GLOBUS: A set of MCL routines for Global Optimization

Authors

September 20, 2001

Department of Computer Science, University of Ioannina

Ioannina 45110 - GREECE

Abstract

We present implementations of various global optimization algorithms in the Merlin Control Language (MCL). Developing global optimization routines in MCL has several advantages especially for methods that invoke local searches, since the programmer has at his disposal all the facilities of the Merlin optimization environment. We have implemented the random multistart, a controlled random search, the Metropolis-Boltzmann simulated annealing, a simulated annealing with local optimization, the single linkage clustering, and a genetic algorithm with local tuning.

1 Introduction

Optimization has proved to be an invaluable tool in many scientific fields. Its strength comes in part from the fact that a plethora of diverse scientific as well as practical problems, can be reduced to optimization problems, and in part from the existence of robust and effective optimization methods. At this point we would like to make the distinction between local optimization and global optimization. A function may have more than one points where the optimality conditions are satisfied, ie many local minima. Any of these is an acceptable solution as far as local optimization is concerned. Among the local minima, the one with the lowest value, ie the global minimum, is the solution that global optimization seeks for. Various global optimization methods have been developed rather recently, most of them being of stochastic nature. These methods recover the global minimum with a probability tending to one only asymptotically. Deterministic methods that supposedely guarantee the recovery of the global minimum, face various computational difficulties.

We base our implementation on the Merlin 3.0 [1] optimization environment and its programming language MCL [2]. Merlin offers several local optimization algorithms and many useful facilities. MCL is a programming language for implementing optimization strategies in the context of the Merlin environment. Hence we coded the global algorithms as optimization strategies in MCL. This choice is the key to the great simplification of the programming burden, especially for global methods that use as an intermediate step local searches as one can easily verify by inspecting the MCL code.

This article is organized as follows. In section 2, we present the implemented algorithms and we make some comments concerning their strong and weak points. In section 3 we present the results of the application of these algorithms to test problems and finally in section 4, we give the MCL code listings and examples of usage. We also refer to applications where some of these methods proved to be very useful.

2 Algorithm description

2.1 Controlled random search

This is a modification of Price's [3] algorithm, described in [4]. The algorithm seeks a global minimum in a given domain D. The steps of the algorithm are given by:

Input Data:

- M, an integer such that $M > N + 1$, where N is the space dimension. (Suggested value: $M = 25N$)
- ϵ, a small positive constant. (Suggested value: $\epsilon = 10^{-6}$)
- ω, a rather large positive constant. (Suggested value: $\omega = 1000$)

Step 0:

- Set $k = 0$. Form the initial set $S^k = \{x_1^k, x_2^k, \ldots, x_M^k\}$ by picking M points randomly from D.
- Evaluate: $f_i^k = f(x_i^k)$ for $i = 1, 2, \ldots, M$

Step 1:

- $f_{max}^k = \max\{f_i^k\}$, and let the corresponding point be denoted as x_{max}^k. Similarly,
- $f_{min}^k = \min\{f_i^k\}$, and the corresponding point is denoted as x_{min}^k.
- IF $f_{max}^k - f_{min}^k \leq \epsilon$, STOP

Step 2:

- Choose at random $N + 1$ points $\{x_{i_0}^k, x_{i_1}^k, \ldots, x_{i_N}^k\}$ from S^k.

- Calculate the weighted centroids:

$$c_w^k = \sum_{j=1}^{N} w_j^k x_{i_j}^k, \quad f_w^k = \sum_{j=1}^{N} w_j^k f(x_{i_j}^k)$$

where:

$$w_j^k = \frac{n_j^k}{\sum_{j=1}^{N} n_j^k}, \quad n_j^k = \frac{1}{f(x_{i_j}^k) - f_{min}^k + \phi^k}, \quad \phi^k = \omega \frac{(f_{max}^k - f_{min}^k)^2}{f_{max}^0 - f_{min}^0}$$

- Calculate a trial point $\overline{x}^k$as:

$$\overline{x}^k = (x_{i_0}^k - c_w^k) \frac{f(x_{i_0}^k) - f_w^k}{f_{max}^k - f_{min}^k + \phi^k} + \Delta_w^k$$

- where $\Delta_w^k = 2c_w^k - x_{i_0}^k$ if $f_w^k \leq f(x_{i_0}^k)$ and $\Delta_w^k = 2x_{i_0}^k - c_w^k$ if $f_w^k > f(x_{i_0}^k)$
- IF $\overline{x}^k \notin D$ REPEAT Step 2
- Compute $f(\overline{x}^k)$

Step 3:

- IF $f(\overline{x}^k) \geq f_{max}^k$ THEN
  - Calculate the success rate (the fraction of function evaluations that led to a new lower upper bound)
  - IF success rate >50%, Set $S^{k+1} = S^k$, $k = k + 1$ and GOTO Step 2
  - Calculate $y^k = \frac{c_w^k + x_{i_N}^k}{2}$, compute $f_y = f(y^k)$
  - IF $f_y \geq f_{max}^k$, Set $S^{k+1} = S^k$, $k = k + 1$ and GOTO Step 2
  - Set $S^{k+1} = S^k \bigcup \{y^k\} - \{x_{max}^k\}$, $k = k + 1$ and GOTO Step 1
- ENDIF

Step 4:

- Set $S^{k+1} = S^k \bigcup \{\overline{x}^k\} - \{x_{max}^k\}$
- Increment: $k = k + 1$ and GOTO Step 1

Comments The above algorithm (as quoted in [4]) has been designed for problems where the objective function is expensive to calculate, is affected by the presence of noise and its gradient is not analytically available. Such problems, in the case of local optimization are treated with reasonable success by the irregular Simplex method [5]. The presented global algorithm is inspired in part by the tactic followed in that method. (Maintaining a population of points and performing operations such as reflection with respect to a centroid, etc.)

2.2 Simulated Annealing (SA)

This algorithm was originally [6] devised for discrete problems. However its extension to the case of continuous functions was rather straightforward. We refer to [7], [8] and [9] for an account of the method and for interesting applications and to [10] for an integrated review and complete list of references.

SA is associated with a non-negative "temperature" parameter. This "temperature" is used to regulate the probability that allows steps to be taken to parts of the space, where the objective function increases. The probability of accepting a step that causes a change $\Delta f = f_{new} - f_{old}$ is given by :

$$P(T, \Delta f) = \begin{cases} e^{-\Delta f / T} & \Delta f \geq 0 \\ 1 & \Delta f < 0 \end{cases}$$

By inspection, downhill steps ($\Delta f < 0$) are always accepted, while uphill steps may be accepted if $T > 0$. (At $T = 0$, only downhill steps are taken). So one starts with high temperatures and hence quite often visits areas of the space where the objective function increases. Then an annealing schedule is applied. The subtlety here is that in order to guarantee that the probability of visiting the area of the global minimum is non-vanishing, the annealing must follow a certain schedule dictated by the acceptance probability. For the Boltzmann distribution above, the cooling cannot be faster than $T(k) = \frac{T_0}{\ln(k)}$[7]. There are many variants using various distributions and cooling schedules. Our implementation follows the article by Goffe et al [11] and their publically available Fortran code. Ingber [10], assigns the name "Simulated Quenching" to this algorithm, since for the Boltzmann distribution, the exponential annealing schedule it uses, is too fast for retaining a true ergodic search. The algorithmic description follows.

Input Data:

- **Initial** temperature T, point $x = \{x_i, i = 1, 2, .., N\}$ and steps $h = \{h_i, i = 1, 2, .., N\}$
- **Bounds** for each variable a_i, b_i such that $x_i \in [a_i, b_i]$, $i = 1, 2, .., N$
- **Annealing** Factor α
- **Loop** Iterations: N_A, N_T , N_S

Step 0:

- **Calculate** $f_c = f(x)$ and set : $x_{best} = x$ and $f_{best} = f_c$

Loop N_A times (Annealing Schedule)

- **Loop** N_T times (Step Adjustment Schedule)
  - **Loop** N_S times (Basic Simulated Annealing move)
    - **Loop** over free variables x_i
      - **Set:** $x^t = x$ (the whole vector)

$x_i^t = x_i + (2\xi - 1)h_i,\ \xi$ random in $(0,1)$
if it is outside of feasible range $[a_i, b_i]$ reset:
$x_i^t = a_i + (b_i - a_i)\xi,\ \xi$ random in $(0,1)$
$f_t = f(x^t)$
endif
if $f_t \leq f_{best}$ then
$x_{best} = x^t,\ f_{best} = f_t$
endif
if $f_t < f_c$ then
set $x = x^t$ and $f_c = f_t$
else
calculate $p = e^{-\frac{f_t - f_c}{T}}$
if $p > \xi$ (where ξ random in $(0,1)$) set $x = x^t$ and $f_c = f_t$
endif
EndLoop
EndLoop
Adjust the step size (as described below)
EndLoop
Anneal: Set $T = \alpha T$
set $x = x_{best},\ f_c = f_{best}$

EndLoop

The step adjustment is performed via the following algorithm that manages to attain an acceptance move rate of approximately 50%.

Loop over the free variables x_i

Calculate $r_i = \frac{M_i}{N_S}$, where M_i are the number of accepted moves in the direction x_i in the last N_S trials.
if $r_i > \frac{3}{5}$ set: $h_i = h_i(1 + 2\frac{r_i - 3/5}{2/5})$
if $r_i < \frac{2}{5}$ set: $h_i = \frac{h_i}{1+2\frac{2/5 - r_i}{2/5}}$
if $h_i > b_i - a_i$ reset: $h_i = b_i - a_i$

EndLoop

2.2.1 Simulated Annealing with Local Tuning (SALT)

2.3 Single Linkage Clustering

Here we briefly describe a stochastic algorithm due to Boender et al [12]. A few comments are appropriate. The algorithm attempts to find all local minima inside a bounded set $S \subset R^n$, that are potentially global. These local minima are obtained by a local-search procedure starting from suitably chosen points in a properly maintained sample. Algorithms of stochastic nature have the problem of recovering the same local minimum repeatedly, a fact that diminishes their efficiency. The present algorithm is devised in such a way so as to avoid this undesirable repetition. Thus, for a given local search technique, the region of attraction of a local minimum x^* is defined to be the set of all points in S, starting from which the local search will end up to x^*. The algorithm tries to identify these regions of attraction as economically as possible, idealy by invoking only one local search per region, and employs a so called clustering procedure to be described.

Input Data:

> **A** number M which is the size of the initial sample.
>
> A number $\gamma \in [0, 1]$ representing the fraction of the points in the sample to be retained.

Let $X^*$and $X^{(1)}$denote the set of the local minima found so far and the sample points from which the local search did not return a new local minimum. Initially both of these sets are empty.

STEP 1: Draw M points and add them to the sample

STEP 2: Construct a transformed sample by taking the points of the current sample with the γ percent lowest function values and by performing a steepest descent step to each of these points.

STEP 3: Apply a clustering procedure to the transformed sample. The elements of $X^*$are first chosen a seed points, followed by the elements of $X^{(1)}$. If all points of the tranformed sample can be assigned to a cluster, go to STEP 5.

STEP 4: Among the points not yet clustered, let $x^{(1)}$ be the point with the lowest function value. Apply the local search to $x^{(1)}$ to find a local minimum x^*. If $x^* \notin X^*$, add $x^*$to $X^*$and choose it as the next seed point. If $x^* \in X^*$ add $x^{(1)}$ to $X^{(1)}$ and choose $x^{(1)}$ as the next seed point. Repeat STEP 4 until all points have been assigned to a cluster. If a new point has been added to X^*, go to STEP 1.

STEP 5: Determine y^*, the smallest local minimum value found, and stop.

2.3.1 The Clustering procedure

Let z be the seed point of a cluster (that initially contains no points). The first point added to the cluster is the one with the shortest Euclidean distance from the seed. Let this distance be denoted by $d(z,x) \equiv \sqrt{\sum_i (z_i - x_i)^2}$. Subsequent points are added to the cluster using the following procedure. Let C be the set of points that belong to the cluster. Let x be a point in the sample. Then its distance from the cluster is defined to be: $D(x,C) = min_y(d(x,y))$ where $y \in C$. Namely, the point in the cluster that is closest to point x, defines the distance between the point x and the cluster. The point x in the sample for which $D(x,C)$ is minimal is a candidate member of the cluster. There are two tests that have to be satisfied in order that this point is added to the cluster.

1. The minimal distance $min_x(D(x,C)) \leq d_c$, a preset threshold distance.
2. The gradient test.
   - If the seed point $z \in X^*$ then we must have: $(z-x)^T \nabla f(x) < 0$
   - If the seed point $z \in X^{(1)}$ then we must have: $(z-x)^T(\nabla f(z) - \nabla f(x)) > 0$

The gradient test originates from the observation that around a local minimum x^*, where a cluster develops, the objective function can be approximated by:

$$f(x) = f(x^*) + \frac{1}{2}(x-x^*)^T H^*(x-x^*)$$

where H^* the Hessian matrix at the minimum $H^* = H(x^*)$, a positive definite matrix.

Taking the gradient of the above we obtain: $\nabla f(x) = H^*(x-x^*)$. Similarly for any other point y in the neigborhood of x^*, $\nabla f(y) = H^*(y-x^*)$ and hence by subtracting and multiplying from the left by $(x-y)^T$ we obtain:

$$(x-y)^T[\nabla f(x) - \nabla f(y)] = (x-y)^T H^*(x-y) > 0$$

since H^* is positive definite. Setting $y = z$ above, the meaning of the gradient test becomes obvious.

References

[1] D. G. Papageorgiou, I. N. Demitropoulos, I. E. Lagaris, "Merlin- 3.0 A multidimensional optimization environment" Comput. Phys. Commun. 109(1998)227-249

[2] D. G. Papageorgiou, I. N. Demitropoulos, I. E. Lagaris, "The Merlin Control Language for strategic optimization", Comput. Phys. Commun. 109(1998)250-275

[3] W. L. Price, "A controlled random search procedure for global optimization", in Towards Global Optimization 2, L.C.W. Dixon and G.P.Szego (eds.), North-Holland, Amsterdam.

[4] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo and S. Lucidi, "A new version of the Price's algorithm for global optimization", Journal of Global Optimization 10(1997)165-184.

[5] J. A. Nelder and R. Mead, "A Simplex method for function minimization", Comput. J. 7(1965)308-313

[6] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing", Science 220(1983)671-680

[7] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distribution and Bayesian restoration of images", IEEE Trans. Patt. Anal. Mac. Int. 6(1984)721-741

[8] P. J. M. van Laarhoven and E. H. L. Aarts, "Simulated Annealing: Theory and Applications", 1987, D. Riedel, Boston.

[9] E. Aarts and J. Korst, "Simulated Annealing and Boltzmann Machines", 1989, John Wiley and Sons.

[10] L. Ingber, "Simulated Annealing: Practice versus Theory", J. Mathl. Comput. Modelling 18(1993)29-57

[11] W. L. Goffe, G. D. Ferrier and J. Rogers, "Global Optimization of Statistical Functions with Simulated Annealing", J. Econometrics 60(1994)65-100.

[12] C. G. E. Boender, A. H. G. Rinnooy Kan and G. T. Timmer, 'A Stochastic Method for Global Optimization', Mathematical Programming 22(1982)125-140