Noname manuscript No.
(will be inserted by the editor)

Machine Learning for Global Optimization

A. Cassioli[*] · D. Di Lorenzo[*] · M. Locatelli[**] · F. Schoen[*] · M. Sciandrone[*]

Abstract In this paper we introduce the LeGO (LEarning for Global Optimization) approach for global optimization in which machine learning is used to predict the outcome of a computationally expensive global optimization run, based upon a suitable training performed by standard runs of the same global optimization method. We propose to use a Support Vector Machine (although different machine learning tools might be employed) to learn the relationship between the starting point of an algorithm and the final outcome (which is usually related to the function value at the point returned by the procedure). Numerical experiments performed both on classical test functions and on difficult space trajectory planning problems show that the proposed approach can be very effective in identifying good starting points for global optimization.

Keywords

global optimization, machine learning, support vector machines, space trajectory design

Introduction

Many instances of global optimization algorithms require the execution of a procedure starting from randomly chosen points in a domain or they require choosing suitable initial values for a finite number of parameters. When dealing with multi–modal problems, i.e., optimization problems with many local optima which are not global, it is a common procedure to run several instances of the same algorithm either starting from different points or using a different set of parameters. Usually the large amount of data generated during these runs is lost, as the user is typically interested in the best run only, i.e., the one which has produced the best overall result. To the authors' knowledge, at least in the field of global optimization, there has been no formalized attempt of learning from the whole computational procedure. In this paper we present

[*] Dip. Sistemi e Informatica, Univ. di Firenze (Italy)

[**] Dip. di Informatica, Univ. di Torino (Italy)

Address(es) of author(s) should be given

a novel approach, which we call LeGO (LEarning for Global Optimization), in which standard machine learning tools are employed in order to learn the unknown relationship between the starting condition (initial point or parameters) and the final value obtained. The idea underlying LeGO is indeed very simple: first run some instances of an algorithm which requires an initialization; then from the results of these runs, train a machine learning tool to be capable of estimating the outcome of future runs. Some precursors of this idea might be found in the literature. As a quite well known example, in (Rinnooy Kan & Timmer, 1987a, 1987b) clustering methods were employed in order to select promising starting points for Multistart. In those approaches some form of statistical learning was employed in order to be able to select randomly generated points as candidate starting points for local searches. In a more deterministic setting in (Dill, Phillips, & Rosen, 1997a, 1997b; Mangasarian, Rosen, & Thompson, 2005) methods based on using a first set of function evaluations to build convex underestimators are introduced. Using this information, new, promising, candidate points for function evaluation are selected. Many other methods make use of past information to build improved approximation or surrogate models in order to select new candidates. In (Jin, Olhofer, & Sendhoff, 2002; Jin, 2005) some form of learning is embedded in evolutionary methods. However we are not aware of previous papers dealing with the application of machine learning to the problem of deciding a priori whether a starting point is promising or not and which are general enough to be applicable both to simple methods like Multistart as well as to more complex and refined global optimization algorithms. A possible exception might be (Ampatzis & Izzo, 2009); however we remark that while in many cases in the literature machine learning is used to build a suitable approximation of the objective function which is used to guide the choice of next iteration, here the fundamental difference is that learning is used to make inference on the relationship between starting points and quality of the final result of an optimization run, i.e. learning is directly applied to the feasible domain.

What we present in this paper is not a new algorithm, but a framework which can be adopted in many computational schemes; of course this will not replace standard global optimization methods. The LeGO approach can be seen as a refinement procedure, to be carried out in parallel with (and not as a replacement for) global optimization. In the problems we tested, this learning procedure was successful in improving the solutions found in previous runs and in generating bunches of very good solutions. However it is clear that, in particular when dealing with high dimensional spaces, finite sampling in the feasible region will never be dense enough to guarantee a perfect learning. So, while retaining the generalization capabilities that machine learning is usually able to deliver, it is clear that LeGO will not, in general, generate completely unpredictable and radically different solutions. It will, however, significantly accelerate the search for good solutions, quickly discarding unpromising ones. So it can be seen as an acceleration technique.

The paper is structured as follows: in Section 1 the idea of the LeGO approach will be presented and its possible application to global optimization introduced; we remark that, although the approach is general enough to be useful for general nonlinear optimization problems, in this paper, in order to focus on the new approach, we choose to restrict experimentation to box-constrained problems. in Section 2 a brief introduction to Support Vector Machines as learning tools will be given. In Section 3 we will discuss numerical results obtained with standard test functions for global optimization: the aim of these experiments is to show the potential of this new approach on relatively easy test problems; in Section 4 we tested in a systematic way our approach on a standard set of test functions for global optimization and propose some guidelines to choose relevant parameters; in Section 5 the ideas developed in this paper are applied to a challenging optimization problem made available to the scientific community by the Advanced Concept Team at ESA, the European Space Agency. This problem consists in finding optimal trajectories for long range inter-planetary space missions. The proposed approach displayed excellent behaviour for this hard test problem.

1 General framework

Given an optimization problem

$$\min_{x \in S \subseteq \mathbb{R}^n} f(x)$$

let:

- $G()$ be a procedure to generate starting points (usually embedding a random component);
- $R(x)$ be a "refinement" procedure, which receives an input point x and returns a point in $\mathbb{R}^n$ (hopefully in S), like, e.g., a standard local search procedure;

We assume that the computational cost of G is significantly lower than the computational cost of R. Many heuristic techniques for solving optimization problems are based on multiple runs of procedure R starting from points generated by procedure G. Formally, the scheme is the following:

```
$f^\star = +\infty$ ;          /* Initial estimate of the global minimum value * */
$k = 0$;
repeat
    $x_k = G()$ ;                              /* generate a starting point */
    ; $y_k = R(x_k)$ ;                         /* generate a refined solution */
    ; if $f(y_k) < f^\star$ then
        $f^\star = f(y_k)$ , $x^\star = y_k$ ;   /* current global optimum estimate */
    end
    $k = k + 1$;
until stop condition ;
return $(x^\star, f^\star)$;
```

Algorithm 1: $\mathcal{P}(G, R)$ procedure.

The different runs of procedure R are usually independent from each other. Then, we might wonder whether it is possible to improve the results of some runs by exploiting the results of previous runs using a *machine learning*-based approach.

From a general point of view, machine learning deals with the development of algorithms and techniques that learn from observed data by constructing mathematical models that can be used for making predictions and decisions.

Here we focus on supervised learning of classification functions, that is on the problem of learning an unknown function $g : S \to \{-1, 1\}$ given a set of training examples $\{x_i, d_i\} \in S \times \{-1, 1\}$, where the label d_i denotes the class of the input vector x_i.

With reference to our context, we assume that there exists an unknown relationship between the starting point x_k generated by G and the final point y_k determined by R. We associate the label $d_k = +1$ to the input vector x_k if the final function value

$f(y_k)$ is "sufficiently low" and the label $d_k = -1$ otherwise. In this paper, we used a threshold T to identify sufficiently low function values; this threshold might be chosen either through prior knowledge on the value of the global minimum or, as it will be shown later, through a cross-validation procedure.

A starting point x_k with associated label $d_k = +1$ represents a "good" starting point.

We devote a fixed number of runs to generate pairs composed of initial points and final function values $(x_k, f(y_k))$ through the execution of G and R, and hence to construct the corresponding training set (x_k, d_k), where the label $d_k \in \{-1, 1\}$ indicates if, given the starting point x_k, the refinement procedure leads to a final point y_k whose function value is, respectively, higher or lower than the threshold T. The training set obtained this way can then be used for training a classifier CLS which, given in input a point x, returns "yes" $(+1)$ if point x is accepted and "no" (-1) otherwise.

Formally, we might employ the following scheme:

```
$f^\star = +\infty$, $k = 0$;
$TS = \emptyset$ ;                                   /* training set initialization */
let $T \in \mathbb{R}$ ;                                          /* choose a threshold */
repeat
    $x_k = G()$ ;
    $y_k = R(x_k)$ ;
    if $f(y_k) \leq T$ then
        $TS = TS \cup \{(x_k, +1)\}$
    else
        $TS = TS \cup \{(x_k, -1)\}$
    end
    if $f(y_k) < f^\star$ then
        $f^\star = f(y_k)$   ,   $x^\star = y_k$;
    end
    $k = k + 1$;
until stop condition for training ;
$CLS = train(TS)$ ;                         /* train a classifier on data $TS$ */
; repeat
    $x_k = G()$ ;
    if $CLS(x_k) = +1$ then   /* is the $x_k$ accepted by the classifier? */
        $y_k = R(x_k)$ ;      /* execute the algorithm from accepted starting
        points */
        if $f(y_k) < f^\star$ then
            $f^\star = f(y_k)$   ,   $x^\star = y_k$;
        end
        $k = k + 1$;
    end
until stop condition ;
return $(x^\star, f^\star)$
```

Algorithm 2: $\mathcal{P}'(G, R, CLS)$ scheme

2 A brief introduction to classification via SVM

In this section, in order to have a self–contained description of our approach, we briefly present the Support Vector Machine (SVM) classifiers used in our experiments. For a general introduction to SVM theory we cite, for example, (Vapnik, 1998; Burges, 1998; Schölkopf & Smola, 2002).

We refer to the standard classification problem to construct (train) a classifier to distinguish between two disjoint point sets in a Euclidean space.

Consider the training set

$$TS = \{(x_i, d_i),\ x_i \in \mathbb{R}^n,\ d_i \in \{-1, 1\},\ i = 1, \dots, N\}$$

and assume it is linearly separable, that is, there exists a separating hyperplane

$$H(w, b) = \{x \in \mathbb{R}^n : w^T x + b = 0\}$$

such that

$$\begin{aligned} &w^T x_i + b \geq 1 \quad \forall x_i :\ d_i = 1 \\ &w^T x_i + b \leq -1\ \forall x_i :\ d_i = -1 \end{aligned} \tag{1}$$

The *margin* $\rho(w, b)$ of a separating hyperplane $H(w, b)$ is the distance from the hyperplane to the closest training points, i.e.,

$$\rho(w, b) = \min_{x_i, i=1,\dots,N} \frac{|w^T x_i + b|}{\|w\|}.$$

Linear SVM approach picks out, among linear classifiers, the optimum separating hyperplane (i.e., the hyperplane having maximum margin). The basic training principle of SVM, motivated by statistical learning theory (Vapnik, 1998), is that the expected classification error for unseen test samples is minimized, so that SVM defines a good predictive model. The optimum hyperplane can be determined by solving the following quadratic programming problem

$$\begin{aligned} &\min_{w \in \mathbb{R}^n, b \in \mathbb{R}} \frac{1}{2}\|w\|^2 \\ &d_i(w^T x_i + b) \geq 1 \quad i = 1, \dots, N. \end{aligned} \tag{2}$$

In practice linear classifiers may perform poorly when the data are not linearly separable, and we need classifiers that produce nonlinear discriminants. The idea underlying nonlinear SVM is to map the input vectors into a high-dimensional space, called *feature space*, where the optimal separating hyperplane is constructed. Formally, denoting by $\phi : R^n \to H$, a nonlinear map from the input space to the feature space, the problem is

$$\begin{aligned} &\min_{w \in H, b \in \mathbb{R}, \xi \in \mathbb{R}^N} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi^i \\ &d_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \quad i = 1, \dots, N \\ &\xi_i \geq 0 \quad i = 1, \dots, N, \end{aligned} \tag{3}$$

where ξ_i for $i = 1, \dots, N$ are the slack variables, and the term $\sum_{i=1}^{N} \xi_i$ is an upper bound on the training error. The regularization parameter $C > 0$ trades off margin size and training error.

Using Wolfe's dual theory it is possible to construct the nonlinear SVM classifier without having to consider the mapping ϕ in explicit form but only requiring the knowledge of the inner product in the feature space (the inner product *kernel*). Common kernels are

- Polynomial $k(x, z) = \left(x^T z + \gamma\right)^p$, where $p \geq 1$ and $\gamma \geq 0$,
- Gaussian $k(x, z) = \exp\left(-\gamma \|x - z\|^2\right)$, with $\gamma > 0$.

The regularization parameter $C > 0$ and kernel parameters are usually determined by a standard cross-validation tuning procedure, which is a widely employed technique to prevent overfitting.

According to a standard $k-$fold cross-validation strategy, the training data is divided into k subsets of equal size. Sequentially, one subset is used as validation set and the remaining $k - 1$ subsets are employed to train the classifier. In this way, each training vector is predicted once and the cross-validation accuracy is given by the percentage of training vectors which are correctly classified. The cross-validation accuracy so defined is computed in correspondence to a finite set of SVM parameter vectors. The vector yielding the best cross-validation accuracy is selected and is used to train the classifier. Note that the cross-validation prediction accuracy can reflect the performance on classifying unknown data. In our computational experiments described later we used `Libsvm` (Chang & Lin, 2001), a simple, easy to use, and efficient software for SVM classication and regression. In particular we proceed to the SVM training using Gaussian kernel; in some tests we performed a cross–validation procedure based on a grid search to estimate the best values of the parameters, i.e., C and γ. According to this procedure, a dyadic coarse grid search in the parameters space $\left(i.e., C = \left[2^0, 2^1, \dots, 2^{10}\right] \text{ and } \gamma = \left[2^0, 2^1, \dots, 2^4\right]\right)$ is performed and the pair (C, γ) yielding the best cross-validation accuracy is selected.

Finally we remark that we employed nonlinear SVM classifiers based on Gaussian kernel in order to tackle problems which may require separations with complex boundaries. However, in some cases it would be preferable (in terms of prediction accuracy) to adopt a simpler classifier, e.g. a linear SVM, or other kernel functions. Choosing what kind of classifier to use deserves in practice particular attention, and can be performed using again a cross-validation approach. This practical issue has not been investigated in the numerical experiments, whose aim has been that of pointing out the validity of the methodology.

3 Numerical Experiments

Although the proposed approach can be applied to different optimization problems and algorithms, we focused our attention on Global Optimization (GO) problems. Within this field, we considered two approaches, namely the standard Multistart (MS) and the Monotonic Basin Hopping (MBH) (see, e.g., (Leary, 2000; Grosso, Locatelli, & Schoen, 2007)), which is a local exploration procedure similar to Iterated Local Search (see, e.g., (Lourenço, Martin, & Stülze, 2003)). The basic structure of these algorithms (also related with the notation of Algorithm 1) is the following:

Multistart (MS) uniformly samples the feasible set (using a suitable operator G) and then starts a local search from each sampled point (which represents the refinement procedure R in this case);

Monotonic Basin–Hopping (MBH) randomly generates an initial local minimizer (operator G); performs at each iteration a perturbation of the current local minimizer $\bar{x}$; starts a standard local search: if the resulting local minimizer improves the objective function in $\bar{x}$, it substitutes $\bar{x}$, otherwise $\bar{x}$ is left unchanged. Note that the perturbation phase is usually characterized by a perturbation radius r which has to be tuned carefully, being the main responsible of the global exploration capability of MBH. MBH usually terminates when for a certain number of iterations (denoted by *MaxNoImprove*) no improvement has been observed. In this case the refinement procedure R is a whole MBH run.

We might apply this approach to any global optimization problem, provided a suitable random generator of feasible starting points and a local optimization method are available. However, in order to give more evidence to the proposed approach, we choose to apply LeGO to box–constrained optimization problem only.

In this section we show in detail how LeGO can be applied to a pair of well known test functions. In the next section we will then perform a more systematic application on a wider test set.

We performed experiments with the Rastrigin and the Schwefel test functions. The aim of these experiments is that of checking whether the distribution of the values returned *after* training is better than the distribution of the values returned *during* the training phase.

In all the computational experiments, the `Libsvm` package for SVM classification and regression has been used (see for details (Chang & Lin, 2001)). The procedure used for training is a standard one proposed by the authors of the package. In particular, we first scale the data set to the interval $[-1, 1]$; then we proceed to the SVM training, using Gaussian kernel and a grid search to estimate the best values to be used as parameters in the training phase. All these operations can be easily performed by means of the `Libsvm` package.

In the following sections, the ratio between the percentages of points (of the whole data set) included in the training and validation sets respectively, is denoted as $\%_{tr}/\%_{val}$.

3.1 Experiments with the Rastrigin test function

The n–dimensional Rastrigin test is defined as

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} 10n + \sum_{i=1}^{n}(x_i^2 - 10.0\cos(2\pi x_i)) \\ &\qquad x_i \in [-5.12, 5.12] \quad , \quad \forall\ i \in 1, \ldots, n \end{aligned}$$

with the global optimum at the origin, with value 0.

The very high number of local minimizers of this function represents a serious challenge for the classical Multistart approach, while the problem becomes trivial when solved by means of MBH.

We started with a few experiments with $n = 2$. For these experiments, we ran 1 000 Multistart iterations, using L-BFGS-B (Byrd, Nocedal, Lu, & Zhu, 1995) as a local descent method; the resulting set of 1 000 starting points and 1 000 local optimum values were used to train a SVM using a data set partitioned as 75%/25%. Setting the threshold value for acceptation equal to 1, we obtained 61 (out of 750) positive samples. In Figure 1 we show the level curves of the 2–dimensional Rastrigin function. Crosses ($\times$) represent positive points (i.e., starting points leading to a local optimum of value less than the threshold 1), stars ($\star$) are negative points. The darker region around the center of the picture represents the set of starting points which are accepted by the trained SVM.

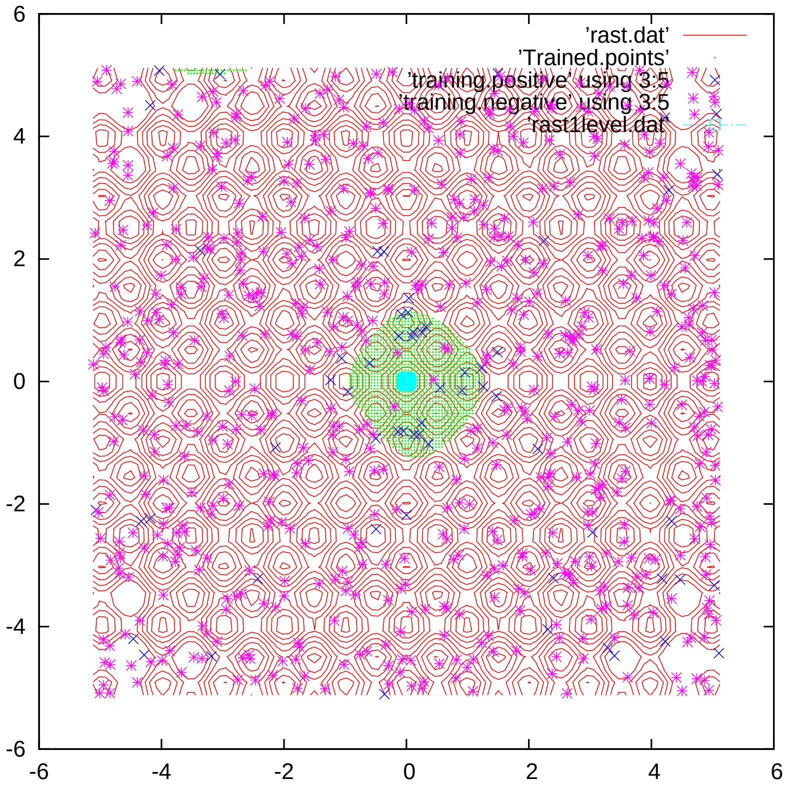


Fig. 1 Training a 2–dimensional Rastrigin function

From the figure it can be seen that training is quite accurate in identifying a set which, very likely, will lead L-BFGS-B to the global minimizer.

We then performed 10 000 Multistart runs at dimension $n = 10$, using a threshold 40 for considering a starting point a positive one and generated 10 000 acceptable start-

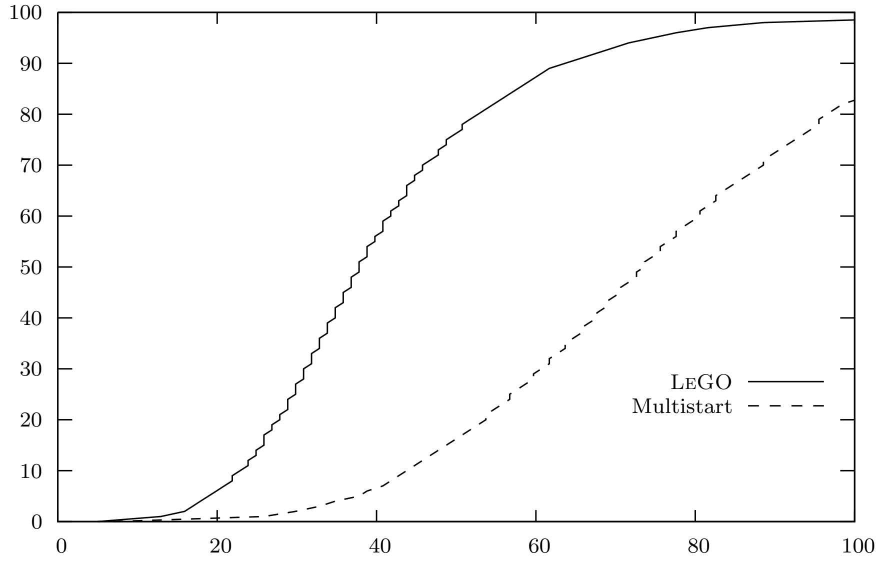


Fig. 2 Empirical distribution function for 10 000 runs with the Rastrigin function at $n = 10$

ing points after training. In Figure 2 we report the empirical cumulative distribution function (ECDF) of the optima found starting a local search from each of the 10 000 points accepted by LeGO versus the empirical distribution function obtained running 10 000 local searches from random starting points. We recall that at point x, the value of an ECDF represents the percentage of observations with values less than or equal to x. From the figure it is evident how training significantly improves in obtaining better starting points; the best local optimum observed from points generated by Lego has value 4.9748 while in 10 000 generic Multistart runs the record was 6.96474. Taking into account the threshold used for training, the percentage of LeGo runs leading to an optimum value not larger than 40 was 56.5%, while for standard Multistart these values were observed in 6.9% cases.

3.2 Experiments with the Schwefel test function

Let us consider now the n–dimensional Schwefel test problem:

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|})$$
$$x_i \in [-500, 500] \quad , \quad \forall\, i \in 1, \ldots, n$$

This is a classical test function whose global optimum value is $-418.9829n$, atteined at $x_i = 420.9687, i \in 1, \ldots, n$. From the point of view of global optimization, also this problem is quite trivial, as it is a separable one and thus it is sufficient to solve it for $n = 1$ and to replicate the one-dimensional optimum in order to find the global one. However this test turns out to be quite a hard benchmark for those methods which are unable to exploit separability. In particular, it has been observed in some papers,

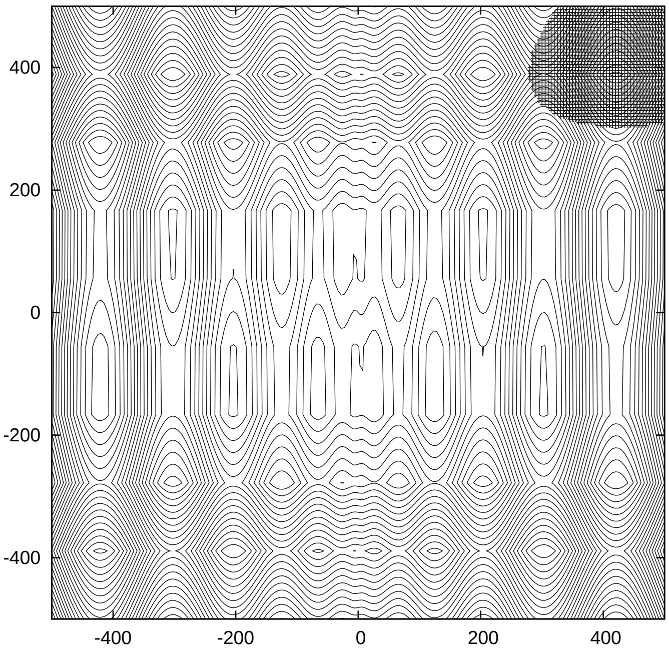


Fig. 3 Acceptance region of LeGO for the Schwefel function with $n = 2$

like, e.g., in (Addis, Locatelli, & Schoen, 2005), that MBH has serious difficulties in solving this problem, in particular when the dimension n increases. Examination of the graphs of Schwefel function for $n = 1$ or $n = 2$ reveals that this problem has a multiple funnel structure, so that simple, repeated, executions of MBH searches started from randomly selected points is doomed to fail as n increases, unless the number of restarts is prohibitively large. Of course different strategies should be employed in order to solve a problem like this, but our aim in this section is not to develop a new method for the Schwefel test function, but to show that machine learning is useful in improving the quality of an optimization method. What we would like to show is that after having executed MBH for a fixed number of times, a learning tool like LeGO might be successfully employed in order to generate promising starting points for MBH. As an illustrative example, we show in Figure 3 the region of acceptable starting points for LeGO after a suitable training.

The Schwefel-10 data

We begin with the analysis of the relatively easy case $n = 10$. In order to choose the perturbation radius r for MBH, we performed 1 000 randomly started MBH runs with *MaxNoImprove* set to 1 000 We observed that the global optimum was found with a quite low success rate both for relatively small values of the radius r and with significantly larger ones. This is not a surprise, as the multi-funnel structure of this function makes it hard to optimize with MBH. After these initial experiments we decided to fix the value $r = 130$ (26% of the box). We performed 1 000 runs to collect

set	positive (%)	negative	total
train	342 (45.6%)	408	750
valid	113 (45.2%)	137	250
total	455 (45.5%)	545	1000

Table 1 Training and validation sets for Schwefel 10–dim. function, $T = -3700$

set	positive (%)	negative	total
train	174 (23.2%)	576	750
valid.	48 (19.2%)	202	250
total	222 (22.2%)	778	1000

Table 2 Training and validation sets for Schwefel 10–dim. function, $T = -3800$

set	positive (%)	negative	total
train	59 (7.9%)	691	750
valid.	14 (5.6%)	236	250
total	73 (7.3%)	927	1000

Table 3 Training and validation sets for Schwefel 10–dim. function, $T = -3900$

pairs of first local optima – final value (a choice made in order to exploit the correlation between start and end points). The data set partitioned as 75%/25%. After training the LeGO SVM with threshold −3700, the situation in the training and validation data sets was that reported in Table 1.

We obtained a trained SVM which, for the validation set of 250 runs, gave us 76.45% accuracy, i.e., correct prediction of 191 out of 250 instances. The trained SVM has then been used to generate 1 000 starting points for MBH. The execution of these 1 000 runs lead to the global optimum only twice, the same frequency observed running MBH from randomly generated starting points. However, with the chosen threshold −3700, it was observed that, after training, the percentage of MBH runs which terminate at an acceptable local optimum (i.e., one with function value not greater than the threshold) grows from the original 46% to 76%.

In order to check the influence of the threshold on the SVM prediction capabilities, we repeated the test lowering the threshold first to −3800, obtaining the result as in Table 2, and then pushing down further to −3900, with the result reported in Table 3. The trend already observed with threshold −3700 gets more and more accentuated; in particular, with threshold −3800 the global minimum is found in 9 out of 1 000 trials and the percentage of runs leading to a final value which is below the threshold changes from 22% to 69%. Pushing the threshold further to −3900 this trend goes to quite astonishing results: not only the percentage of successful MBH runs (those leading to a final value not greater than −3900) grows from 7% to 75%, with 98% of runs below the original threshold of −3700, but also the global minimum is found as many as 83 times (in 1 000 runs), compared to the original 2 times. All of these results are reported in Figure 4 where the empirical cumulative distribution functions for the non trained MBH and the LeGO runs with different thresholds are compared.

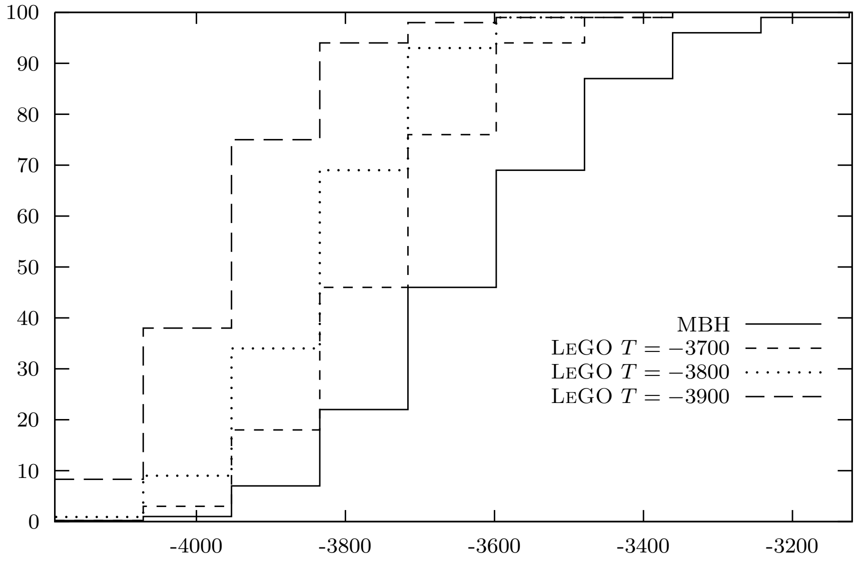


Fig. 4 Empirical distribution function of MBH and LeGO optimal values for the Schwefel 10–dim. function

The Schwefel-50 test function

In order to further test the effectiveness of the learning mechanism, we ran similar tests at higher dimension, testing MBH with the Schwefel function at dimension $n = 50$. This is a very hard test case for the MBH method and the global minimum cannot in general be found. We performed also in this case 1 000 independent MBH runs, with the same parameters chosen for the case $n = 10$; this seemed to be a correct choice, as the symmetry and separability of the objective functions lead us to believe that the same kind of perturbation should perform similarly for any dimension. This has been indeed confirmed by running MBH with different perturbation radii and, although the procedure never found the global minimum, it displayed the best results (-19527.9) for $r = 130$; we recall that in this case the global minimum is -20949.145. We kept *MaxNoImprove* equal to 1 000. Indeed, extending *MaxNoImprove* from 1 000 to 5 000 the best value observed was again the same. Performing more MBH runs can give some improvement: the best result after 10 000 runs is -19764.8 which is quite a small improvement if compared with a ten-fold increase in computational effort. These experiments once again confirm that due to the multi-funnel structure of this test problem, MBH is not the best algorithm. By the way, we notice that a pure Multistart method is in this case even worse: considering the fact that 1 000 MBH runs required 2 906 000 local searches, we ran 3 000 000 Multistart local optimizations, but the best observed value was just -15678.0, thus confirming the strong superiority of MBH over Multistart in these problems.

We started with a threshold value of $-18\,500$, which represents a 23% percentile of the frequency distribution of the observed local optima values (for $n = 10$ the 22% percentile gave a reasonable rate of success). From the trained SVM we obtained the following statistics:

set	positive (%)	negative	total
train	169 (25.5%)	581	750
valid.	58 (23.2%)	192	250
total	227 (22.7%)	773	1000

The results are not particularly impressive in this case: in Figure 5, a slight improvement is visible, but the behaviour of the trained method is quite similar to the untrained one.

This might be due to the much higher difficulty of this test case as well as to the fact that a 50-dimensional box is enormously larger than a 10-dimensional one, so that learning from a data set containing the same number, 1 000, of instances might be too optimistic. However, we tried to exploit the information contained in the data set and we lowered the threshold to $-19\,000$ (a 3.3% percentile); as quite expected, the trained SVM turned out to be a trivial one (always refuse). We thus exploited the possibility to give more weight to some observations during the training phase. We choose to assign weight 5 to positive instances and 1 to negative ones; moreover, in order to avoid the case of a validation set with no positive instances, we split the data set evenly (i.e., 50%/%50). The obtained accuracy of the trained SVM has been the 96.8% in the validation set, which, however, was not trivial. With this SVM we generated 1 000 starting local optima and run MBH from these. Using this method we obtained an improvement in the best local optimum observed. In particular we obtained in one case -19883.2, in 2 instances -19764.8 and in 11 the value -19646.3. So we were able to observe, for as many as 14 times, an improvement over the best observed value. This is remarkable because one of the possible critic to this approach is that it only allows to replicate the best results of the training phase. This is not the case: the acceptance region is large enough to include points from which improvements can be attained. Moreover, the behaviour of the method systematically leads to low function values , as can be seen in Figure 5 where the results of this experiment are compared with those of MBH.

More in general, if the selected algorithm during the training phase often reaches a good region without hitting the global minimum, then through learning we might be able to reach the global minimum or, at least to improve with respect to the training phase, if the region of improvement (i.e., the region containing starting points for the selected algorithm leading to better results) is not ”too far” from the region where good starting points have been detected in the training phase (which is confirmed by the improvements attained in the MBH runs over Schwefel-50 but also by the improvements attained by Multistart reported in the following Section 4). On the other hand, we also need to point out that the detection of the global minimizer might be impossible if this lies in a region completely unrelated with the good starting points detected during the training phase.

4 Systematic numerical experiments

In order to gain confidence on the feasibility of the proposed approach, in this section we observe the behaviour of LeGO on standard global optimization test sets. These experiments will also give us the possibility of outlining a general procedure for chosing the most relevant parameter of the approach, namely the threshold T used to distinguish between negative and positive examples.

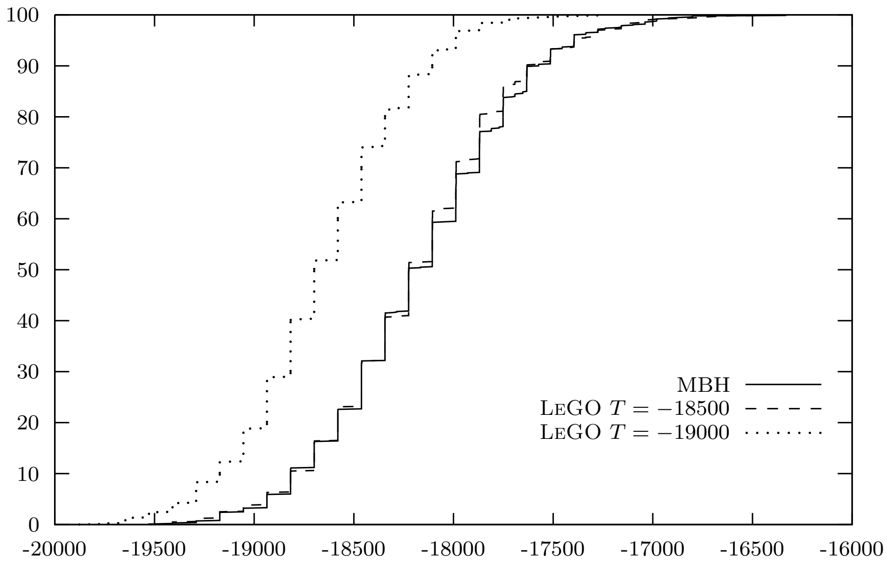


Fig. 5 Empirical distribution function for Schwefel $n = 50$

For what concerns standard test functions for global optimization, the situation in the literature is quite unsatisfactory. There are indeed collections of test problems, but most of them are by far too easy to solve by means of modern global optimization techniques. In order to test the capabilities of the approach proposed in this paper we looked for a test set which was reasonably difficult to be a challenge for global optimization methods, but not too difficult to prevent any possibility of learning from a first, small, set of runs.

After browsing the literature, and discarding the test sets proposed by some of the authors of this paper, we choose the test set cited in (Vaz & Vicente, 2007). Of these tests, we decided to exclude all non differentiable ones as well those with too few (less than or equal to 5) or too many (more than 50) variables. After this selection we were left with 25 box constrained test problems. Almost of all these problems were trivially solved with standard MBH runs. So the decision was taken to test the effectiveness of LeGO with Multistart. We performed a first set of 10 000 Multistart trials for each test function, using LBFGS-B as a solver. The results are summarized in Table 4, where problem names refer to those published in (Vaz & Vicente, 2007) and colum "% succ" represents the observed percentage or runs leading to the global minimum, from which it is easy to derive the estimated probability of getting to the global minimum with a single local search: From this table it is evident how easy it is for the most elementary method for global optimization to find the global optimum in most cases; we can thus observe the absolute inadequacy of standard global optimization test problems. In order to run significant experiments, we choose to run LeGO only for the most difficult tests, namely for 8 instances for which Multistart had at most 1% success rate.

For what concerns training, we choose to use the following procedure, which we adopted consistently for the 8 difficult instances:

problem	n	% succ	problem	n	% succ
ack	10	0.00	em_10	10	0.00
exp	10	100.00	fx_10	10	0.00
gw_20	20	8.69	h6	6	68.58
lm2_10	10	3.79	mgw_10	10	1.00
mgw_20	20	0.14	ml_10	10	0.00
nf3_10	10	100.00	nf3_15	15	100.00
nf3_20	20	100.00	nf3_25	25	100.00
nf3_30	30	100.00	pp	10	100.00
ptm	9	1.99	rb	10	83.80
rg_10	10	0.00	sal_10	10	0.03
sin_10	10	15.98	sin_20	20	4.98
xor	9	60.87	zkv_10	10	100.00
zkv_20	20	100.00			

Table 4 Test problems and related percentage of successes for Multistart

1. build the data set from 10 000 Multistart runs using the (uniform random) starting point associated with the local optimum value found by the local optimization method employed
2. normalize the data, by means of the `svm-scale` tool
3. split the data set as 75%/25%;
4. for every possible choice of the threshold parameter T from the 70% to the 90% percentile (in steps of 5) and for every choice of the weight for positive instances (as already showed for Schwefel-50 test function), ranging from 2 to 10, `svm-train` was run with default parameters;
5. after this grid search in the space of parameters, the classification error was computed on the validation set. A score was obtained through a weighted sum of false positives and false negatives, with weight 1 for false positives and weight 2 for false negatives (this way we considered a more dangerous error discarding a good starting point than accepting a bad one). Using this score, the best pair of values for the threshold and the weight was retained.

After training, for each of the 8 test functions we uniformly generated points until we obtained 5 000 starting points accepted by `svm-predict` and 5 000 refused. After this sampling, we ran Multistart from both sets and collected the results. Some statistics about such results are reported in Table 5.

It is quite evident from the table that starting a local search from points which are accepted from the trained svm delivers significantly better results than starting from points which are refused. It seems, from the table, that only for problem `ml_10` the situation is unclear. Indeed this problem is extremely peculiar, as it consists of an objective function which is pretty constant and close to 0 in most of the feasible set, with the exception of a very small region around the global optimum, where function values go to -0.965. In this case it is reasonable to guess that no learning is possible, as finding the global minimum is just a matter of guessing the correct starting point within a region of attraction whose volume is almost negligible. We note in passing that for the `ack` and `rg` test cases the best optimum found from accepted starting points is strictly lower than that found in the training and validation sets.

In order to confirm our impression on the clear advantage in using trained starting points, we performed a one–sided Kolmogorov–Smirnov test on the empirical distribution function of the 5 000 results obtained starting from accepted and from refused

Problem	set	Min.	1^{st} Quartile	Median	Mean	3^{rd} Quartile	Max
ack	Accepted	2.04	4.54	4.85	4.74	5.04	5.36
ack	Refused	4.59	5.66	6.03	6.06	6.41	7.97
em_10	Accepted	-8.88	-6.13	-5.24	-5.16	-4.25	-0.488
em_10	Refused	-8.68	-4.96	-4.18	-4.12	-3.31	0.002
fx_10	Accepted	-10.21	-2.13	-1.48	-1.97	-1.48	-1.28
fx_10	Refused	-10.21	-1.48	-1.48	-1.58	-1.48	-1.15
mgw_10	Accepted	4.4e-16	3.9e-03	8.9e-03	1.9e-02	1.8e-02	3.63
mgw_10	Refused	4.4e-16	1.7e-02	3.2e-02	4.0e-02	5.0e-02	3.63
mgw_20	Accepted	-1.3e-15	7.9e-03	2.5e-02	7.4e-02	9.7e-02	7.80
mgw_20	Refused	-1.3e-15	2.2e-02	4.4e-02	8.4e-02	1.1e-01	9.42
ml_10	Accepted	-1.7e-22	-3.8e-86	-1.0e-132	1.7e-22	3.6e-94	8.6e-19
ml_10	Refused	-8.3e-81	-1.2e-160	1.9e-279	1.6e-74	1.2e-152	8.2e-71
rg_10	Accepted	6.96	44.77	57.71	57.54	68.65	127.40
rg_10	Refused	9.95	64.67	80.59	81.15	96.51	224.90
sal_10	Accepted	2.1e-16	13.80	15.10	14.47	16.10	20.90
sal_10	Refused	1.2e-14	17.60	18.90	18.65	20.40	26.60

Table 5 Statistics on the optimal values found from 5 000 points accepted by LeGO and from 5 000 refused ones

points. The null hypthesys we tested was that the optima obtained from accepted points are stochastically smaller than those obtained from refused one or, equivalently, that the cumulative distribution function of the optima obtained from accepted values is pointwise greater than that of optima obtained from refused ones. The test is based on the statistics

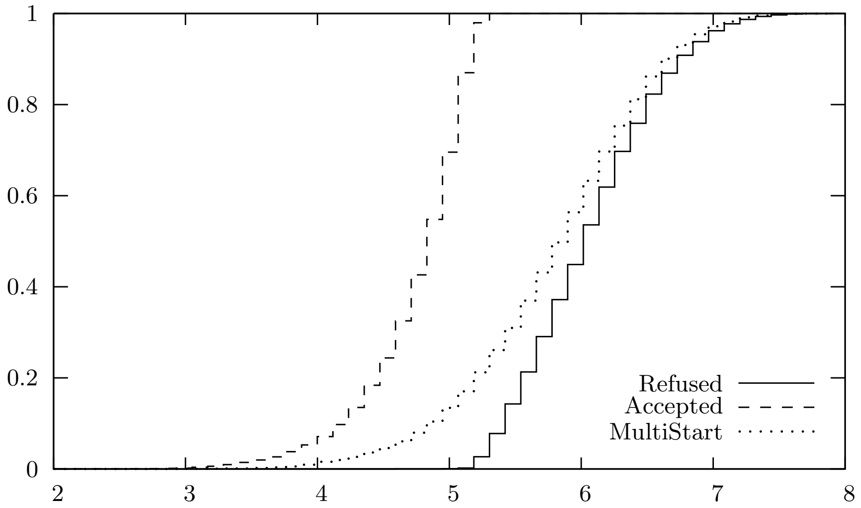
$$D^{+} = \max_{u}(F_{Acc}(u) - F_{Ref}(u))$$

where $F(\cdot)$ indicates the empirical distribution function and the subscripts Acc and Ref refer to points which are respectively accepted or refused by LeGO. The result of the test is the following:

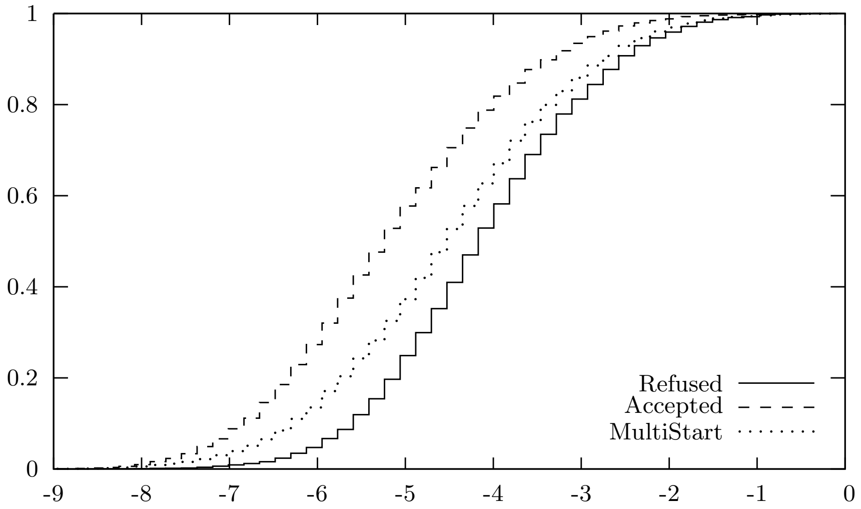
problem	ack	em_10	fx_10	mgw_10	mgw_20	ml_10	rg_10	sal_10
D^{+}	0.9564	0.3338	0.433	0.4928	0.2762	0.423	0.4486	0.741

In all 8 cases the p–value was negligible ($< 2.2e - 16$) so that the results are different with very high level of significance.

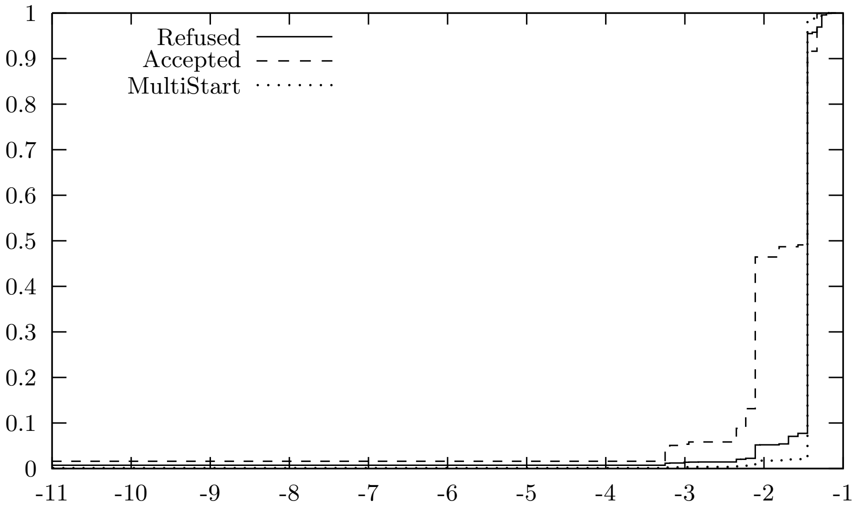
In order to further confirm this analysis, in the following figures we present cumulative distribution functions for the function values returned by pure Multistart (dotted lines), Multistart from points accepted by the classifier (dashed lines), and from points refused by the classifier (solid lines). As expected, the cumulative distribution functions for pure Multistart usually lie exactly in between those for Multistart from points accepted and refused by the classifier.



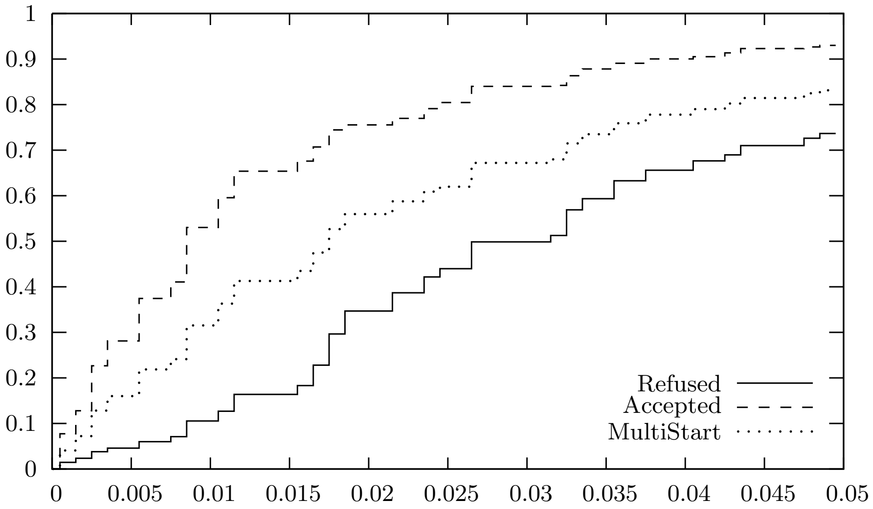
ack - Ackley function - empirical distribution plot



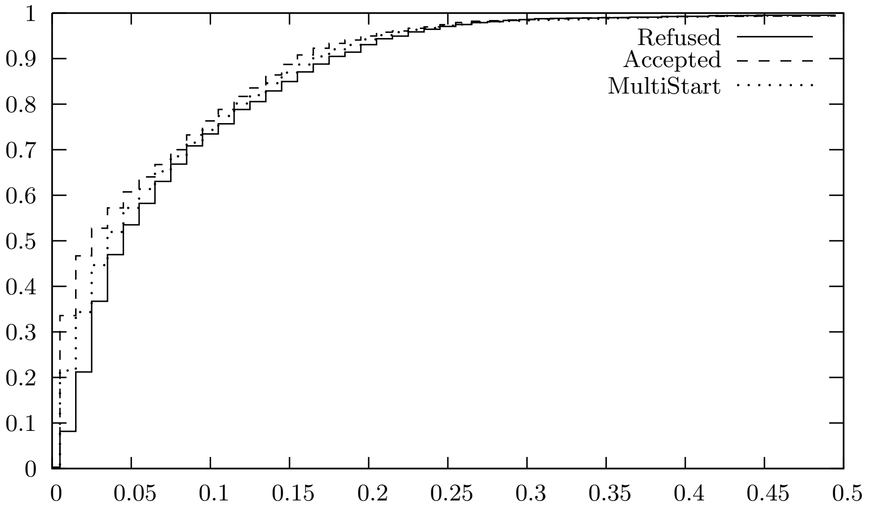
em_10 - Epistatic Michalewicz function - empirical distribution plot fx_10 - Shekel FoxHoles function - empirical distribution plot



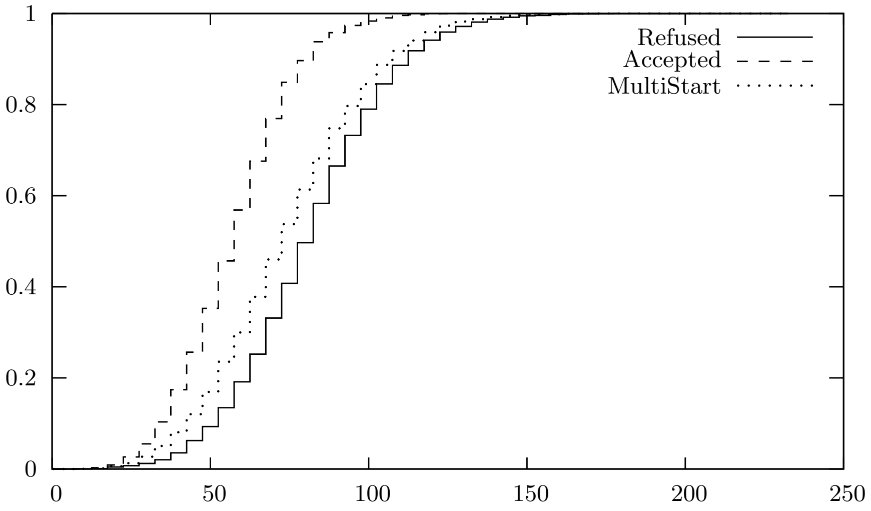
mgw_10 - Modified Griewank (dim 10) function - empirical distribution plot

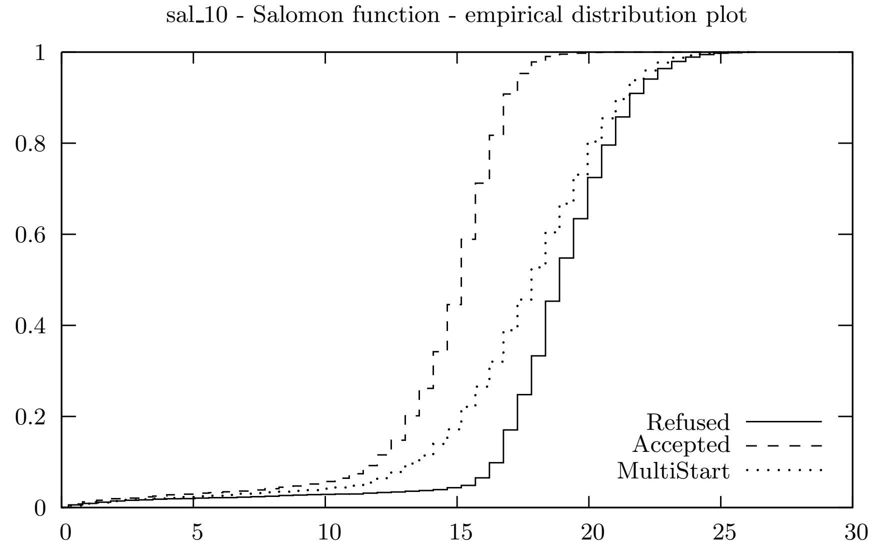


mgw_20 - Modified Griewank (dim 20) function - empirical distribution plot



Rastrigin function - empirical distribution plot





5 Space trajectory design

Although the experiment made on standard test functions seem to support with strong evidence the validity of the LeGO approach, in order to obtain further and possibly more significative confirms, we choose to try the proposed approach on a difficult real-life problem arising in the design of planet to planet missions. This problem consists in finding suitable parameter values which are used to define a trajectory for a space vehicle. The literature on this subject is quite vast – we refer the readers, e.g., to (Addis, Cassioli, Locatelli, & Schoen, 2009; Izzo, Becerra, Myatt, Nasuto, & Bishop, 2007; Olympio & Marmorat, 2008; Vasile, 2005) for references on this subject. Here it may suffice to say that this is a family of very hard global optimization problems with relatively few variables (a few tens at most) but a huge amount of local optima. Moreover, in general, no higher order information, like gradients, on the problem is available, so that either derivative-free codes have to be used or, alternatively, finite differences have to be employed. Luckily, although function evaluation requires the numerical solution of a system of differential equation, usually this is extremely fast, so that, apart from the well known numerical difficulties, central or forward finite difference derivative estimation can be employed. The Advanced Concept Team at ESA, the European Space Agency, maintains a web site where many instances of trajectory optimization problems are available, in the form of Matlab or *C* source code (see `http://www.esa.int/gsp/ACT/inf/op/globopt.htm`).

We focus here on the Tandem mission test set, which consists in 24 possible planet sequences, ending on Saturn, for which the aim is to maximize the mass of the vehicle at the arrival (or, equivalently, minimizes the fuel mass consumption).

Each sequence is then considered either as a box-constrained problems or a linearly constrained one, in which the overall travel time is limited to ten years and computed as a sum of the intermediate trip durations. For the purpose of this paper we made experiments on the currently best planet sequence (Earth-Venus-Earth-Earth-Saturn) for the box constrained case. This problem has 18 variables and the current record corresponds to a final mass of 1 606.59 kg, starting from a 2 000 kg initial mass.

The first trials, starting from a database of 1 000 MBH runs performed as described in (Addis et al., 2009), were prepared using the same scheme as before, with a partition of the starting point set into 500 for training and 500 for validation. Unfortunately, with these data the first attempts to train an SVM typically lead to a failure, which consisted in having a SVM which refuses every point, at least for reasonable choices of the threshold used to distinguish acceptable from unacceptable points. Using a threshold of very low quality (e.g., close to 0, which corresponds to 100% mass consumption, lead to a reasonable SVM, but with only moderate usefulness in generating new starting points. This behaviour might be linked to the fact that these MBH runs are very long (on average roughly 2 000 steps) and, also due to the extremely rugged surface of the objective function, there seems to be scarce correlation between starting points and final value of the objective function. So we tried to use, for learning, the information collected after a few (we, somewhat arbitrarily, choose 100) steps of MBH. In other words, procedure G was implemented as a random uniform generation in the box, followed by 100 steps of MBH. At the end of these 100 steps, we recorded the current point and trained the SVM based upon this information, coupled with the final label corresponding to the fact that the final optimum was below or above a threshold. With these data we obtained the classification results reported in Table 6, using as a threshold -1400 kg (we transformed the original mass maximization problem into one of minimization by changing the objective function sign):

	Training			Validation		
Predicted:	+	-	%	+	-	%
Positive	98	3	97.02	81	15	84.38
Negative	7	392	98.25	17	387	95.79

Table 6 Classification results on 1 000 MBH run for space trajectory optimization

After training, we started 50 000 runs of MBH, stopping each of them after 100 iterations and checked, by means of SVM, which of those points were acceptable. A total of 1 788 points were selected this way and, from each of them, MBH was restarted up to the natural termination (which, both here and in training, was chosen to happen after 500 iterations of MBH with no improvement were observed). The results obtained this way confirm what expected: starting points obtained from LeGO tend to produce high quality final points. In Figure 6 we report curves representing the cumulative distribution function of observed local optima; the figure reports an empirical distribution function obtained from 10 000 independent executions of MBH and the one corresponding to the 1 788 runs of MBH started from points accepted by LeGO. It is evident from the figure that LeGO tends to generate very good points, much better and much more frequently than standard MBH; it is seen from the figure that only slightly more than 40% of the normal MBH runs lead to optimal values strictly greater than 0 (which corresponds to consuming all of the mass of the spacecraft). On the contrary, it can be observed that starting from trained points leads to a final mass higher than 1 400kg in more than 50% of the runs. By the way, having several good solutions, not necessarily the global one, is a characteristic usually required by aerospace engineers, who prefer to have some possibility of choosing among many good solutions rather than being forced to use the unique global one.

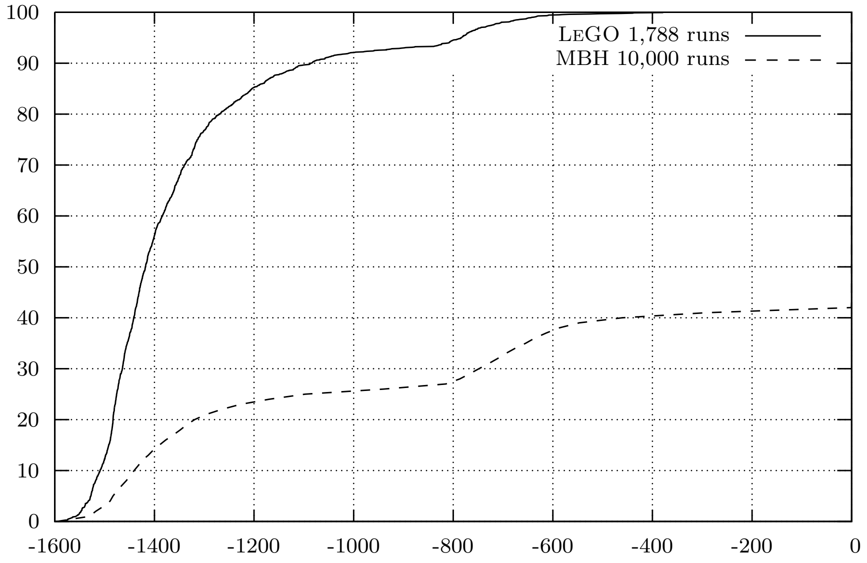


Fig. 6 Comparison between function values obtained by LeGO and MBH on trajectory planning

It might be argued that the total effort required in generating 50 000 points by means of short MBH runs wipes away the savings obtained. However, a count of the total number of local searches performed reveals this is not the case. To generate the sample we ran 50 000 of times 100 local searches, which accounts for a total of five million local searches. To run MBH from the selected 1 788 points we needed 1 342 local searches per run (on average). The total effort in generating points and executing MBH from those selected was 7 388 365 local searches. In contrast, the 10 000 pure MBH runs reported in Figure 6 required a total of 10 255 863 local searches. Thus, not only this procedure saved roughly 30% of the overall computational costs, but also systematically produced much better optima. For what concerns CPU times, we observe that usually points generated by means of the proposed training procedure, tend to be quite good; on the contrary, starting from a random uniform point, it is quite likely that there will be no improvements, so that the total number of local searches required is, in quite a high number of cases, exactly equal to the *MaxNoImprove* parameter. Thus, starting a prefixed number of MBH runs, although requiring more local searches, might seem to be faster than starting an equivalent number of MBH runs from points generated by the learning method. Of course, the resulting quality will typically be significantly different, but, in any case, one might wonder whether the whole process is worthwhile. In the following we summarize the CPU times (measured on an Intel Core 2 Quad CPU running at 2.66GHz):

- Generation of 1 000 points obtained as MBH runs stopped at the 100-th iteration: $26'22''$
- Executing full MBH runs starting from those points in order to build the training and validation sets: $665'2''$
- Training via `Libsvm`: $0'3''$

- Generation of 50 000 starting points through MBH runs stopped at the 100-th iteration: $1343'41''$
- Executing full MBH runs from the 1 788 accepted starting points: $2427'48''$

The complete procedure, thus, required $4463'16''$. In order to make a comparison, performing 10 000 standard MBH runs required $6837'15''$. So it is seen that the whole procedure based upon training required 71% of the local searches and 65% of the CPU time with respect to a standard run of MBH which consistently produced results of much lower quality.

Conclusions and further developments

In this paper we proposed an innovative framework (LeGO) to be coupled with standard global optimization methods in order to increase their capability of quickly exploring promising regions. Although the idea is indeed very simple, its success was not easy to predict. The existence of a quite strong relationship between the starting point of a complex optimization procedure like MBH and the final optimum value came as a surprise. In simple minded methods like Multistart this might have been argued, as, at least in an ideal case, a local search is a deterministic procedure which should lead to a local optimum belonging to the same region of attraction as the starting point; so a deterministic dependency exists between starting points and local optima values and machine learning can be used to approximate this dependency. On the contrary, in more refined methods like MBH, not only the path followed by the method is, in general, "non local" and very long steps, crossing many regions of attraction, are usually performed, but also the relationship between starting and final point is a non deterministic one, as random moves are performed during the algorithm. So the same starting point might lead to radically different optima. This behaviour might in theory be analyzed, looking at MBH as the realization of a Markov chain in the (hopefully finite) state space of local optima. Learning can thus be seen as a method to approximate the probability that the system, started at a specific local optimum, will eventually be absorbed in a state corresponding to a local optimum whose value is below a threshold. We did not pursue further this analysis, but we plan to do this in the future; here we remark that in our experiments we observed that a LeGO approach can be applied with significant success even in this difficult probabilistic framework.

Although the experiments reported in this paper show, in our opinion, that the proposed approach has a great potential, many issues remain to be addressed and will be analyzed in future papers. One deals with LeGO training: as we saw, training usually has a negligible cost in comparison with optimization; however, as optimization runs are performed, the size of the available data increases, so that one might wish to re-train the SVM. However, in this case, soon or late the cost of training might become very significant. It might be interesting, thus, to develop incremental techniques which re-train an already trained SVM after new observations become available. If re-training is efficient enough, we might think of a global optimization procedure with two parallel processes, one composed of the original optimization method and another one which, in parallel, trains the SVM and generates new optimization runs from trained points. Another issue is related to the generation of starting points from a trained SVM: we used a straightforward acceptance/rejection method, but, thanks to the fact that the SVM has a known analytical expression, direct random generation methods might be developed. Another research direction deals with the extension of this approach to problems with constraints different from simple bounds on the variables: in principle, nothing changes in the approach, but the numerical difficulties in generating good starting points might be significantly higher. Still another research issue deals with the development of suitable methods to choose the parameters of the SVM – actually we used a blind grid search, but in some applications knowledge of the problem might enable us to guess reasonable values, at least for some of the parameters. Finally, for what concerns the application to space trajectory design, after an initial failure, we obtained a successful LeGO method by learning from the 100-th iteration of MBH. The choice of these first 100 local searches was totally arbitrary and more research should be carried on in order to obtain a reasonable guideline for further application. One possibility which we are currently exploiting, is that of inserting in the training set not just the starting point of MBH, but several points encountered during the run (e.g., all points which lead to an improvement); this way the training set can be enlarged with no effort except storage and learning the structure of MBH might become much easier, with no need of modifying the standard LeGO approach.

In conclusion, we think this paper is a starting point for the development of new methods which, coupled with global optimization algorithms, can significantly augment their efficacy.

Acknowledgements

This research has been partially supported by Progetto PRIN “Nonlinear Optimization, Variational Inequalities and Equilibrium Problems”

It so common to thank referees for their suggestions that often this seems to be just a formality; in our case this is not: we are really and sincerely indebted to all three referees, as their comments required us to radically change many parts of this paper and to add an entire new section: the result is that the paper improved in a very significative way from its first submission. Each of the referees was extremely critic but, in the same time, really constructive: we are indebted towards all of them for the truly precious suggestions.

References

Addis, B., Cassioli, A., Locatelli, M., & Schoen, F. (2009). A global optimization method for the design of space trajectories. *COAP*, *published on line.*

Addis, B., Locatelli, M., & Schoen, F. (2005). Local optima smoothing for global optimization. *Optimization Methods and Software*, *20*(4–5), 417–437.

Ampatzis, C., & Izzo, D. (2009). Machine learning techniques for approximation of objective functions in trajectory optimisation. In *Proceedings of the ijcai-09 workshop on artificial intelligence in space* (pp. 1–6).

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, *2*(2), 121–167.

Byrd, R. H., Nocedal, J., Lu, P., & Zhu, C. (1995). *L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization* (Tech. Rep.). Northwestern University, Department of Electrical Engineering and Computer Science.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines.* (Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`)

Dill, K. A., Phillips, A. T., & Rosen, J. B. (1997a). Cgu: An algorithm for molecular structure prediction. In L. T. Bigler, T. F. Coleman, A. R. Conn, & F. N. Santosa (Eds.), *Large-scale optimization with applications. part 3: Molecular structure and optimization.* (Vol. 94, pp. 1–21). Springer.

Dill, K. A., Phillips, A. T., & Rosen, J. B. (1997b). Protein structure and energy landscape dependence on sequence using a continuous energy function. *Journal of Computational Biology*, *4*(3), 227-240.

Grosso, A., Locatelli, M., & Schoen, F. (2007). A population based approach for hard global optimization problems based on dissimilarity measures. *Mathematical Programming*, *110*(2), 373–404.

Izzo, D., Becerra, V., Myatt, D., Nasuto, S., & Bishop, J. (2007). Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories. *Journal of Global Optimization*, *38*, 283–296.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, *9*(1), 3–12.

Jin, Y., Olhofer, M., & Sendhoff, B. (2002). A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, *6*(5), 481–494.

Leary, R. H. (2000). Global optimization on funneling landscapes. *Journal of Global Optimization*, *18*, 367–383.

Lourenço, H. R., Martin, O. C., & Stülze, T. (2003). Iterated local search. In F. W. Glover & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 321–353). Boston, Dordrecht, London: Kluwer Academic Publishers.

Mangasarian, O. L., Rosen, J. B., & Thompson, M. E. (2005, May). Global minimization via piecewise-linear underestimation. *Journal of Global Optimization*, *32*(1), 1-9.

Olympio, J. T., & Marmorat, J.-P. (2008, January). *Global trajectory optimization: can we prune the solution space when considering deep space manoeuvres?* (Final Report). ESA.

Rinnooy Kan, A. H. G., & Timmer, G. T. (1987a). Stochastic global optimization methods. Part I: Clustering methods. *Mathematical Programming*, *39*, 27–56.

Rinnooy Kan, A. H. G., & Timmer, G. T. (1987b). Stochastic global optimization methods. Part II: Multi level methods. *Mathematical Programming*, *39*, 57–78.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond.* The MIT Press.

Vapnik, V. N. (1998). *Statistical learning theory.* J, Wiley & Sons.

Vasile, M. (2005). Design of Earth-Mars transfer trajectories using evolutionary-branching technique. *Acta Astronautica*, *56*, 705–720.

Vaz, I. F., & Vicente, L. N. (2007). A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*, *39*, 197–219.