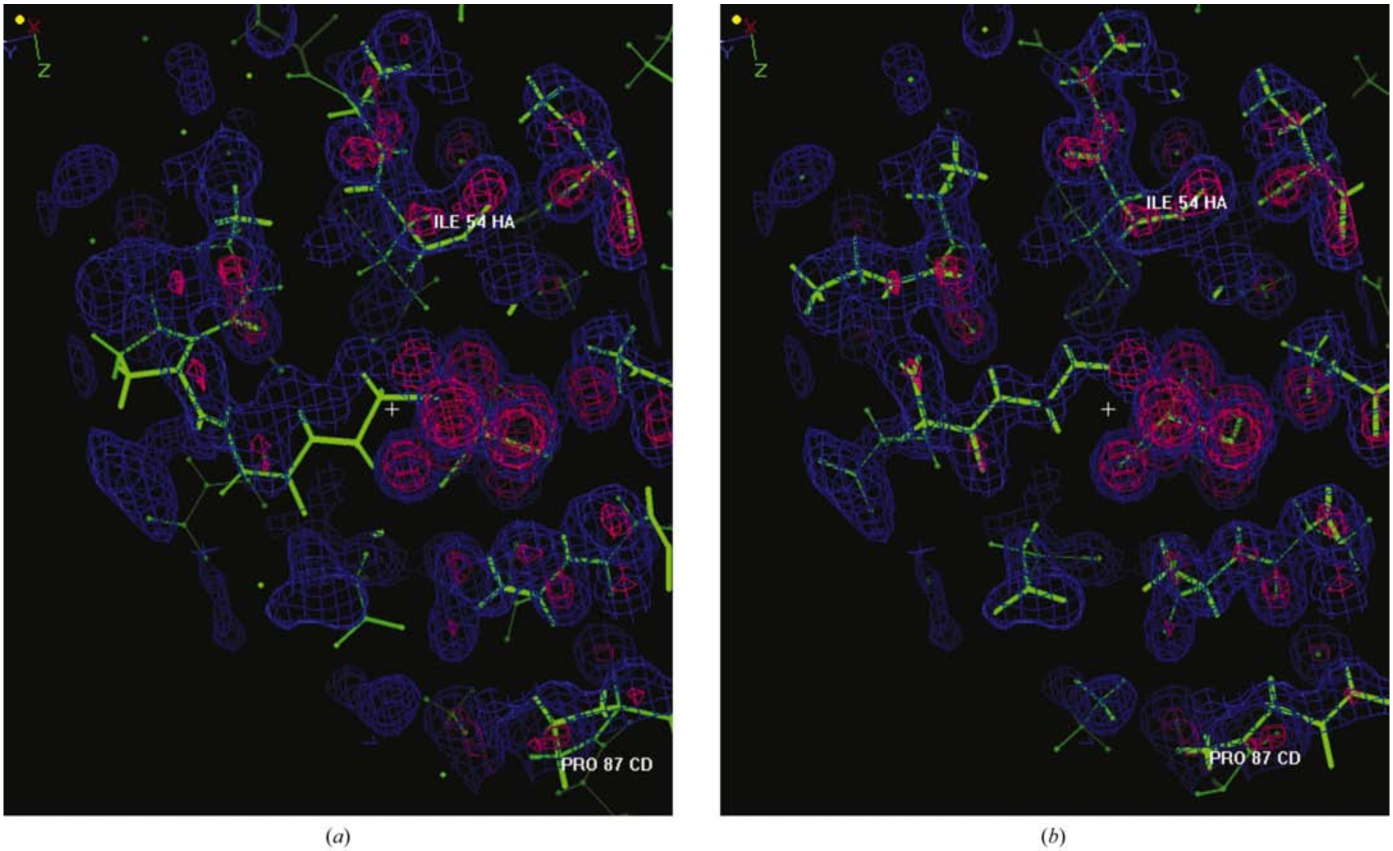

Figure 2
Maps corresponding to the start (*a*) and finish (*b*) of the real-space rigid-body refinement by the differential evolution run shown in Table 1.

The crossover frequency CR (Price, 1999) is set to 0.8 and the mutation factor *F* (Price, 1999) is set to 0.60. The population is set to 120 vectors. Each vector has six members, three translations and three rotations, and the function is run for 40 generations. These numbers were found by trial and error to be enough generations and a large enough population to lead to good convergence. In each generation, a trial daughter is constructed from each of the parent genes by crossover and mutation. If the daughter has a better score than the parent then the parent is replaced; otherwise, the parent is kept. After the 40 generations the best solution is applied to the model, the screen is updated and the score reported to the user. The results of a typical run are shown in Fig. 2 and Table 1. All this takes a fraction of a second for most fragments of a few residues and about a minute for an entire molecule of about 200 residues. The equivalent brute-force algorithm is 24-fold slower than the DE algorithm with only seven steps in each variable, which requires scoring 7^6 or 117 649 trials, whereas the DE algorithm scores 686 trials. The final solution is also sampled on a finer scale for the DE algorithm since the last few generations take very small step sizes as virtually all of the population has converged to the same values. In comparing with a least-squares procedure it must be remembered that the genetic algorithm takes longer. However, many of the problems solved easily by DE would not be possible with least squares, which is only able to converge to the nearest local minimum of a target function. In cases where the starting position of the model does not overlap sufficiently with the density, no solution would be found. Thus, one of the chief advantages of the method is that it has a larger radius of convergence while remaining faster than a brute-force search over all possibilities.

References

Chisholm, K. (1999). *New Ideas in Optimization*, edited by D. Corne, M. Dorigo & F. Glover, pp. 147–158. London: McGraw–Hill.

Kissinger, C. R., Gehlhaar, D. K. & Fogel, D. B. (1999). *Acta Cryst.* D**55**, 484–491.

Lampinen, J. & Zelinka, I. (1999). *New Ideas in Optimization*, edited by D. Corne, M. Dorigo & F. Glover, pp. 127–146. London: McGraw–Hill.

Price, K. V. (1999). *New Ideas in Optimization*, edited by D. Corne, M. Dorigo & F. Glover, pp. 79–108. London: McGraw–Hill.

Storm, R. (1999). *New Ideas in Optimization*, edited by D. Corne, M. Dorigo & F. Glover, pp. 109–125. London: McGraw–Hill.