Simple meta-heuristics using the simplex algorithm for non-linear programming

João Pedro Pedroso

Technical Report Series: DCC-2007-6
http://www.dcc.fc.up.pt/Pubs/

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL
Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

Simple meta-heuristics using the simplex algorithm for non-linear programming

João Pedro PEDROSO*
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
R. Campo Alegre, 1021/1055,
4169-007 Porto, Portugal
jpp@fc.up.pt

May 2007

Abstract

In this paper we present an extension of the Nelder and Mead simplex algorithm for non-linear programming, which makes it suitable for both unconstrained and constrained optimisation. We then explore several extensions of the method for escaping local optima, and which make it a simple, yet powerful tool for optimisation of nonlinear functions with many local optima.

A strategy which proved to be extremely robust was random start local search, with a correct, though unusual, setup. Actually, for some of the benchmarks, this simple meta-heuristic remained as the most effective one. The idea is to use a very large simplex at the begin; the initial movements of this simplex are very large, and therefore act as a kind of filter, which naturally drives the search into good areas.

We propose two more mechanisms for escaping local optima, which, still being very simple to implement, provide better results for some difficult problems.

1 Introduction

The usage of the Nelder and Mead's simplex algorithm [10] for non-linear optimisation as a part of a meta-heuristic is not new; several authors have suggested the inclusion of simulated annealing ideas in it[13, 4]. More recently, it has been used as a basis for applying the scatter search method [8] and tabu search [5]. The results reported in the literature are generally very encouraging.

We suggest further improvements to the simplex algorithm, in particular for tackling constraints, as most of the non-linear problems (NLP) with practical interest are constrained problems. Previously published works used penalties on infeasible solutions as a way of handling constrained problems; this, however, is not general, as it requires the knowledge of bounds on the objective function. In this work we propose a scheme for classifying the solutions of an NLP that takes into account both feasibility and the objective value. Based on it, we are able to sort the points of the simplex even in the case where some points are feasible and other points are not. This makes possible determining through which point of the

*Researcher at INESC - Porto

simplex should reflection, expansion, or contraction be done, thus allowing straightforward application of the Nelder and Mead's algorithm.

The simplex algorithm is a local search method, and does not encompass a procedure for escaping local optima. Random-start iterated local search provides a simple way for avoiding local optima.

We investigate on the performance of the modified simplex algorithm, as well as iterated local-search based on it. We propose additional procedures for trying to get away from local minima.

The idea used in the Nelder and Mead algorithm's extension for constrained problems can be further exploited, allowing us to deal with tabu search in a similar setting. Making some areas on the vicinity of local optima as tabu areas, however, did not lead to good performance. We rather propose the opposite idea: the exploitation of the areas of the current local optimum instead of their avoidance.

2 The simplex method

The simplex method for NLP is a very simple and elegant algorithm for finding the minimum of a function in a multi-dimensional space. It works based only on function evaluations, hence does not require information concerning the gradient of the function.

In an N-dimensional space, a simplex is a polyhedron with exactly $N + 1$ vertices (in general, we will be interested in non-degenerate simplexes, which have a finite volume in its space).

The algorithm, depicted in Algorithm 1, starts with a simplex with $N + 1$ points, and evaluates its vertices. It will then, at each iteration, try to get rid of its worst point, by applying the following steps:

1. Reflect the simplex into the opposite side of the worst point (top-left image at figure 1, line 5 of Algorithm 1).

2. If the reflection led to a point which is better than the simplex's best point, then try expanding further in that direction (top-right image at figure 1, line 7 of Algorithm 1).

3. If the reflection led to a point which is worse than the second-worst point, then contract the simplex in one-dimension, moving the worst point in the direction of the simplex's centroid (bottom-left image at figure 1, line 9 of Algorithm 1).

4. If this contraction did still not improve the simplex's second-worst point, then do a multi-dimensional contraction, by moving all the points in the direction of the best point (bottom-right image at figure 1, line 13 of Algorithm 1).

These steps are repeated until a specified stopping criterion is satisfied. The more frequently used criteria are based on the slack between the best and the worst point, on the distance the simplex's centroid moved, or on the number of function evaluations.

For a more complete description of the method please refer to [10], and for a computational implementation to [12].

In the remain of this paper we will assume that there are box constraints on the nonlinear problems, i.e., for a problem of dimension N there will be constraints:

$$l_i \leq x_i \leq u_i \quad i = 1, \ldots, N. \tag{1}$$

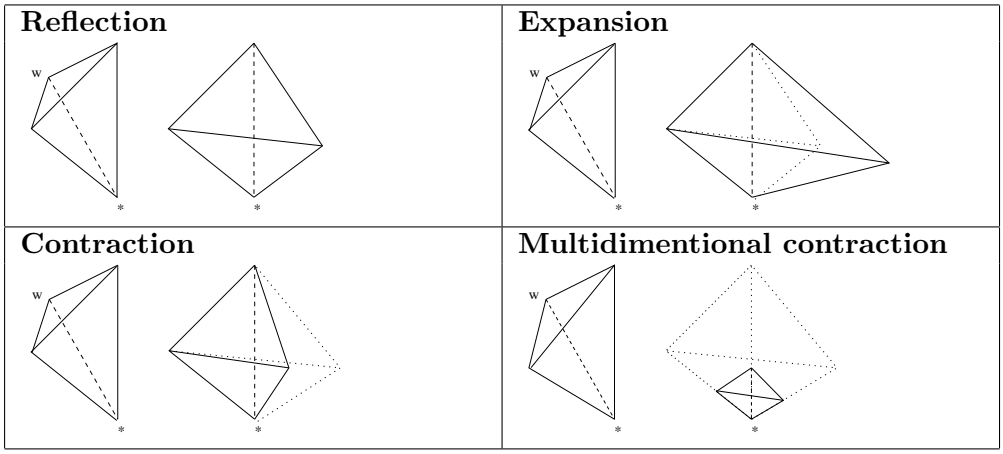

Figure 1: Operations on a simplex. In the initial simplex (on the left side), the best point is indicated by $*$, and the worst point is indicated by w.

In this context, a random solution for any problem is an N-dimensional vector that can be drawn as

$$x_i = \mathscr{U}(l_i, u_i) \quad i = 1, \ldots, N, \tag{2}$$

where we denote a random number with uniform distribution in the interval $[a, b]$ as $\mathscr{U}(a, b)$.

In Algorithm 1 s^* is the best point in the simplex, s' its worst point, and s'' its second worst point. The simplex centroid is $\bar{s} = \frac{1}{N} \sum_{j=1}^{N} s^j$. On line 2 we denote the j-th unit vector by e^j. For the calculation of the stopping criterion (line 4), we use absolute deviations if the objective values are less than one, relative deviations otherwise.

In all the experiments reported in this paper, the usual simplex expansion and contraction parameters were applied: $\alpha = 1, \gamma = 2, \beta = 1/2$.

Algorithm 1: The Nelder and Mead simplex algorithm, for doing a local search starting from point x.

SimplexLocalSearch(x, ϵ, λ, M)

(1) $s^0 = x$
(2) $s^j = x \pm \lambda\,(u_j - l_j)\,e^j, \quad j = 1, \ldots, N$
(3) $k = 0$
(4) **while** $|f(s^*) - f(s')| < \epsilon$ **and** $k < M$
(5) $\quad r = (1 + \alpha)\,\bar{s} - \alpha\,s''$
(6) $\quad$ **if** r better than s^*
(7) $\quad\quad s^* = \gamma\,r + (1 - \gamma)\,\bar{s}$
(8) $\quad$ **else if** r worse than s''
(9) $\quad\quad r = \beta\,s' + (1 - \beta)\,\bar{s}$
(10) $\quad\quad$ **if** r better than s''
(11) $\quad\quad\quad s' = r$
(12) $\quad\quad$ **else**
(13) $\quad\quad\quad s^j = (s^j + s^*)/2, \quad \forall j : s^j \neq s^*$
(14) $\quad$ update s^*, s', s''
(15) $\quad$ k = k+1
(16) **return** s^*

2.1 Classification of NLP solutions

Generally, points on the simplex are ordered according to the value of the objective function, possibly added to a penalty, if the problem is constrained and the solution is infeasible. Notice, however, that the simplex algorithm does not require the actual value of function evaluation at each of the points; all that is required is to *order* the simplex's points, for determining through which vertex should reflection, expansion, or contraction be done. This is a common characteristic to all direct search methods [9].

If in addition to the box constraints there are P more general constraints in the form $g_p(x) \leq 0$, for $p = 1, \ldots, P$, then the total constraint violation for a solution x can be assessed by

$$\delta(x) = \sum_{p=1}^{P} \max(g_p(x), 0) + \sum_{i=1}^{N} [\max(x_i - u_i, 0) + \max(l_i - x_i, 0)]. \quad (3)$$

We propose that the comparison of solutions should be based on this value, as well as on the objective value. For two different solutions x and y, we say that x improves y if and only if $\delta(x) < \delta(y)$, or $\delta(x) = \delta(y)$ and the objective value of x is better than that of y.

Based on this classification scheme, we are able to order the points of the simplex, even if some points are feasible and other not, and directly apply the algorithm to constrained problems. Notice that equality constraints are poorly dealt by this modified method. The simplex will probably converge into a point which is on the surface defined by the equality, but will likely have much trouble for moving on that curve, in order to improve the value of the objective, without increasing infeasibilities. The classification system was used in [11]; a more elaborate method would be the filter method proposed in [2].

2.2 Preliminary results

One possibility for using the Nelder and Mead algorithm is to start from a random point, as described in Algorithm 2.

Algorithm 2: The simplex search starting from a random point.
SIMPLEX(ϵ, λ, M)
(1) read problem's data ($N, f, \ldots$)
(2) x = random solution (Equation 2)
(3) x^* = SIMPLEXLOCALSEARCH(x, ϵ, λ, M)
(4) **return** x^*

Using the solution classification method described in the previous section, the search can start with a very large simplex, as this method tackles the danger of getting out of bounds. Hence, we propose a setting were the initial step is very large: we set $\lambda = 1$. This implies that all the points except the initial random point will be out of bounds, but now this is no longer a problem. Computational experiments have shown that this setup improves the overall performance of the simplex search; for smaller steps, the simplex would soon be trapped in a (generally poor) local optimum.

For giving an insight of the quality of the simplex method with the proposed setup, we show here the results on some standard benchmark functions (maximisation problems were converted into minimisations).

Problem A: Sphere function. Optimal solution: $f^* = 0$.

$$f_A(x) = \sum_{i=1}^{N} x_i^2, \quad x_i \in [-30, 30], \quad i = 1, \ldots, N$$

Problem B: Ackley function [1] (e is Euler's number). Optimal solution: $f^* = 0$.

$$f_B(x) = -20 \ \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{N} x_i^2}\right) + \exp\left(\frac{1}{N}\sum_{i=1}^{n}\cos(2\pi x_i)\right) - 20 - e,$$

$$x_i \in [-30, 30], \quad i = 1, \ldots, N$$

Problem C: Rosenbrock function[14]. Optimal solution: $f^* = 0$.

$$f_C(x) = \sum_{i=1}^{N-1} \left[100 \ \left(x_i^2 - x_{i+1}\right)^2 - (x_i - 1)^2\right], \quad x_i \in [-5, 10], \quad i = 1, \ldots, N.$$

For these test functions with dimension $N = 10$, we performed an experiment using 100 independent runs of Algorithm 2. The parameters used were: maximum number of evaluations $M = 100000$, convergence criterion $\epsilon = 0$, and initial step $\lambda = 1$. The results obtained are reported in table 1.

As expected, for the Sphere function (which is unimodal), the optimal solution is always found. Concerning the Ackley and the Rosenbrock functions, the method finds the global optimum many times, even though these functions have many local optima. The explanation is that with the current setup, the movements of the initial, very large simplex, are large movements, which can filter small neighbourhood local optima, and move the simplex into "good" search areas, where, in these cases, the global optimum is located. Please see figure 2 for a visual insight.

For all of these benchmarks, the escaping mechanisms described in the next section systematically found the global optimum; hence, they were not included in the difficult benchmark set.

3 Escaping local optima

As the simplex method uses downhill movements, its solution will in general be a local optimum. If we wish to overcome this drawback, and be able to potentially obtain the global optimum of an NLP, we have to provide an escape mechanism.

The strategies that we describe for escaping are based on a restart criterion ϵ, and a stopping criterion M. Both of these are user-supplied values.

Restart will occur if the vertices of the simplex have all evaluations which are feasible (or all infeasible), and the deviation between the objective (resp., the infeasibility) of the best and worst vertices is bellow ϵ.

All of the methods will stop if the number of evaluations has reached the limit M.

Problem	Optimal solution	Best found solution	Average solution	% successful runs
sphere	0	4.94066e-324	9.24891e-321	100
rosenbrock	0	5.62635e-26	0.717584	82
ackley	0	5.32907e-14	2.18798	37

Table 1: Results for the Nelder and Mead's simplex method, with problem sizes $N = 10$, stopping criteria $M = 100000$ and $\epsilon = 0$, and inital step $\lambda = 1$.

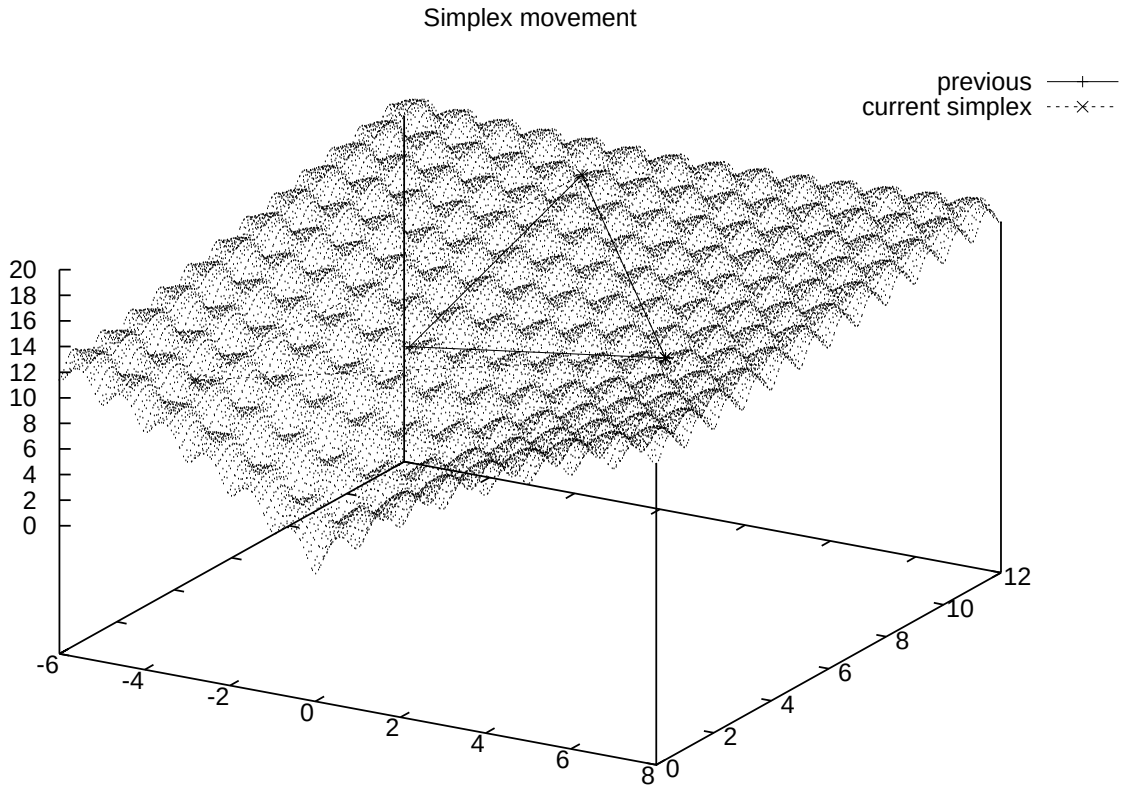

Figure 2: Movements on the large simplex are driven by the general trend of the function, and many narrow local optima are avoided. Example for a 2-dimensional version of the Ackley function.

3.1 Random-start iterated simplex

This method consists of restarting the algorithm from a random solution every time there is convergence of the simplex according to the criterion ϵ, until the maximum number of evaluations M is reached, as presented in Algorithm 3. At each iteration, a random point is drawn and the simplex is reinitialised from that point (with a step λ). Whenever the best found solution is improved, a new local search is performed with a smaller stopping criterion ϵ', for refining this local optimum (line 8 on Algorithm 3).

The algorithm returns the best solution found on all the iterations.

3.2 Directional escape

Another possibility for escaping local optima is the following. When the simplex has converged according to the criterion ϵ, start expanding the simplex through its best vertex (always updating the ranking among the vertices). Expansion will initially decrease the quality of the point; but after a certain number or repetitions, we will reach a local *pessimum*, and the subsequent expansion will lead to an improvement. We propose to expand until the worst point of the simplex has been *improved*. At that point, we expect to be on the other side of the hill; hence, if we restart the simplex algorithm from that point, we expect to reach a different local optimum. We also restart if the bound has been crossed, using the first point outside bounds to initialise the simplex.

Algorithm 3: Iterated simplex: repeated local search starting from random points.
IteratedSimplex(ϵ, ϵ', λ, M)

```
(1)   read problem's data (N, f, ...)
(2)   k = 0
(3)   while k < M
(4)      x = random solution (Equation 2)
(5)      x = SimplexLocalSearch(x, ε, λ, M − k)
(6)      k = current number of function evaluations
(7)      if x* not initialised or x better than x*
(8)         x* = SimplexLocalSearch(x, ε', λ, M − k)
(9)         k = current number of function evaluations
(10)  return x*
```

After an escape point is determined, the simplex is reinitialised around it as describe in section 2, by adding a step independently to each of the coordinates of x^0. We called this strategy *escape*, and detail it in Algorithm 4.

Algorithm 4: Directional escape, based on expansions of the simplex.
Escape(ϵ, ϵ', λ, M)

```
(1)   read problem's data (N, f, ...)
(2)   k = 0
(3)   x = random solution (Equation 2)
(4)   while k < M
(5)      x = SimplexLocalSearch(x, ε, λ, M − k)
(6)      k = current number of function evaluations
(7)      if x* not initialised or x better than x*
(8)         x* = SimplexLocalSearch(x, ε', λ, M − k)
(9)      repeat
(10)        x' = x
(11)        x = γ s'' + (1 − γ) s̄          ←(work on the last simplex)
(12)        s* = x                          ←(work on the last simplex)
(13)     until x better than x' or x is out of bounds
(14)     k = current number of function evaluations
(15)  return x*
```

On this algorithm, after finding a local optimum, the simplex is repeatedly expanded through its best point (lines 9–13), until obtaining an *improvement* in the objective value, and a new local search is started from that point.

This strategy has the nice property of requiring no additional parameters.

3.3 Tabu search

Tabu search for non-linear programming is not a precisely defined concept, as there is not a commonly used notion of *tabu* in this context. Actually, if the tabu concept is related to the kind of movement that is done, the escape mechanism described in the previous section can be considered as a tabu search: after a local optimum is reached, only expansion of the simplex is considered non-tabu.

In this section we propose a different concept: that of tabu based on the region of space being searched (as proposed also, for example, in [5]).

As we will shortly see, for tabu search to work in our setting it will have to be significantly modified, compared to the more usual tabu search in combinatorial optimisation.

3.3.1 Tabu solutions: classification

A trivial extension of the method devised for solution classification described on section 2.1 consists of associating a tabu value to each solution, and then using this value as the primary key for solution sorting.

In this context, for two different solutions x and y, x is said to improve y if and only if:

- x has a smaller tabu value than y;
- both have similar tabu values, and x has a smaller sum of constraint violations than y (i.e., $\delta(x) < \delta(y)$);
- both are feasible and not tabu, and the objective value of x is better than that of y.

3.3.2 Tabu regions

The most straightforward way of implementing a tabu search for continuous, non-linear problems is that of making the region around a local optimum (obtained by the Nelder and Mead algorithm) a tabu region. This way, for the tenure of the tabu status, we are sure that the search will not fall into the same local optimum.

This strategy, however, did not lead to good results, for the benchmarks used in this paper. We have tested many different approaches on this method, all of them with no success. The main reasons for this are related to the size of the tabu region: if it is too narrow, the search tends to find local optima on the border between tabu and non-tabu regions; on the other hand, if the region is too large, good local optima around the current solution are missed. This difficulty in parameterisation, and the observation that search around local optima is frequently essential to find better solutions, lead us to give up true tabu search, and try the opposite strategy: non-tabu search.

3.4 Inversing tabu regions: non-tabu search

As all the strategies that assigned a tabu status to the region of the last found local optima failed, we deduced that this tabu status barred the search from good regions, resulting in poor performance.

It is therefore expectable that for a good performance, the search has to be driven into the areas of previous local optima, instead of avoiding them. The rationale is that good local optima are often close to other local optima; hence, it might make sense to reinforce the search around previous optima, instead of avoiding regions close to them. Of course, the limit of this reasoning occurs when search cannot escape some particular local optimum.

The algorithm that we devised for this, which could be named *non-tabu search* is depicted in Algorithm 5.

In this algorithm, the region around a local optimum is exploited by drawing a random solution on its vicinity, and restarting local search from it (lines 9–18). A parameter σ controls the distance from the current base solution, used to draw new starting solutions. Another parameter, R, controls the number of attempts to do around each base solution.

Algorithm 5: The non-tabu algorithm.

NonTabuSearch(ϵ, ϵ', λ, M, σ, R)

(1) read problem's data $(N, f, \ldots)$
(2) $k = 0$
(3) x = random solution (Equation 2)
(4) x = SimplexLocalSearch(x, ϵ, λ, $M - k$)
(5) k = current number of function evaluations
(6) $x^* = x$
(7) $y = x$
(8) **while** $k < M$
(9) **for** i=1 **to** R
(10) $x_j = y_j \pm \sigma\ (u_j - l_j), \quad j = 1, \ldots, N$
(11) x = SimplexLocalSearch(x, ϵ, λ, $M - k$)
(12) k = current number of function evaluations
(13) **if** x better than x^*
(14) x^* = SimplexLocalSearch(x, ϵ', λ, $M - k$) ←*refine search*
(15) k = current number of function evaluations
(16) **if** x' not initialised **or** x better than x'
(17) $x' = x$
(18) $y = x'$
(19) **return** x^*

After R attempts are made, the base solution moves into the best solution found in theses tentatives.

These two parameters give a way for controlling the search, and to adapt it to the problem being tackled. Good parameters for a particular problem are generally easy to devise; but we could find no parameters that are simultaneously good for all the benchmarks.

4 Computational results

For the evaluation of the strategies that we proposed in this paper, we have relied on a set of multi-modal test functions which include constrained and unconstrained problems. (Benchmarks which were maximisation problems were converted into minimisations; we report the optimal objectives, or our best known objectives if optimality is not proven.)

Problem 1: Griewank's function ($d = 4000$). Optimal solution: $f^* = 0$.

$$f_1(x) = \frac{1}{d}\sum_{i=1}^{N}(x_i - 100)^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i - 100}{\sqrt{i}}\right) + 1, \quad x_i \in [-600, 600] \quad i = 1, \ldots, N$$

Problem 2: Shekel's foxholes ($m = 30$; c_i, $A(i)$ available in [3]). Optimal solution: $f^* = -10.20787\ldots$.

$$f_2(x) = -\sum_{j=1}^{m}\frac{1}{||x - A(j)||^2 + c_j}, \quad x_i \in [0, 10] \quad i = 1, \ldots, N$$

Problem 3: Michalewicz' function ($m = 10$). Optimal solution: $f^* = -9.6601517\ldots$.

$$f_3(x) = -\sum_{i=1}^{N}\sin(x_i) \cdot \sin^{2m}\left(\frac{i \cdot x_i^2}{\pi}\right), \quad x_i \in [0, \pi] \quad i = 1, \ldots, N$$

Problem 4: Langerman's function. ($m = 30$; c_i, $A(i)$ available in [3]). Optimal solution: $f^* = -1.5$.

$$f_4(x) = -\sum_{j=1}^{m} c_j \cdot e^{-\frac{||x-A(j)||^2}{\pi}} \cdot \cos(\pi \cdot ||x - A(j)||^2) \quad x_i \in [0, 10] \quad i = 1, \ldots, N$$

Problem 5: Crescent function ($N = 2$) [7]. Best known solution: $f^* \approx -6961.814$.

$$f_5(x) = (x_1 - 10)^3 + (x_2 - 20)^3, \quad \text{subject to:}$$
$$(x_1 - 5)^2 + (x_2 - 5)^2 \geq 100$$
$$(x_1 - 6)^2 + (x_2 - 5)^2 \leq 82.81$$
$$x_1 \in [13, 100], x_2 \in [0, 100]$$

Problem 6: Luus's function ($N = 3$) [4]. Best known solution: $f^* \approx -11.67664$.

$$f_6(x) = x_1^2 + x_2^2 + x_3^2, \quad \text{subject to:}$$
$$4(x_1 - 0.5)^2 + 2(x_2 - 0.2)^2 + x_3^2 + 0.1x_1x_2 + 0.2x_2x_3 \leq 16$$
$$2x_1^2 + x_2^2 - 2x_3^2 \geq 2$$
$$x_i \in [-2.3, 2.7] \quad i = 1, 2, 3$$

Problem 7: Keane's function. Best known solution: $f^* \approx -0.747303$.

$$f_7(x) = -\frac{\left|\sum_{i=1}^{N} \cos^4(x_i) - 2\prod_{i=1}^{N} \cos^2(x_i)\right|}{\sqrt{(\sum_{i=1}^{N} ix_i^2)}}, \quad \begin{array}{l}\text{subject to:} \\ \prod_{i=1}^{N} x_i \geq 0.75 \\ \sum_{i=1}^{n} x_i \leq 15N/2 \\ x_i \in [0, 10] \quad i = 1, \ldots, N\end{array}$$

Problem 8: Polygon model [6]. (The actual number of variables is $2N$.) Best known solution: $f^* \approx -0.746984$.

$$f_8(x, y) = -\tfrac{1}{2}\sum_{i=1}^{N-1} x_{i+1}x_i \sin(y_{i+1} - y_i),$$
subject to:
$$x_i^2 + x_j^2 - 2x_ix_j \cos(y_i - y_j) \leq 1, \; i = 1, ..., N, j = i, ..., N$$
$$y_i \leq y_{i+1}, \quad i = 1, \ldots, N$$
$$x_i \in [0, 1], \quad y_i \in [0, \pi] \quad i = 1, \ldots, N$$

For the benchmarks which admit choosing the dimension of the problem, we have set N=10. In all the runs a random solution x^0 in the box defined by the problem bounds was used as the first vertex s^0 of the initial simplex. The remaining vertices s^i, $i = 1, \ldots, N$, were obtained adding a step on each coordinate, as described in section 2. In this experiment we have set $\lambda = 1$, implementing that all the points except s^0 will be infeasible, as they will be out of the bounding boxes. As for the case of the standard simplex method, computational experiments have shown that this improves the overall performance of all the tested methods.

For each of the methods, the initial solution is different from run to run; but for a given run, all the methods will start on the same solution. This explains why the performance curves presented below are all identical during the initial steps (until the first restart).

In this experiment the maximum number of function evaluations allotted to each method was $M = 100000$. For all the methods except the pure simplex method, we established a criterion $\epsilon = 10^{-4}$ for stopping the current downhill search. This implies that when the deviation between the objective of the best and the worst point of the simplex is less than that value, all the escape strategies will restart on a different solution. The pure simplex method will continue the search trapped on that local optimum, until reaching M evaluations.

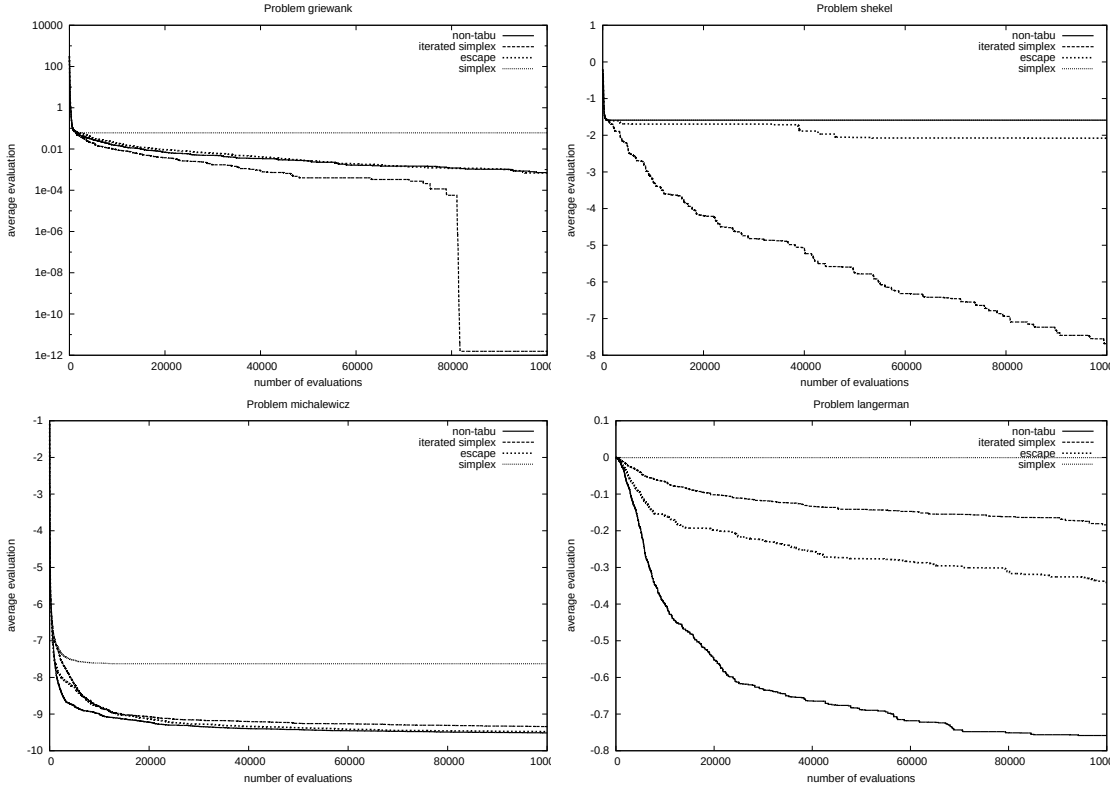

Figure 3: Performance on unconstrained benchmarks.

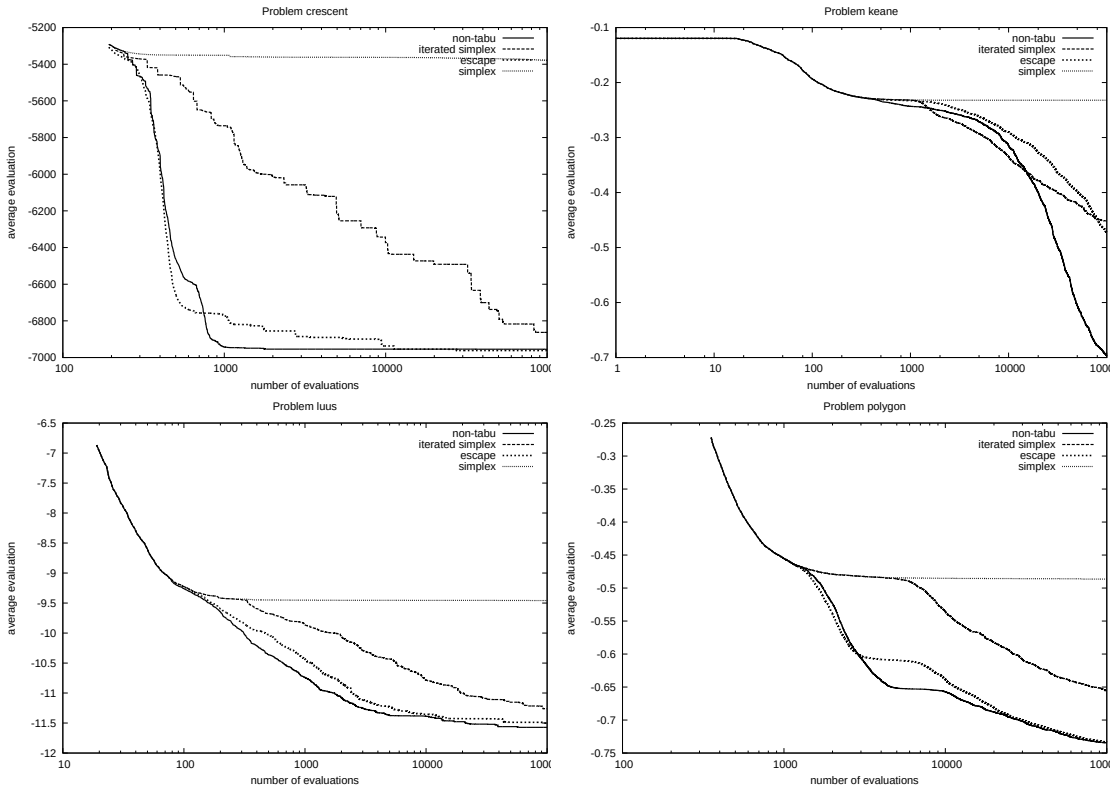

Figure 4: Performance on constrained benchmarks.

Problem	Method	Best found solution		Average solution		Standard	% succ.
		value	dev. from opt.	value	dev. from opt.	deviation	runs
griewank	simplex	5.42101e-20	5.42101e-20	0.0610834	0.0610834	0.0648968	7
	it-simplex	3.40844e-13	3.40844e-13	1.5458e-12	1.5458e-12	1.05875e-12	100
	esc+rand	5.2005e-13	5.2005e-13	0.0111093	0.0111093	0.0091483	23
	esc+reinit	2.99198e-13	2.99198e-13	0.000690256	0.000690256	0.00221823	91
	nontabu	3.55938e-13	3.55938e-13	0.000717737	0.000717737	0.00230486	89
shekel	simplex	-10.2079	0	-1.58797	8.61991	0.889729	1
	it-simplex	-10.2079	0	-7.68605	2.52182	3.46323	65
	esc+rand	-10.2079	0	-3.46104	6.74684	3.60729	22
	esc+reinit	-10.2079	0	-2.08117	8.1267	2.08814	6
	nontabu	-10.2079	0	-1.58797	8.61991	0.889729	1
michalewicz	simplex	-9.38692	0.273235	-7.62653	2.03363	1.06321	0
	it-simplex	-9.58442	0.0757324	-9.34113	0.319019	0.174226	0
	esc+rand	-9.65524	0.00491115	-9.35476	0.305387	0.15857	0
	esc+reinit	-9.66015	-1.00009e-11	-9.48182	0.178334	0.117634	2
	nontabu	-9.66015	-1.00009e-11	-9.51409	0.146057	0.114983	4
langerman	simplex	-0.0174966	1.4825	-0.000398003	1.4996	0.00198529	0
	it-simplex	-0.797694	0.702306	-0.183889	1.31611	0.12337	0
	esc+rand	-0.797694	0.702306	-0.452344	1.04766	0.167528	0
	esc+reinit	-0.797694	0.702306	-0.341029	1.15897	0.236473	0
	nontabu	-1.5	0	-0.758496	0.741504	0.145192	1
keane	simplex	-0.424008	0.323295	-0.232212	0.515091	0.0503369	0
	it-simplex	-0.559488	0.187815	-0.45247	0.294833	0.0436819	0
	esc+rand	-0.677179	0.0701241	-0.529374	0.217929	0.0623664	0
	esc+reinit	-0.739872	0.00743129	-0.473476	0.273827	0.107812	0
	nontabu	-0.747218	8.50598e-05	-0.696889	0.0504137	0.0515507	1
crescent	simplex	-6961.81	0	-5378.59	1583.22	2162.72	61
	it-simplex	-6961.81	6.5e-07	-6862.86	98.9504	669.05	97
	esc+rand	-6961.81	1e-06	-6961.8	0.0119889	0.0210561	69
	esc+reinit	-6961.81	1e-06	-6961.8	0.0112842	0.0184502	67
	nontabu	-6961.81	1.8e-06	-6954.83	6.98607	69.6741	44
luus	simplex	-11.6758	0.000807526	-9.45878	2.21785	1.5797	0
	it-simplex	-11.6766	1.39e-08	-11.261	0.415647	1.1832	29
	esc+rand	-11.6766	1.882e-07	-11.4715	0.205172	0.655578	39
	esc+reinit	-11.6766	2.41e-08	-11.5025	0.174092	0.623979	38
	nontabu	-11.6766	0	-11.572	0.104608	0.392146	59
polygon	simplex	-0.651455	0.0955292	-0.486476	0.260509	0.0906192	0
	it-simplex	-0.713004	0.0339798	-0.655703	0.0912811	0.0259117	0
	esc+rand	-0.735198	0.0117861	-0.706547	0.0404369	0.0170169	0
	esc+reinit	-0.746781	0.000203215	-0.73376	0.0132238	0.0104011	0
	nontabu	-0.745532	0.00145254	-0.734565	0.0124194	0.00549593	0

Table 2: Results for the several escape methods, stopping criteria $M = 100000$ and $\epsilon = 1.e{-}4$, initial step $\lambda = 1$, and refinement stopping criterion $\epsilon' = 1.e - 12$.

We have also used the best and the average solution found on the 100 runs as a performance measure; it is shown in table 2. This table shows that the iterated simplex fails to find truly good results for some of the benchmarks, even when the average solution is acceptable.

For the non-tabu search, the parameters used were $\sigma = 0.1$ and $R = 10$. Notice that for some benchmarks these are not good; in particular, for the Shekel's problem non-tabu behaved approximately as the pure simplex, as it could not get off the region of a local optimum. (Good parameters for this benchmark, however, would hurt the performance on the others.)

A performance measure considered is the value of the best evaluation as a function of the number of evaluations. That value, averaged on 100 independent runs, is plotted on figures 3 and 4. For constrained problems, lines were plotted after *all* the 100 runs obtained feasible solutions (hence, after it was possible to average the objective values).

These graphics give an idea of the quality of each of the methods. They show that there is not a clear winner for all the benchmarks. For the Griewank and the Shekel problems, iterated simplex is the best strategy. In particular for Griewank, this strategy could always find the optimal solution around evaluation 80000; from that point on, there are only refinements. For the other benchmarks, non-tabu is possibly the best method. The escape method is the most balanced strategy.

5 Conclusions

In this paper we presented an extension of the simplex method for non-linear programming which allows its straightforward application to constrained problems.

For avoiding stagnation in local optima, we analysed the behaviour of several escaping mechanisms. The simplest of them is random-start local search. Another one was based on expanding the simplex from the local minimum, going uphill, until the expansion goes downhill again. At that point, we expect to be on the other side of the hill, and restarting simplex descent will likely lead to a different local optimum. The other possibility presented is based on the exploitation of the area of a the previous local optimum, by drawing starting points for local search in its vicinity: we called it non-tabu search.

Computational experiments have shown that all the escaping mechanisms were effective for avoiding stagnation in local optima.

Due to the simplicity of its implementation, random start iterated local search is a highly attractive method. However, for some problems it is not able to find truly good solutions (though the average solution is generally of high quality).

The simplex expansion escaping mechanism is for most of the test cases slightly superior to random start local search, but in general the non-tabu search provides the best results.

Test and improvement of the escape methods for problems with equality constraints, and other possibilities of dealing with these constraints, remain as topics for future research. More research topics are their incorporation in more elaborate strategies, like strategic oscillation or population based methods.

References

[1] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, Boston, 1987.

[2] C. Audet and J. Dennis Jr. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14(4):980–1010, 2004.

[3] H. Bersini, M. Dorigo, L. Gambardella, S. Langerman, and G. Seront. First international contest on evolutionary optimization, 1996. In IEEE International Conference on Evolutionary Computation.

[4] M. F. Cardoso, R. L. Salcedo, and S. F. de Azevedo. The simplex-simulated annealing approach to continuous non-linear optimization. *Computers Chemical Engineering*, 20(9):1065–1080, 1996.

[5] R. Chelouah and P. Siarry. A hybrid method combining continuous tabu search and nelder-mead simplex algorithms for the global optimization of multiminima functions. *European Journal of Operational Research*, 161:636–654, 2005.

[6] E. D. Dolan and J. J. Moré. Benchmarking optimization software with COPS. Technical Report ANL/MCS-246, Argonne National Laboratory, 2001.

[7] C. A. Floudas and P. M. Pardalos. *Recent Advances in Global Optimization*. Princeton University Press, 1992.

[8] F. Glover, M. Laguna, and R. Martí. Scatter search. Technical report, Graduate School of Business and Administration, University of Colorado, 2000.

[9] R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methods: then and now. *Journal of Computational and Applied Mathematics*, 124(1-2):191–207, 2000.

[10] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[11] J. P. Pedroso. Meta-heuristics using the simplex algorithm for nonlinear programming. In *Proceedings of the 2001 International Symposium on Nonlinear Theory and its Applications*, pages 315–318, Miyagi, Japan, 2001.

[12] J. P. Pedroso. Extensions of the nelder and mead simplex algorithm: an implementation in the C++ programming language. Internet repository, version 0.1, 2005. http://www.ncc.up.pt/˜jpp/nlp-simplex.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipies in C: the Art of Scientific Computing*. Cambridge University Press, second edition, 1997.

[14] H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.