A Local Linear Embedding Module For Evolutionary Computation Optimization

Fabio Boschetti
CSIRO, Australia

Journal of Heuristics, in print

Corresponding author: Fabio Boschetti
Research Scientist - CSIRO, Marine and Atmospheric Research/
Tel ++ 61 8 9333 6563 Fax ++ 61 8 9333 6555
Postal address: CSIRO CMAR, Private Bag 5, Wembley WA, 6913
Fabio.Boschetti@csiro.au

Abstract

A Local Linear Embedding (LLE) module enhances the performance of two Evolutionary Computation (EC) algorithms employed as search tools in global optimization problems. The LLE employs the stochastic sampling of the data space inherent in Evolutionary Computation in order to reconstruct an approximate mapping from the data space back into the parameter space. This allows to map the target data vector directly into the parameter space in order to obtain a rough estimate of the global optimum, which is then added to the EC generation. This process is iterated and considerably improves the EC convergence. Thirteen standard test functions and two real-world optimization problems serve to benchmark the performance of the method. In most of our tests, optimization aided by the LLE mapping outperforms standard implementations of a genetic algorithm and a particle swarm optimization. The number and ranges of functions we tested suggest that the proposed algorithm can be considered as a valid alternative to traditional EC tools in more general applications. The performance improvement in the early stage of the convergence also suggests that this hybrid implementation could be successful as an initial global search to select candidates for subsequent local optimization.

Key Words: Evolutionary Computation, Locally Linear Embedding, Optimization.

Evolutionary Computation (EC) is often used as global search method in real world optimization problems. This is due to the fact that many real world optimization problems are nonlinear and result in multimodal objective functions. In such applications, local optimization methods, (e.g., matrix inversion, steepest descent, conjugate gradients) which are prone to trapping in local minima, have limited success.

Given some (usually measured) target data, an optimization problem seeks to reconstruct the unknown parameters that, via the use of an appropriate function, can generate a good approximation to the target data. This function (called a 'forward model') is expected to reliably mimic the process generating the data. Thus, the forward model represents a mapping between the problem's parameter space and the data space. Any optimization algorithm tries to find the unknown parameters (the 'solution' to the optimization problem) via a trial and error sampling of the parameter space, through repeated use of the forward model function. This is required because, in most real-world problems, a direct mapping between data space and parameter space is not available (see Figure 1).

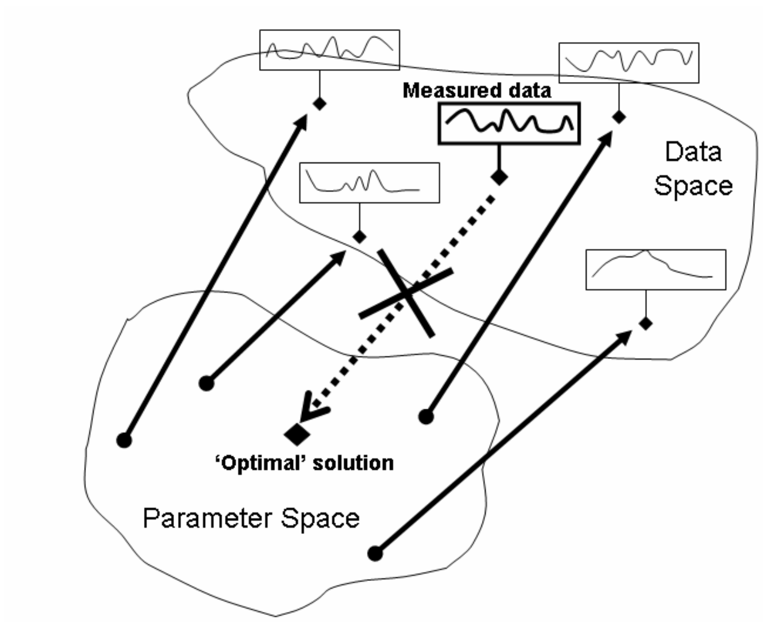


Figure 1. Sketch describing the general concept behind any optimization problem. A function directly mapping the measured data to the optimal solution does not exist. Consequently, the parameter space is searched by trial and error, iterating on a function that maps different parameter combinations into the data space.

The large dimensionality involved in many real world problems can make such trial and error search computationally very expensive. In this paper we propose a hybrid technique, which works by attempting, at each EC generation, to reconstruct an approximation of the direct mapping between data space and parameter space. It achieves this by using the Local Linear Embedding technique. The approach uses the stochastic sampling of the parameter space, inherent in an EC process, in order to approximate, and progressively refine, such a mapping. The rationale is that, by using such approximate mapping, at each EC generation, we can map the 'measured data' directly back into the parameter space, whereby recovering good approximation to the global solution, which is then inserted into the EC population. Our results suggest that this approach can considerably shorten the length of the trial and error process.

In general, the data and parameter spaces of an optimization problem do not necessarily have the same dimensionality. Accordingly, for the approach to be of general applicability, any mapping between the two spaces has to account for such potential difference. Within the machine learning and image processing communities, a number of recently developed algorithms perform approximate mappings between spaces of potentially different dimensionality (Tenenbaum et al., 2000, Balasubramanian et al., 2002, Roweis and Saul, 2000, Saul and Roweis, 2000, Kouropteva et al., 2002, Donoho and Grimes, 2003). Unlike traditional techniques, such as principal component analysis (PCA) (Jolliffe, 1986) or multi-dimensional scaling (MDS) (Cox and Cox, 1994), these methods assume only a locally linear (rather than globally linear) approximation between data points. Local linear embedding (LLE) is one such technique that has the desirable property of fast computation, a non-iterative scheme, and a guarantee of optimal convergence (Roweis and Saul, 2000, Saul and Roweis, 2000). LLE recovers satisfactory mappings between spaces of potentially different dimensionality in several real-world, non-linear problems (Kouropteva et al., 2002). In this work, a similar idea is used to model an approximate mapping between data space and parameter space in optimization problems. The aim is to map the 'measured data' back into parameter space by using local information, obtained via the stochastic sampling of the parameter space by evolutionary algorithms, such as a genetic algorithm or a particle swarm optimization. Such a mapping is expected to approximate the location of the 'optimal' solution in the parameter space. The mapping improves along with the convergence of the evolutionary algorithm.

We tested the proposed algorithm on a large number of standard benchmark problems and on two real-world problems. The test function set includes very different optimization problems, of varying difficulty and of dimensionality ranging from 11 to 200. The consistent positive results are extremely encouraging and suggest that the LLE module can be improve the performance on EC search in more general applications.

Problem Formulation

Let's assume we have a set of *N* observed measurements $\vec{X}^{i}_{measured}, i = 1...N$. These may be physical measurements obtained from natural systems or some man made industrial process. Let's suppose we have a function *F* which models within a satisfactory approximation the physical/industrial process which generates the data. The output of *F* depends on a number *n* of continuous parameters or initial conditions $\vec{x}_j, j = 1...n$. Such parameters are the unknown in our problem and the purpose of the optimization problem is to recover them by using the measured data $\vec{X}_{measured}$ and the function *F*. In the rest of the paper we call data space the space of the $\vec{X}$ and parameter space the space of the $\vec{x}$.

If we could determine an inverse function F^{-1} the problem would be simply solved by applying

$\vec{x}_{solution} = F^{-1}(\vec{X}_{measured})$.

Unfortunately, as mentioned in the Introduction, most real world problems do not allow F^{-1} to be determined either analytically or algorithmically. Consequently, we are forced to tackle the problem in terms of optmisation.

Let's call $\vec{X}_{calculated} = F(\vec{x})$ the approximation to the observed measurements obtained by applying the model *F* to a set of initial conditions $\vec{x}$. We aim to recover the unknown $\vec{x}_{solution}$ by minimizing some measure of the misfit between $\vec{X}_{calculated}$ and $\vec{X}_{measured}$.

In many applications, a simple measure of such misfit is given by

$$M(\vec{x}) = \left|\vec{X}_{measured} - F(\vec{x})\right|^{k} = \sum_{i=1..N}\left|X^{i}_{measured} - X^{i}_{calculated}\right|^{k} \qquad \text{Eq. 1}$$

where, usually, *k* takes the value 1 or 2. Other expressions of the misfit between $\vec{X}_{calculated}$ and $\vec{X}_{measured}$ can also be found.

We employ EC to search for the global minimum of this optimization problem. Additionally, at each generation of the EC run, we attempt to generate an approximation F^{-1}_{gen} to F^{-1} (where *gen* is the current EC generation) which is valid only locally, in the vicinity of $\vec{X}_{measured}$. Once such local approximation is obtained we calculate

$$\vec{x}_{gen} = F^{-1}_{gen}(\vec{X}_{measured})$$

as our current best guess at the target $\vec{x}_{solution}$. This is then inserted into the EC population. We attempt to reconstruct F^{-1} with the use of the Local Linear Embedding algorithm, which we describe next.

Out approach can thus be briefly described as follow (more details are given below):

1) initiate a EC optimization search;
2) at each generation, choose the EC individuals which best fit the measured data;
3) on this individuals, calculate the current approximation F^{-1}_{gen} to F^{-1} via the LLE method;
4) calculate $\vec{x}_{gen} = F^{-1}_{gen}(\vec{X}_{measured})$;
5) add $\vec{x}_{gen}$ to the current EC population, if its misfit is better than that of the worst individual in the current population;
6) repeat 2-5, until number of allowed function evaluation is reached.

We should note another significant difference between the approach we propose and other common implementations of optimization algorithms. It lies in the use of the full vector of measured data ($\vec{X}_{measured}$) and outputs of the forward model ($\vec{X}_{calculated}$), rather than just a single scalar misfit value $M(\vec{x})$. This difference is important, since a significant amount of information about the search space is lost in collapsing the *N*-D difference between the measured and calculated data into a single number.

The Local Linear Embedding Algorithm

The purpose of the LLE algorithm is to map points from one high-dimensional space into another one, possibly of different dimensionality. The idea underlying the LLE approach is very simple. LLE finds local representations of data by expressing each data point in the original space as a linear combination of neighboring points. The weights of the linear combination for each point become the local coordinate system for the representation. LLE then maps points into a different space in such a way that the coordinates of each point in the new space are a linear combination of the same neighboring points, with the same weights as calculated in the original embedding (Figure 2). The result is a mapping that respects local relations between neighboring points. Linearity is imposed only locally, not on the global mapping. Consequently, LLE is expected to map curved manifolds with an acceptable approximation.

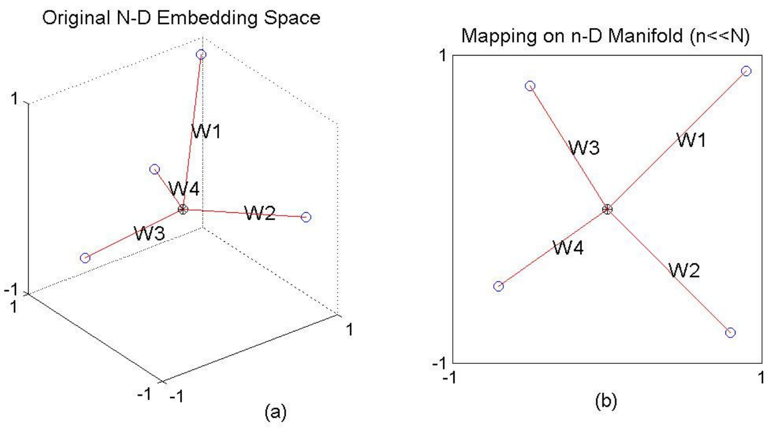


Figure 2. Basic concept behind the LLE approach. In the original *N*-D embedding space, a point (black, in the center) is expressed as a linear combination of neighboring points (four, in this case). The weights *W* for the linear combination are stored. (b) The point is mapped into an *n*-D target space ($n \neq N$), in such a way that it can be expressed as a linear combination of the same neighbors, with weights approximately equal to the original *W*.

The following brief description of the LLE algorithm is supplemented by greater detail in Roweis and Saul (2000), Saul and Roweis (2000), and Kouropteva et al. (2002). Assuming *m* points embedded in the original *N*-D space, with coordinates *X*, LLE seeks to map the points into an *n*-D target space, with coordinates *x*, where *n* and *N* may or may not be equal. The algorithm can be divided into three steps:

1) In the original space, define the neighborhood of each point by calculating distances between each pair of points. For a point *i*, the neighborhood $\vec{X}_i^j, j = 1..K$ can then be defined either as the set of *K* closest points or as the set of points within a certain radius. Standard implementations use the first option;
2) for each point, determine the weights that allow the point to be represented as a linear combination of the *K* points in its neighborhood. This involves minimizing the cost function:

$$C(W) = \sum_{i=1...m} \left| \vec{X}_i - \sum_{j=1...K} W_{i,j} \vec{X}_i^j \right|^2 , \qquad \text{Eq. 2}$$

where $i = 1...m$ represent the points in the original *N*-D space and $j = 1...K$ are the neighboring points. The minimization of Eq. 2 is carried out under the constraint

$$\sum_{j=1...K} W_{i,j} = 1 \text{ and } W_{i,j} = 0 \text{ if } X_i \text{ is not a neighbor of } X_j . \qquad \text{Eq. 3}$$

The solution to Eq. 2, under the constraint of Eq. 3, is invariant to rotation, rescaling, and translation (Saul and Rowins, 2003, pp. 124,131). This is an important feature of the LLE algorithm. It means that the weights *W* do not depend on the local frame of reference. They represent local relations between data points, expressed in a frame of reference that is valid globally.

3) Map the *m* points into the target space with coordinates *x*. This is achieved by minimizing the cost function

$$E(x) = \sum_{i=1...m} \left| \vec{x}_i - \sum_{j=1...K} W_{i,j} \vec{x}_i^j \right|^2 , \qquad \text{Eq. 4}$$

(where the weights *W* obtained from Eqs. 2 and 3 are kept fixed, conserving the local relation between neighboring points) under the constraints

$$\sum_{i=1...n} \vec{x}_i = \vec{0} , \qquad \text{Eq. 5}$$

which centers the *x* coordinates around zero and imposes translational invariance, and

$$\frac{1}{m} \sum_{i=1..n} \vec{x}_i \vec{x}_i^T = 1 , \qquad \text{Eq. 6}$$

which imposes rotational invariance (see Saul and Rowins, 2003, p. 134). This equates to finding some global coordinates *x*, over the target space, that conserve the local relations between neighboring points in the original space. Each individual point $\vec{x}$ is obtained only from local information within its neighborhood. Importantly, the overlap between neighborhoods generates a global reference. Solving Eq. 4 is the most delicate and computationally intensive part of the LLE algorithm. Fortunately, it can be achieved without an iterative scheme. Via the Rayleitz-Ritz theorem (Horn and Johnson, 1990), this reduces to finding *n* eigenvectors of the matrix

$$\mathrm{M}_{\mathrm{i,j}} = \partial_{\mathrm{i,j}} - \mathrm{W}_{\mathrm{i,j}} - \mathrm{W}_{\mathrm{j,i}} + \sum_{K} \mathrm{W}_{\mathrm{k,i}} \mathrm{W}_{\mathrm{j,k}} \quad . \qquad \text{Eq. 7}$$

The *n* eigenvectors correspond to the *n* smallest non-zero eigenvalues of M (Saul and Rowins, 2003, pp. 134-135).

The overall LLE algorithm only involves searching for closest points and performing basic matrix manipulations, which can easily be implemented via standard computational tools (MatLab, Numerical Recipes, LAPACK, Mathematica, etc.).

An example of LLE performance is given in Figure 3. In Figure 3a we see the ‘swiss roll’ data set, a standard test for dimensionality reduction problems (see, for example Roweis and Saul, 2000). It represents a challenging test because no lower dimensional projection allows to respect the topological relationships among the data point. The purpose of the dimensionality reduction exercise in this case is to ‘unroll’ the data set and stretch it over a 2D plane, in such a way that points close to one another along the ‘roll’ are also close on the plane. The low dimensionality of the problem allows the result to be analyzed visually. In Figure 3b we see the mapping performed by the LLE. As we can see with the help of the gray-scale tones, the dark point are mapped close to one another, as do the light color ones. An approximate ‘unrolling’ has been achieved. The stretch, which results in uneven distribution along the *y* axis, is a known problem in LLE applications (the analysis of which is beyond the scope of this work) and is mostly due to numerical accuracy in the solution of Eq. 4. Nevertheless, the main topological relation between data points is well respected. Notice also that the ‘swiss roll’ problem is a particularly challenging one, and we expect structures in the data space of real world problems to be distributed on ‘easier’ manifolds.

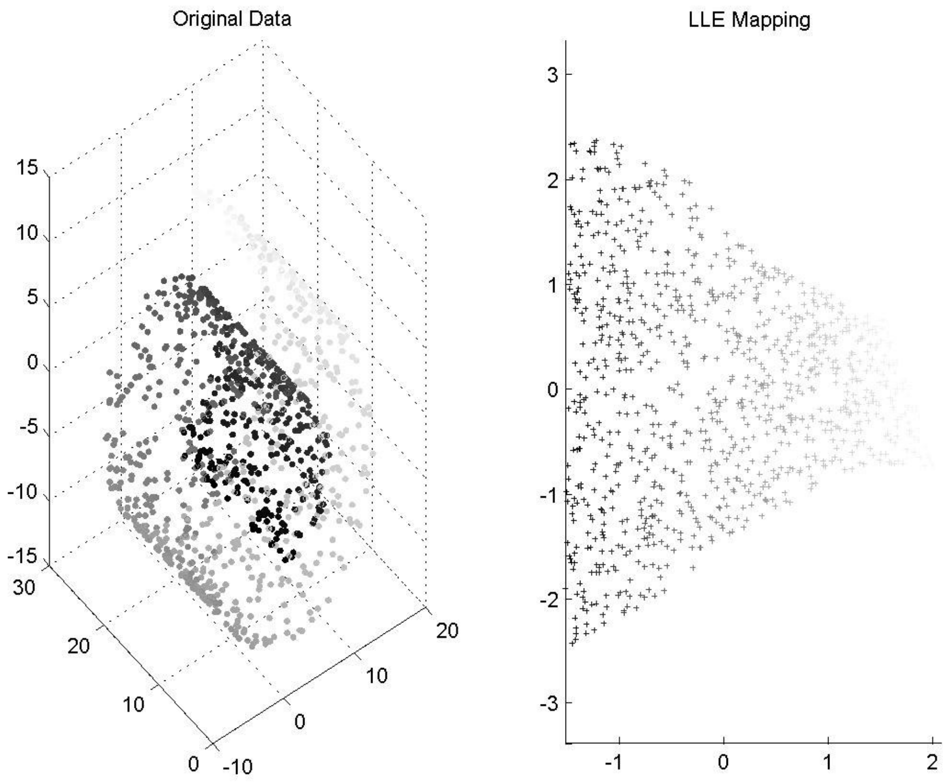


Figure 3. Example of LLE mapping of the ‘swiss roll’ data. Original data (a) and their mapping into a 2D plane (b). Topological relation between points are respected, as can be seen from the gray tones.

Evolutionary Computation Algorithms

Under the class of EC techniques we find several algorithms like Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming, Particle Swarm Optimization, Memetics Algorithms. Each algorithm also comes in different variations, different implementation and hybridizations.

In this work we test two algorithms: a real coded Genetic Algorithm (GA) and a Particle Swarm Optimization (PSO) code. Both algorithms are described in this section. In the rest of the document, they will be referred to as standard GA and standard PSO, while their hybrid counterparts which include the LLE module will be referred to as GA-LLE and PSO-LLE.

It is important to emphasize that the hybridization of such codes with the LLE does not depend on any specific detail of the GA or PSO algorithms here presented. The LLE acts on the sampling of the data space performed by the EC algorithms and produces its mapping into the parameters space before the next EC generation. In other words, the

LLE mapping does not interfere with the intergeneration functioning of the EC algorithms. According, its use can be easily extended to other EC strategies not covered in this work.

Genetic Algorithms. GAs work by mimicking biological evolution. Their rationale is that biological species solve very complex non linear problems in order to adapt to their environment and that a combination of selection, reproduction and mutation can also be successful in solving numerical problems. The real-coded GA used below has been extensively applied to a number of high-dimensional, highly non-linear geological, geophysical, and geo-mechanical problems (Wjins et al, 2003, Boschetti and Moresi 2001). For a full description of the algorithm we refer the reader to Boschetti et al., (1996). Here we summarise its main features:

1) the unknowns are coded directly into the chromosome as real numbers without making use of binary or gray code representations;
2) we use linear normalization selection (Davis, 1991; Goldberg, 1989) as selection operator. In linear normalization selection, an individual is ranked according to its fitness, and then it is allowed to generate a number of offspring proportional to its rank position. Using the rank position rather than the actual fitness values avoids problems that occur when fitness values are very close to each other (in which case no individual would be favored) or when an extremely fit individual is present in the population (in such a case it would generate most of the offspring in the next generation).
3) we use uniform crossover with a crossover rate of 0.9. We implemented uniform crossover in the following way: two individuals are chosen randomly and a random number *n* is selected; then *n* random gene locations are chosen and the floating point values of the parameters at such locations are swapped between the two individuals.
4) we use random mutation with a mutation rate of 0.01;
5) at each generation the best individual is copied in the next generation (elitism).

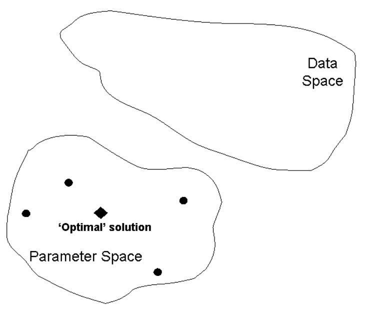
Particle Swarms Optimisation. The PSO algorithm works by mimicking the coordinated behavior of insects in their search for food. Information about the promising areas of the search space is communicated to all agents in the swarm by affecting the movement of the agents and attracting them toward such locations. The algorithm can be summarised as follows:

1) a population of agents is randomly initialized in the search space and is given a random initial direction of motion;
2) at each generation an agent follows its current direction and samples the space at the new location;
3) it is then assigned a new direction and speed of motion according to a) its current direction, b) whether the currently sample point has better fitness than the previous one and c) the current location of the agent with the best fitness value.

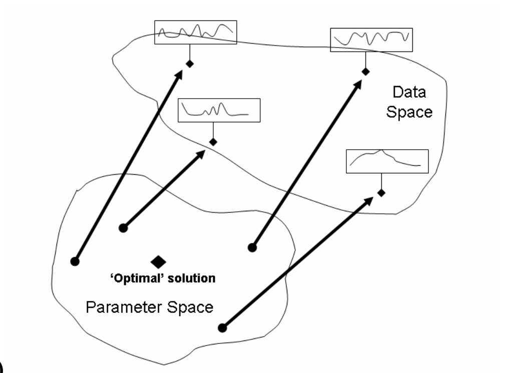
This coordinated behavior allows the swarm of agents to converge towards the areas in the search space with best fitness while still exploring adjacent areas. More detailed information about the specific code used can be found in Mouser and Dunn (2004), where some applications to real world problems are also described.

LLE Module in EC optimisation

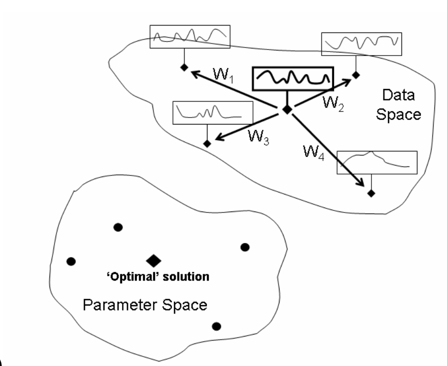
The concept underlying LLE can be used within an EC algorithm. The procedure is identical for both the GA and PSO and is described with the help of Figure 4.



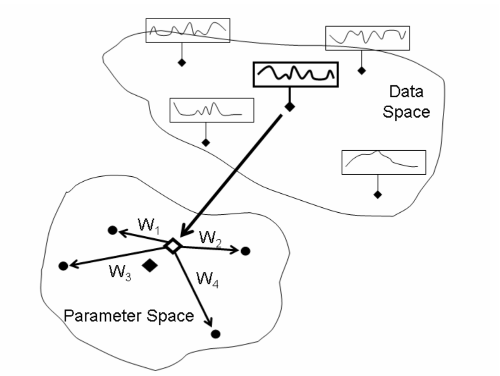
(a)



(b)



(c)



(d)

Figure 4. Sketch describing the LLE module. A black diamond marks the (unknown) optimal solution. (a) A number of random points in the parameter space is chosen by the EC. (b) These points are mapped into the data space via the forward model. (c) The measured data vector is expressed in the data space as a linear combination of neighboring points, and the weights *W* of the linear combination are stored. (d) An approximation to the optimal solution is found in the parameter space by a linear combination of neighboring points, using the same weights *W*.

It can be summarized in the following way:

1) run the EC for one generation (i.e., generate a number of individuals, and for each individual, run the forward model; see Figure 4a). For each forward model run (i.e., for each output from the EC population), store not only the (single-valued) scalar misfit measure (*M(x)*), but also the full vector of data values output by the forward model ($\vec{X}_{calculated}$), as shown in Figure 4b.

2) In the *N*-D *data* space, generate the neighborhood of $\vec{X}_{measured}$ within the set of $\vec{X}_{calculated}$, i.e., look for the *K* closest points to $\vec{X}_{measured}$. Call this set $\vec{X}_{neigh}$.
3) Express $\vec{X}_{measured}$ as a linear combination of $\vec{X}_{neigh}$, by calculating the weights $W_{measured}$ via Eqs. 2 and 3 (Figure 4c).
4) In the *n*-D *parameter* space, calculate the approximation to the target solution (global minimum) $\vec{x}_{gen}$ as a linear combination of the *n*-D points $\vec{x}_{neigh}$ (which correspond to the *N*-D $\vec{X}_{neigh}$) via the weights $W_{measured}$, using Eqs. 4-6. The point $\vec{x}_{gen}$ is the current mapping of the measured data back into the parameter space (Figure 4d) (that is the current guess to the global optimum $\vec{x}_{solution}$).
5) Evaluate the single-valued scalar misfit for $\vec{x}_{gen}$. If the misfit is lower than the misfit of the worst individual in the EC population, replace the worst individual with $\vec{x}_{gen}$.
6) Repeat steps 1-5 until acceptable convergence.

The following remarks are worth noting:

1) In most test cases, the point found by the LLE module has a considerably smaller misfit than that of the best individual in the EC population, which results in considerably improved performance.
2) The standard LLE algorithm aims to reconstruct a local linear approximation in the neighborhood of each point in the domain. Its use within an optimization problem need only map one point ($\vec{X}_{measured}$) back into the parameter space. Accordingly, Eqs. 2-6 need to be applied only to the matrix including the neighborhood of the measured data $\vec{X}_{measured}$. The computational time involved in the process is consequently negligible, compared to the overall optimization.
3) The performance of the LLE algorithm depends on the choice of the neighborhood size *K*. A number of heuristic methods have been proposed to determine the optimal choice (Kouropteva et al., 2002). The test cases below use an LLE module with different choices of *K* (usually *K*=7,8,…15), and the *K* resulting in the best misfit is chosen.
4) In relatively small dimensional problems is may happen that *K>D*. In this case the local reconstruction weights are no longer uniquely defined. Regularization must then be used to deal with the degeneracy. A simple regularizer is to favor weights W that are uniformly distributed in magnitude in Eq. 2.

Test Cases

We performed a number of tests on 13 benchmark functions taken from two sets of standard, publicly available test functions, and on two real-world applications, in order to compare the performances of a standard GA and PSO versus the same algorithms including the LLE module.

Standard Test Functions

The performance comparison uses 13 benchmark functions (Table 1). Six functions have been selected among standard benchmark functions for EC studies. These are 1) De Jong's function 2, 2) Rastrigin's function, 3) Schwefel's function, 4) Griewangk's, 5) Sum of different power, and 6) Ackley's Path function. They correspond to functions 2 and 6-10 of the "GEATbx: Genetic and Evolutionary Algorithm Toolbox" benchmark functions (see http://www.geatbx.com/docu/index.html for a description and implementation of the functions).

In order to use these functions for our tests, we need to slightly modify their implementation. For each of these functions, the scalar misfit measure is a properly designed combination of some higher dimension function of the inputs. This higher dimension function corresponds to $X_{measured}$ in our notation. The use of the LLE module requires that we have knowledge of a set of 'measured data' ($X_{measured}$), not just a scalar misfit function. For each function, we generated such set by evaluating the function on the known global minimum and stored the output vector as 'measured data' for later use in the tests.

The other 7 functions are taken from the Moré, Garbow, and Hillstrom (Moré et al, 1981) collection of standard test functions for global optimization problems (for a description and implementation of these functions ftp://ftp.mathworks.com/pub/contrib/v4/optim/uncprobs/). The Moré, Garbow, and Hillstrom functions have been chosen because they are high dimensional (the dimensionality of some of them can be changed arbitrarily). This allowed us to test the algorithms on problems ranging from 11 to 200 dimensions. For all of these functions, the misfit measure matches Eq. 1 with k=2.

Table 1. Four test functions used in the performance comparison (Moré, Garbow, and Hillstrom collection).

Test problem	Moré, Garbow, and Hillstrom function name	Param. Space Dim.	Global Minimum
Problem 1	Osborne Function n 2	11	0.0401
Problem 2	Discrete Boundary Value	200	0
Problem 3	Broyden Tridiagonal	100	0
Problem 4	Discrete Integral Equation	80	0
Problem 5	Trigonometric	150	0
Problem 6	Broyden banded function	80	0
Problem 7	Penalty 2	10	2.9604e-004
	GEATbx function name		
Problem 8	Rosenbrock function (De jong function 2)	10	0
Problem 9	Schwefel function	10	-4189.8
Problem 10	Rastrigin function	20	0
Problem 11	Griewangk function	10	0
Problem 12	Sum of different powers	30	0

Problem 13	Ackley's path function	20	0

In order to account for stochastic variability, and the effect of different population sizes and convergence length, the comparison have been performed under the following conditions:

1) we used three different population sizes of 30, 50, and 100 individuals;
2) for each test, 20 runs are performed, initialized with different random seeds, in order to account for the stochastic variability inherent in EC. Performance are evaluated in terms of averages and standard deviations of the 20 runs;
3) for each benchmark function, we compare the performance for the codes at two stages along the convergence. First, after a number of function evaluations equal to 20 times the population size, and then at the end of the run, that is, after a number of function evaluations equal to 100 times the population size. We account for function evaluations, rather than number of generations, in order to compensate for the extra function evaluations of the individuals generated by the LLE module;
4) on each scenario, we compare the standard GA versus the GA-LLE module and the standard PNO versus the PNO-LLE module;
5) the statistical significance of the comparisons is evaluated via a standard Mann-Whitney U-test.

Different population sizes are tested at different times during the EC convergence in order to mimic a range of different scenarios going from: a) being constrained by a very high computational cost for the function evaluation (which may force the use of a very small population size and short run), to b) using a very fast function evaluation, and, consequently, being able to afford a full EC convergence with large populations.

In Figure 5 we see some examples of converge curves (averaged over the 20 runs) for the GA (dashed lines) and GA-LLE (solid lines) with population size of 100. In most test cases the GA-LLE clearly outperforms the standard GA. In case of benchmark functions 2, 3, 4, 5, 6 and 13 the different in performance is considerable since the very early generations. In the other functions the difference is less noticeable and a numerical analysis is required. The full results for all benchmark functions are given in Tables 2-7. Here, for each benchmark function, we show the average results over 20 runs and the statistical significance from the Mann-Whitney U-test, after a number of function evaluations equal to 20 and 100 times the population size.

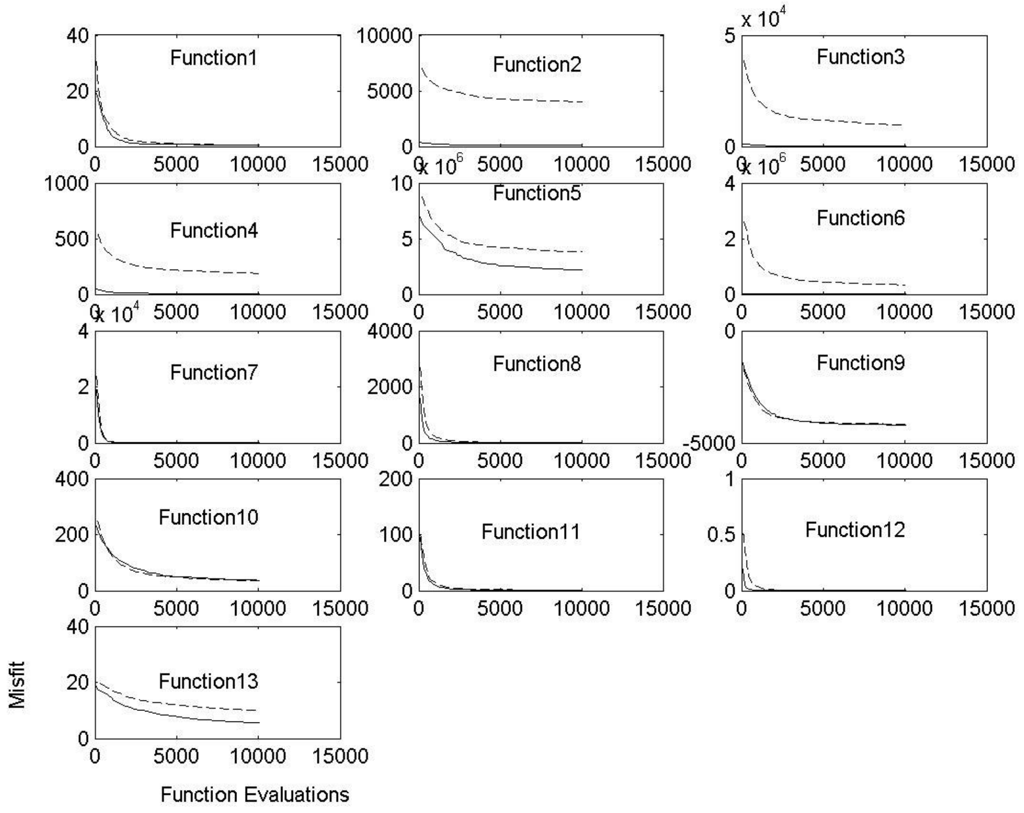


Figure 5. Example of convergence of the GA-LLE (solid lines) and standard GA (dashed lines) for a population size of 100 for the 13 benchmark functions. All curves are averaged over 20 runs.

Three main results are noticeable in the comparison (Tables 2 to 7):

1) In almost all tests, the EC with an LLE module outperforms the traditional EC (for both GA and PNO) both in the early stage and at the end of the convergence; in some cases, the difference in performance is considerable. The only exceptions are a) in the early optimization of function 9 with the GA (while the GA-LLE again performs better at the end of the convergence), b) function 8 for PNO optimization, again for a population size of 30, at the end of the convergence;
2) although, in general, the PNO-LLE seems to perform better, the LLE module improves the performance of both the GA and PSO in a comparable manner. In other words, the improvement due to the LLE module does not appear to depend on the chosen EC algorithm;
3) in the vast majority of tests, the Mann-Whitney U-test show significant difference between the convergence values obtained in the 20 runs of the EC-LLE versus the standard EC, as measured by $p<0.05$ as for standard convention. (Recall that p gives an estimate of the probability that the two groups of 20 convergence results are taken from the same sample population. A very low (<0.05) value rejects such null hypothesis, suggesting instead that the two groups belong to two different

sample populations.) Notice also that, among the cases in which the EC-LLE actually performs more poorly that the standard LLE, only 5 of them are statistically significant according to the Mann-Whitney U-test;

4) both GA-LLE and PNO-LLE show good performance improvement in the early stages of the convergence. This may suggest that the LLE module is beneficial in short EC runs, in order to locate areas in the parameter space that are deserving of a more focused, local search.

Table 2. Comparison between standard GA and a GA-LLE for a population size of 30 individuals. For each benchmark function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	GA Population size = 30					
	Fun. Eval = 20*pop. Size			Fun. Eval = 100*pop. Size		
	GA-LLE	GA	**T-test P**	GA-LLE	GA	**T-test P**
Function 1	2.47	5.53	**1.6e-3**	0.62	1.16	**5.6e-4**
Function 2	192.5	5140.1	**1.5e-21**	124.4	4000.53	**6.29e-26**
Function 3	450.5	1.79e4	**1.2e-19**	187.0	1.09e4	**1.50e-19**
Function 4	14.27	311.4	**2.1e-23**	6.77	204.50	**7.99e-22**
Function 5	5.10e6	5.62e6	**5.0e-3**	3.97e6	4.06e6	**0.51**
Function 6	1116.8	8.8e5	**2.5e-18**	317.7	3.77e5	**1.86e-15**
Function 7	20.03	175.1	**2.2e-2**	3.10e-4	0.14	**0.028**
Function 8	49.08	162.4	**3.6e-5**	6.99	34.19	**2.42e-4**
Function 9	-3125.1	-3385.2	**5.8e-4**	-4079.4	-4064.7	**0.53**
Function 10	77.71	118.26	**2.90e-9**	29.52	50.67	**9.63e-8**
Function 11	4.19	9.68	**3.2e-5**	0.99	1.55	**6.36e-7**
Function 12	2.3e-3	0.02	**7.6e-5**	9.34e-7	7.92e-4	**2.11e-3**
Function 13	13.41	15.90	**3.46e-9**	6.13	10.73	**1.44e-13**

Table 3. Comparison between standard GA and a GA-LLE for a population size of 50 individuals. For each benchmark function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	GA Population size = 50					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	GA-LLE	GA	**T-test P**	GA-LLE	GA	**T-test P**
Function 1	2.17	3.85	**0.017**	0.47	0.74	**1.9e-3**
Function 2	168.1	5046.4	**1.22e-24**	120.6	3901.3	**1.57e-23**
Function 3	370.5	1.60e4	**2.79e-19**	167.8	1.01e4	**3.85e-18**
Function 4	13.00	292.7	**2.97e-22**	6.47	190.3	**4.82e-20**
Function 5	4.88e6	5.35e6	**5.13e-3**	2.97e6	3.95e6	**2.89e-8**
Function 6	729.4	7.71e5	**4.91e-16**	275.46	3.49e5	**2.21e-15**
Function 7	9.58	56.47	**9.95e-5**	3.02e-4	0.027	**8.89e-5**

Function 8	42.26	118.7	**1.85e-6**	9.16	24.31	**8.6e-3**
Function 9	-3444.2	-3646.7	**1.1e-3**	-4170.0	-4113.2	**6.85e-6**
Function 10	72.50	94.91	**1.34e-4**	23.97	41.18	**4.94e-7**
Function 11	3.45	8.55	**1.06e-6**	1.02	1.43	**5.61e-8**
Function 12	2.03e-3	0.012	**2.24e-6**	2.95e-6	7.7e-4	**1.05e-5**
Function 13	12.57	15.57	**1.85e-9**	5.55	10.45	**6.66e-20**

Table 4. Comparison between standard GA and a GA-LLE for a population size of 100 individuals. For each benchmark function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	GA Population size = 100					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	GA-LLE	GA	**T-test P**	GA-LLE	GA	**T-test P**
Function 1	1.28	2.45	**8.73e-5**	0.48	0.56	**0.10**
Function 2	160.04	5017.2	**2.40e-23**	114.26	4015.8	**1.35e-25**
Function 3	315.0	1.53e4	**1.89e-19**	149.9	9551.8	**2.38e-19**
Function 4	10.76	276.8	**4.14e-26**	6.47	189.25	**1.52e-23**
Function 5	3.84e6	5.18e6	**5.52e-8**	2.21e6	3.80e6	**2.21e-18**
Function 6	589.6	7.21e5	**1.19e-17**	226.5	3.37e5	**1.25e-15**
Function 7	14.19	39.98	**0.021**	3.03e-4	1.19e-2	**2.68e-5**
Function 8	33.91	85.81	**7.72e-7**	6.42	10.59	**5.12e-6**
Function 9	-3733.1	-3802.1	**0.12**	-4189.1	-4153.2	**2.09e-9**
Function 10	66.72	86.56	**1.25e-5**	22.63	31.83	**1.24e-4**
Function 11	3.01	5.11	**1.07e-4**	0.91	1.23	**3.93e-8**
Function 12	4.8e-4	8.01e-3	**4.04e-6**	6.09e-7	4.99e-4	**2.13e-5**
Function 13	11.19	14.75	**2.3e-10**	5.53	9.97	**2.24e-7**

Table 5. Comparison between standard PNO and a PNO-LLE for a population size of 30 individuals. For each benchmark function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	PNO Population size = 30					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	PNO-LLE	PNO	**T-test P**	PNO-LLE	PNO	**T-test P**
Function 1	2.65	2.84	**0.82**	0.68	1.03	**1.3e-2**
Function 2	100.5	1290.8	**3.01e-16**	98.04	1070.1	**1.13e-16**
Function 3	109.5	1829.0	**4.47e-14**	98.12	1272.5	**5.90e-15**
Function 4	6.20	74.30	**2.80e-17**	5.90	58.91	**1.82e-14**
Function 5	1.51e6	1.69e6	**0.31**	1.12e6	1.34e6	**0.10**

Function 6	207.3	2.98e5	**2.32e-9**	159.6	1.58e5	**1.05e-9**
Function 7	0.18	4.44	**0.031**	3.01e-4	3.87e-4	**0.16**
Function 8	14.54	23.22	**0.09**	7.73	7.62	**0.83**
Function 9	-2240.8	-2186.6	**0.59**	-2557.2	-2374.2	**0.11**
Function 10	112.42	160.23	**2.58e-6**	83.96	126.66	**1.44e-7**
Function 11	1.50	2.03	**2.93e-3**	0.25	0.52	**5.31e-3**
Function 12	1.23e-5	3.51e-4	**4.02e-3**	3.01e-7	1.62e-5	**8.98e-5**
Function 13	5.25	11.43	**1.54e-14**	3.23	9.56	**2.32e-12**

Table 6. Comparison between standard PNO and a PNO-LLE for a population size of 50 individuals. For each benchmark function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	PNO Population size = 50					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	PNO-LLE	PNO	**T-test P**	PNO-LLE	PNO	**T-test P**
Function 1	1.46	2.73	**5.80e-2**	0.64	1.11	**0.002.18e-3**
Function 2	51.88	1065.1	**3.77e-17**	45.91	833.1	**4.63e-16**
Function 3	76.52	1156.5	**8.86e-13**	58.42	700.1	**1.70e-11**
Function 4	2.35	64.23	**2.16e-16**	1.79	46.25	**3.49e-14**
Function 5	7.57e5	1.25e6	**4.15e-6**	4.67e5	8.73e5	**2.61e-7**
Function 6	81.65	1.30e5	**2.71e-10**	53.46	5863.8	**1.87e-9**
Function 7	1.05e-3	0.64	**0.11**	3.02e-4	3.04e-4	**0.12**
Function 8	8.87	16.38	**1.47e-2**	6.21	7.33	**4.8e-2**
Function 9	-2459.1	-2396.4	**0.57**	-2602.1	-2485.0	**0.30**
Function 10	100.94	143.9	**7.17e-7**	81.66	109.18	**4.67e-4**
Function 11	1.20	1.49	**3.33e-5**	0.21	0.33	**3.87e-2**
Function 12	4.87e-6	1.23e-4	**4.01e-4**	5.57e-8	3.87e-6	**5.73e-4**
Function 13	4.27	9.00	**1.5e-12**	2.61	7.07	**3.19e-12**

Table 7. Comparison between standard PNO and a PNO-LLE for a population size of 100 individuals. For each benchmark function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	PNO Population size = 100					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	PNO-LLE	PNO	**T-test P**	PNO-LLE	PNO	**T-test P**
Function 1	0.95	2.15	**9.02e-2**	0.63	0.97	**7.82e-3**
Function 2	28.97	913.4	**7.94e-20**	18.03	630.0	**2.38e-18**
Function 3	52.40	878.7	**2.27e-13**	31.18	401.5	**1.55e-11**
Function 4	0.72	49.86	**8.27e-17**	0.15	30.68	**6.46e-15**
Function 5	4.47e5	1.07e6	**2.02e-11**	2.17e5	6.54e5	**7.62e-12**

Function 6	42.97	7028.1	**8.97e-11**	17.71	1759.5	**1.18e-9**
Function 7	4.05e-4	1.08e-2	**0.19**	3.00e-4	3.01-e4	**0.14**
Function 8	8.91	9.34	**0.24**	5.08	6.25	**5.293-2**
Function 9	-2646.2	-2386.8	**1.67e-2**	-2692.5	-2440.5	**2.3e-2**
Function 10	93.21	118.24	**0.011**	85.38	94.01	**0.33**
Function 11	1.07	1.22	**3.95e-4**	0.15	0.21	**0.14**
Function 12	1.72e-6	5.53e-5	**7.55e-3**	2.82e-8	7.10e-7	**1.3e-9**
Function 13	3.40	7.22	**6.44e-17**	1.61	4.77	**1.23e-11**

Real-World Problems

Two optimization problems taken from the mineral exploration industry serve as real-world tests. The first problem is to recover the distribution of subsurface rock density from measurements of the anomaly in the Earth's gravitational field. A sketch of the problem setting is given in Figure 6. A gravity anomaly profile is measured at 48 stations along a survey line. The density distribution is modeled by assuming the presence of three layers with a known average density, but each with an unknown deviation from the average. These density deviations, as well as the depths of the layers at various locations, are the variables to recover via optimization. The problem results in 60 parameters to recover (60-*D*). This is a high-dimensional, but fairly simple, optimization problem (Boschetti et al., 1997).

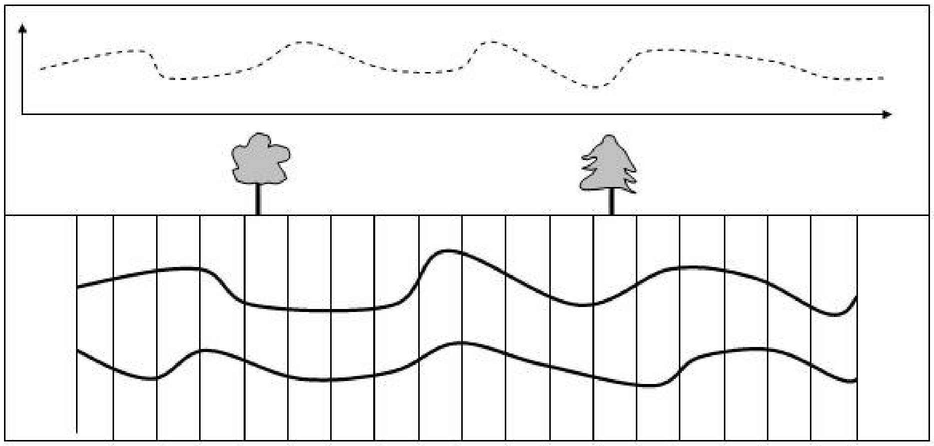


Figure 6. Sketch representation of the first real world optimization problem.
The purpose is to recover the distribution of subsurface rock density from measurements of the anomaly in the Earth's gravitational field. The dashed line represents a set of gravity anomaly measurements. The unknowns are the density values of the underground blocks and the local depths of the geological layers.

The second problem is to recover the seismic velocity of the subsurface (which also relates to rock density) from measurements of the travel time of seismic waves, from a number of sources, to several receivers located along a survey line on the Earth's surface. A sketch of the problem setting is given in Figure 7. The underground domain is modeled by assigning values of seismic velocity to a 2-*D* vertical grid of 5x9 nodes, and by interpolating bilinearly between grid nodes. The optimization must recover the seismic velocity values at the 45 grid nodes (45-*D*). This is a very difficult optimization problem because of the strongly non-linear effect of the seismic velocity at each node on the final seismic travel times (Boschetti et al., 1996).

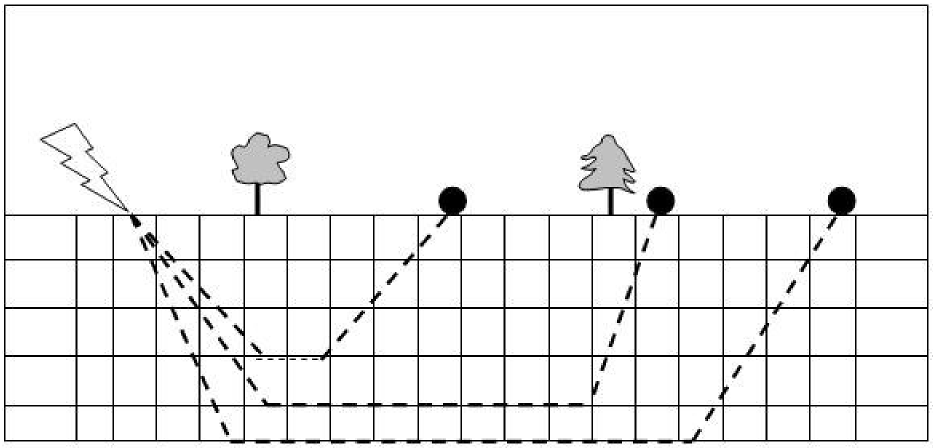


Figure 7. Sketch representation of the second real world optimization problem. The purpose is to recover the distribution of subsurface rock seismic velocities from measurements of the travel times of the seismic waves. The seismic waves are shot form a source and the travel times are measured at a number of receivers (black dots) along the earth surface.

The conclusions drawn from the test functions shown in the previous section seem to be confirmed by the two real-world tests (Tables 8 and 9). The LLE module considerably improves the performance of both the GA and PSO in all runs and all results are statistically significant.

Table 8. Comparison between standard PNO and a PNO-LLE for a population size of 30, 50 and 100 individuals on the real world problems. For each function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	PNO Population size = 30					
	Fun. Eval = 20*pop. Size			Fun. Eval = 100*pop. Size		
	PNO-LLE	PNO	**T-test P**	PNO-LLE	PNO	**T-test P**
Real World Function 1	9.77e-3	2.93	**3.33e-8**	9.05e-4	0.29	**1.27e-6**
Real World	8086.4	29686.8	**2.25e-10**	7002.5	16765.1	**3.96e-5**

Function 2						
	PNO Population size = 50					
	Fun. Eval = 20*pop. Size			Fun. Eval = 100*pop. Size		
	PNO-LLE	PNO	**T-test P**	PNO-LLE	PNO	**T-test P**
Real World Function 1	6.05e-3	1.68	**2.83e-6**	3.57e-4	0.21	**2.52e-8**
Real World Function 2	7768.4	28177.6	**6.62e-10**	6341.3	15245.1	**4.44e-9**
	PNO Population size = 100					
	Fun. Eval = 20*pop. Size			Fun. Eval = 100*pop. Size		
	PNO-LLE	PNO	**T-test P**	PNO-LLE	PNO	**T-test P**
Real World Function 1	2.31e-3	0.90	**8.30e-8**	2.10e-4	0.10	**1.83e-7**
Real World Function 2	6616.6	25145.2	**3.22e-10**	5521.5	13124.5	**5.91e-9**

Table 9. Comparison between standard GA and a GA-LLE for a population size of 30, 50 and 100 individuals on the real world problems. For each function, we include the average misfit calculated over 20 runs, as well as the UT-Test significance measure *p*. In all cases, results are presented after a number of evaluation equal to 20 times the population size and 100 time the population size.

	GA Population size = 30					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	GA-LLE	GA	**T-test P**	GA-LLE	GA	**T-test P**
Real World Function 1	0.26	37.00	**6.69e-11**	0.11	15.52	**1.16e-9**
Real World Function 2	9014.2	30777	**9.14e-9**	1520.3	8741.6	**2.51e-10**
	GA Population size = 50					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	GA-LLE	GA	**T-test P**	GA-LLE	GA	**T-test P**
Real World Function 1	0.19	30.66	**6.62e-10**	0.07	13.30	**1.37e-9**
Real World Function 2	5991.6	30199.3	**2.91e-11**	1490.1	8903.66	**1.32e-9**
	GA Population size = 100					
	Fun. Eval = 30*pop. Size			Fun. Eval = 100*pop. Size		
	GA-LLE	GA	**T-test P**	GA-LLE	GA	**T-test P**
Real World Function 1	0.19	39.13	**3.34e-10**	0.07	14.18	**1.07e-10**
Real World Function 2	4126.3	28945.1	**6.12e-10**	1317.3	8365.4	**5.15e-9**

Discussion

Despite its fundamentally simple structure, an EC-LLE algorithm uses the stochastically and progressively convergent sampling of the EC algorithm to produce an approximate mapping between data space and parameter space. This is a conceptually significant difference compared to traditional optimization approaches, which merely map the parameter space into the data space. This is not the first time such an idea is presented in the literature (Oldenburg and Ellis, 1991), but the simplicity of the algorithm, and its closed-form solution, are particularly appealing.

Faced with the choice of using the LLE module, a practitioner may ask "Is it worthwhile for me to implement and use the LLE module on my specific real-world problem?" Based on this research, the answer is "yes", or, at least, "there is little reason not to". Computationally, the only significant cost in using the LLE module is in the extra forward modeling involved in evaluating the individuals it produces. The actual computation in the LLE module involves only algebraic manipulation of small matrices, and is comparable to the computation required in the internal running of a standard EC algorithm. The worst-case scenario is that the LLE module does not find any good individual. In that case, the EC would proceed as if the LLE had not been used (i.e., no individual in the population is replaced), and the only extra cost is the function evaluation of that individual. Since the number of K values is usually kept much smaller than the population size, the additional cost is a small percentage of the standard EC cost. All tests suggest that, in any event, such a scenario is quite unlikely.

Another concern may arise from interfering with the normal EC process by accelerating the convergence too much, at the expense of adequately exploring the solution space. Premature convergence in a local minimum may be the consequence. This also seems unlikely while, as in the above implementation, a single individual produced by the LLE module replaces a single (worst) individual in the population. Basically, the modification in the EC population is comparable to standard 'elitism', which is a module now included in most EC algorithms.

Most heuristic algorithms depend on a number of parameters that need to be experimentally tuned. The LLE module, in its current implementation, depends on two parameters: the range of K (its values and the number of Ks used), and the maximum number of LLE individuals accepted at each generation. Currently, only one individual is inserted into the EC population, if its misfit is better than that of the worst individual in the population. This is a safe choice, since it carries minimal risk of causing premature convergence. Inserting a larger number of individuals can possibly speed up convergence.

The choice of the optimal K is probably worth more attention, since it can reduce the extra computational effort implied in the use of the LLE module (although, as described above, this extra cost is quite small). Work within the LLE research community (Kouropteva et al., 2002) has not provided any definitive algorithm. One option might be to predict the calculated data (not the scalar misfit) by using the LLE weights, and to use this prediction to guide the selection of K. Departures between predicted and actual vector values may also indicate areas where local linearity is broken, and which may consequently be worth exploring via the EC. This avenue of research will form part of future investigations.

The approach has been tested on problems with continuous parameterizations, for which LLE is best suited. We expect that extension to discrete optimization problems, with relatively fine discretisation should work equally well. The use of the LLE module to problems which are inherently discrete in nature, like coarsely discrete problems or combinatorial optimizations, may not be straightforward. More research for these applications is needed.

As a final note, it is reasonable to ask whether some non-linear alternatives can better approximate the mapping than a locally linear method. The machine learning and image processing community has recently been very active in the development of several dimensionality reduction algorithms (e.g., Balasubramanian et al., 2002). The EC community should follow this progress, because several of those tools can greatly benefit optimization approaches as well.

Conclusions

A module based on the Local Linear Embedding (LLE) algorithm can be used to improve the performance of two Evolutionary Computation (EC) algorithms. At each EC generation, the module performs a locally linear mapping between the data space (measured data) and the parameter space, thereby mapping a measured data vector into the parameter space, and obtaining an approximate reconstruction of the unknown, optimal solution. On a number of synthetic and real-world problems, optimizations including the module consistently outperform a standard genetic algorithm and a standard particle swarm optimization algorithm.

References

M. Balasubramanian, E. L. Schwartz, J. B. Tenenbaum, V. de Silva and J. C. Langford. The Isomap Algorithm and Topological Stability. Science, vol. 295(5552), 7a, 2002.

M. Belkin, P. Niyogi, Laplacian Eigenmaps for Dimensionality Reduction and Data Representation, Neural Computation, June 2003; 15 (6):1373-1396.

F.Boschetti, M. Dentith, R. List, Inversion of seismic refraction data using Genetic Algorithms, 1996, Geophysics, 1715-1727.

F.Boschetti, M. Dentith, R. List, Inversion of gravity and magnetic data by Genetic Algorithm, 1997, Geophysical Prospecting, 461-478

F. Boschetti and L. Moresi, 2001, "Interactive Inversion in Geosciences", Geophysics, 64, 1226-1235.

M. Brand, 2002, Charting a manifold. Neural Information Processing Systems 15 (NIPS'2002)

T. Cox and M. Cox. Multidimensional Scaling. Chapman & Hall, London, 1994.

Davis, L., 1991, Handbook on genetic algorithms: Van Nostrand Reinhold.

D. L. Donoho and C. E. Grimes. Hessian eigenmaps: locally linear embedding techniques for highdimensional data. Proceedings of the National Academy of Arts and Sciences, 100:5591–5596, 2003.

I. T. Jolliffe. Principal Component Analysis. Springer-Verlag, New York, 1986.

Goldberg,D. E., 1989, Genetic algorithms in search, optimization, and machine learning: Addison-Wesley Publ. Co., Inc.

Kouropteva O, Okun O, Hadid A, Soriano M, Marcos S & Pietikäinen M (2002) Beyond Locally Linear Embedding Algorithm. Technical Report MVG-01-2002, University of Oulu, Machine Vision Group, Information Processing Laboratory.

Moré, J.J., Garbow, B.S. and Hillstrom, K.E., Testing Unconstrained Optimization Software, ACM Trans. Math. Software 7 (1981), 17-41.

Mouser C., and Dunn S., 2004, Comparing Genetic Algorithms and Particle Swarm Optimisation for an Inverse Problem Exercise, The 12th Biennial Computational Techniques and Applications Conference, Melbourne, Australia (submitted).

Oldenburg D.W., and Ellis R.G., 1991, Inversion of Geophysical Data Using an Approximate Inverse Mapping, Geophysical Journal International, 105, 325-353

S. T. Roweis and L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science, 290, 2323--2326, 2000.

L. Saul and S. Roweis. Think Globally, Fit Locally: Unsupervised Learning of Nonlinear Manifolds. Technical Report MS CIS-02-18, University of Pennsylvania, 2002.

J.B. Tenenbaum, V. de Silva and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. Science, 290, 2319--2323, 2000.

Yao X., Evolutionary Computation: Theory and Applications. World Scientific, 1999

Wijns, C., F. Boschetti and L. Moresi, "Inversion in Geology by Interactive Evolutionary Computation", 2003, Journal of Structural Geology, 25(10), 1615-1621, doi:10.1016/S0191-8141(03)00010-5.