A heuristic for nonlinear global optimization

M. Bierlaire[*] M. Thémans[†] N. Zufferey[‡]

April 30, 2007

Report TRANSP-OR-070430
TRANSP-OR Transport And Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
Submitted for possible publication to
Operations Research.

[*]Ecole Polytechnique Fédérale de Lausanne (EPFL), TRANSP-OR Transport and Mobility Laboratory, CH-1015 Lausanne, Switzerland. Email: michel.bierlaire@epfl.ch
[†]Ecole Polytechnique Fédérale de Lausanne (EPFL), TRANSP-OR Transport and Mobility Laboratory, CH-1015 Lausanne, Switzerland. Email: michael.themans@epfl.ch
[‡]Université Laval, Facultés des sciences de l'administration, Département opérations et systèmes de décision, Québec, Canada.Email: Nicolas.Zufferey@osd.ulaval.ca

Abstract

We propose a new heuristic for nonlinear global optimization combining a variable neighborhood search framework with a modified trust-region algorithm as local search. The proposed method presents the capability to prematurely interrupt the local search if the iterates are converging to a local minimum which has already been visited or if they are reaching an area where no significant improvement can be expected. The neighborhoods as well as the neighbors selection procedure are exploiting the curvature of the objective function. Numerical tests are performed on a set of unconstrained nonlinear problems from the literature. Results illustrate that the new method significantly outperforms existing heuristics from the literature in terms of success rate, CPU time, and number of function evaluations.

1 Motivation and literature review

We are interested in the identification of a global minimum of the nonlinear optimization problem defined by

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable. No special structure is assumed on f. The vast literature on nonlinear optimization (Bertsekas, 1999, Nocedal and Wright, 1999, Conn et al., 2000, Bierlaire, 2006 to cite a few) focusses on the global convergence of algorithms towards a local optimum, with a fast local convergence. Note that "local" and "global" are used in two different ways in this literature. A point x^* is a global minimum of f if $f(x^*) \leq f(x)$, for all $x \in \mathbb{R}^n$. It is a local minimum of f if there exists $\varepsilon > 0$ such that $f(x^*) \leq f(x)$ for each x such that $\|x - x^*\| \leq \varepsilon$. An algorithm is said to be globally convergent if it converges to a (local) minimum from any starting point. It is locally convergent when it is converging to a (local) minimum when the starting point x_1 is in a neighborhood of x^*. We refer the reader to the nonlinear optimization literature for more details. In this paper, we are interested in a heuristic which is (hopefully, not provably) globally convergent towards a global minimum.

Nowadays, there exist efficient and robust methods and softwares to solve unconstrained nonlinear optimization problems (in the sense of identifying a local minimum). In the case that the problem presents several local minima, the convergence to a global minimum cannot be ensured. There are several practical applications where a global minimum of a nonlinear function is required. Among them, we can cite two important categories: the maximum likelihood estimation of econometric models, and the computation of equilibrium of energy functions in mechanical and chemical engineering.

In econometrics, more and more nonlinear models are being developed to address the complexity of real phenomena under analysis. For instance, discrete choice models are mathematical models used to analyze and predict the behavior of individuals when faced to choice situations (see, for instance, Ben-Akiva and Lerman, 1985, and Ben-Akiva and Bierlaire, 2003 for a deep review of these models). The maximum likelihood estimation of those models involves the maximization of nonlinear non-concave functions, exhibiting several local maxima.

The issue of finding a global optimum in the context of econometric models estimation, and in particular discrete choice models, has interestingly almost not been addressed in the literature. The standard simulated annealing heuristic is a widely used algorithm in this context (see Goffe et al., 1992, Goffe et al., 1994). Dorsey and Mayer (1995) have also proposed genetic algorithms.

Several problems in global optimization arise from chemical and mechanical engineering (see, for instance, Floudas et al., 1999). Lin and Stadtherr (2004) have proposed deterministic approaches for global optimization in the context of parameter estimation of models from computational chemistry and molecular modeling. The identification of a global optimum of continuous functions is also needed in phase equilibrium calculations via Gibbs free energy minimization (Teh and Rangaiah, 2003) as well as in protein structure prediction (Klepeis et al., 2003). In the context of mechanical engineering, global optimization problems occur in the robust design of structures (see, for instance, Sandgren and Cameron, 2002).

The literature on nonlinear global optimization can be divided in two parts: deterministic and exact approaches on the one hand, and heuristics and meta-heuristics on the other hand.

The most important deterministic approaches are (i) methods based on real algebraic geometry (see Lasserre, 2001, Henrion and Lasserre, 2003 and Lasserre, 2004), (ii) exact algorithms as the adaptation of Brand and Bound proposed by Androulakis et al. (1995), and (iii) interval analysis (see Hansen and Walster, 2003 for a review of these methods). Some other approaches exploit the structure of f and the fact that a nonconvex function can be described by the difference of convex functions. The DC (difference of convex functions) programming and related DCA algorithms have been applied successfully to global optimization of nonconvex functions (Horst and Thoai, 1999, and Le and Pham, 2005).

The use of heuristics to address in practice the difficult problem of global nonlinear optimization has been intensive for several decades and is still relevant (see, for instance, Hedar and Fukushima, 2004, Addis and Leyffer, 2004, Addis et al., 2005, Hedar and Fukushima, 2006, and Mladenovic et al., 2006). Many approaches consist in hybridizing derivative-free methods with heuristics originally designed for discrete optimization problems. For example, Hedar and Fukushima (2002) have developed an hybrid simulated annealing method by combining a Nelder-Mead algorithm with simulated annealing.

The same authors have proposed another simulated annealing algorithm using other derivative-free methods like the approximate descent direction (ADD) and pattern search (see Hedar and Fukushima, 2004) as well as a new tabu search method which is "directed" by direct search methods in Hedar and Fukushima (2006). Franzè and Speciale (2001) have adapted a tabu search method using a pattern search algorithm. Tabu search has also been combined with both scatter search and direct search (see Glover, 1994). Recently, Mladenovic et al. (2006) have proposed an adaptation of the variable neighborhood search (VNS) heuristic for unconstrained nonlinear optimization problems using random distributions to compute neighbors. Vaz and Vicente (to appear) have developed a hybrid algorithm that combines a particle swarm heuristic with a pattern search method.

Continuous adaptations of classical heuristics in discrete optimization are also proposed for nonlinear global optimization. The simulated annealing algorithm has been widely adapted for the continuous case (see, for example, Chelouah and Siarry, 1997, Locatelli, 2000). New algorithms based on tabu search (see Chelouah and Siarry, 2000b, Battiti and Tecchiolli, 1996) and genetic algorithms (see Chelouah and Siarry, 2000a, and Chelouah and Siarry, 2003) have also been derived.

Clearly, most of the heuristics designed for nonlinear global optimization are inspired from the discrete optimization literature. It is interesting to note that these papers are using either simple local searches (random search for instance) or derivative-free methods such as direct search strategies like Nelder-Mead. These algorithms are not making use of first and second order derivatives. We have found only a few algorithms using more advanced local searches. Renders and Flasse (1996) have hybridized genetic algorithms with a quasi-Newton method, while Dekkers and Aarts (1991) made an hybridation of simulated annealing with both steepest descent and quasi-Newton methods as local search. Recently, Mladenovic et al. (2006) have proposed a variable neighborhood search framework for nonlinear global optimization, within which efficient algorithms from nonlinear optimization are used. It is one of the rare approaches where an efficient local algorithm for nonlinear optimization is adapted to global optimization.

The heuristic proposed in this paper is directly inspired by state-of-the-art algorithms for nonlinear optimization, and state-of-the-art heuristics in discrete optimization.

Among the wide variety of Newton-like methods proposed in the literature of nonlinear optimization for solving (1), we are particularly interested in quasi-Newton methods which use only the gradient of the objective function f to be supplied and construct an approximate second-order model of f (see Nocedal and Wright, 1999). By measuring changes in the gradient ∇f at previous iterates, they perform a secant approximation of the Hessian matrix $\nabla^2 f$ and exhibit a fast local converge rate (typically superlinear). As the second derivatives are not required, they represent a good tradeoff between fast convergence and low computational burden.

In addition to the use of approximated derivatives, we want to ensure convergence from remote starting points in order to get a practical method for finding local minima of (1). Global convergence can be enforced using specific techniques. On the one hand, linesearch strategies control, at each iteration of the optimization algorithm, the step length taken in the Newtonian search direction (see, for example, Nocedal and Wright, 1999). On the other hand, trust-region methods approximately compute the minimum of a quadratic model, centered at the current iterate x_k, in an appropriate neighborhood of x_k called the trust-region (see Conn et al., 2000). More recently, filter-trust-region methods have been proposed by Fletcher and Leyffer (2002), as an extension of the trust-region framework.

In this paper, we adopt a trust-region algorithm using a quasi-Newton framework for constructing the quadratic model of the objective function.

In discrete optimization, local search heuristics operate in a *search space* S, also called a *solution space*. The elements of this space are called *solutions*. For every solution $s \in S$, a neighborhood $\mathcal{N}(s) \subset S$ is defined. A local search method starts at an initial solution, and then moves repeatedly from the current solution to a neighbor solution in order to try to find better solutions, measured by an appropriate objective function. The most popular local search methods are the following: simulated annealing (see Kirkpatrick et al., 1983), tabu search, which was originally proposed by Glover (1986) and Hansen (1986), and variable neighborhood search (see Mladenovic and Hansen, 1997). A more recent local search heuristic is the *variable space search* algorithm proposed by Hertz et al. (2006), which uses not only several neighborhoods, but also several objective functions and several search spaces.

Evolutionary heuristics encompass various algorithms such as genetic algorithms (Davis, 1991), scatter search (Glover, 1998), ant systems (Dorigo and Blum, 2005) and adaptive memory algorithms (Rochat and Taillard, 1995). They can be defined as iterative procedures that use a central memory where information is collected during the search process. Each iteration is made of two complementary phases which modify the central memory. In the cooperation phase, a *recombination operator* is used to create new offspring solutions, while in the self-adaptation phase, the new offspring solutions are modified individually. The output solutions of the self-adaptation phase are used for updating the content of the central memory. The most successful evolutionary heuristics are hybrid algorithms in the sense that a local search technique (tabu search for example) is used during the self-adaptation phase.

In this paper, we propose a new heuristic inspired by the Variable Neighborhood Search (VNS) framework originally proposed by Mladenovic and Hansen (1997), which is one of the most recent and efficient heuristics for discrete optimization. The choice of this heuristic is motivated by its ability to widely explore the solution space ($\mathbb{R}^n$ in our case) thanks to the use of several types of neighborhoods. VNS proceeds, for example, by exploring increasingly distant neighborhoods from the current solution. It is also simple to adapt and implement as it only requires two elements: a list of neighborhoods and a local search algorithm. Moreover, a direct and simple adaptation of the VNS algorithm of Mladenovic and Hansen (1997) has recently been proposed for unconstrained continous global optimization by Mladenovic et al. (2006) and has shown very encouraging results compared to other existing approaches and heuristics.

Our method combines a VNS framework with a trust-region algorithm. The philosophy of our approach is to diversify the set of iterates in order to increase the probability of finding a global minimum of (1) and to prematurely interrupt the local search if it is converging to a local minimum which has already been visited or if the iterates are reaching an area where no significant improvement can be expected.

The paper is organized as follows. In Section 2, we present our VNS algorithm. Intensive numerical experiments have been conducted, and the results are presented in Section 3. Finally, we conclude and give some perspectives for future research in Section 4.

2 Algorithm

The local search procedure plays a significant role in our algorithm. In particular, we propose a framework where the local search procedure has the ability to prematurely interrupt its iterations, in order to save computational efforts.

In the following, we refer to the local search procedure as

$$(\text{SUCCESS}, y^*) \leftarrow \text{LS}(y_1, \ell_{\max}, \mathcal{L}), \tag{2}$$

where y_1 is the starting point of the local search, $\ell_{\max}$ is the maximum number of iterations, $\mathcal{L} = (x_1^*, x_2^*, \ldots)$ is a list of already identified local minima. If this set is non empty, the local search may be prematurely interrupted if it is likely to converge to an already identified local minimum . If the set is empty, the local search converges to a local minimum, except in the presence of numerical problems. SUCCESS is a boolean variable which is true if the procedure has converged to the local minimum y^*, and false if the method has failed to converge, or has been prematurely interrupted. If SUCCESS = false, y^* is irrelevant. The local search procedure is globally convergent, so that failure to converge may be due only to severe numerical problems, or to a small value of $\ell_{\max}$. We describe our specific local search procedure in Sections 2.1 and 2.2.

A VNS heuristic requires also a procedure to define neighbors of a current iterate. We adopt the conventional structure of nested neighborhoods $\mathcal{N}_k(x)$, $k = 1, \ldots, n_{\max}$, where $\mathcal{N}_k(x) \subset \mathcal{N}_{k+1}(x) \subseteq \mathbb{R}^n$, for each k, and $n_{\max}$ is typically of the order of 5. For each k, we use a procedure providing a list of p neighbors of x (typically, $p = 5$) within $\mathcal{N}_k(x)$, that we denote by

$$(z_1, z_2, \ldots, z_p) = \text{NEIGHBORS}(x, k). \tag{3}$$

We describe two specific neighbors generation procedures in Section 2.3.

The VNS framework we are proposing can be described as follows.

Initialization The algorithm must be started from x_1^*, which is a local mimimum of f. We propose two different ways to obtain x_1^*: a *cold start* and a *warm start* procedures. A cold start happens when the user provides a local optimum, or when the local search procedure is run once until convergence. In this case, the set of visited local minima $\mathcal{L}$ is initialized as $\mathcal{L} = \{x_1^*\}$. The warm start procedure, which has shown to be useful in practice, proceeds as follows.

1. Initialize the set of local minima as $\mathcal{L} = \emptyset$.

2. Generates randomly m points y_j, $j = 1, \ldots, m$.
3. Apply m times the local search procedure, that is

$$(\text{SUCCESS}_j, y_j^*) \leftarrow \text{LS}(y_j, \ell_{\text{small}}, \emptyset), \tag{4}$$

where typical values of the parameters used in our experiments are $m = 5$ and $\ell_{\text{small}} = 20$.
4. If SUCCESS_j is true, then a local minimum has been identified, and $\mathcal{L} = \mathcal{L} \cup \{y_j^*\}$.
5. Select $y_1 = \text{argmin}_{j=1,\ldots,m} f(y_j^*)$, the best point generated by the above procedure.
6. Apply the local search procedure from y_1, that is

$$(\text{SUCCESS}, x_1^*) \leftarrow \text{LS}(y_1, \ell_{\text{large}}, \emptyset), \tag{5}$$

where typical values of ℓ_{large} range from 200 to 1000, depending on the size n of the problem. If y_1 is a local minimum, (5) is not applied. Also, we update

$$\mathcal{L} = \mathcal{L} \cup \{x_1^*\}. \tag{6}$$

If this last local search fails (that is, SUCCESS is false), we declare the heuristic to fail, and stop.

The best iterate found is denoted by x^1_{best} and is initialized by x_1^*.

The iteration counter k is initialized to 1.

Stopping criteria The iterations are interrupted when one of the following criteria is verified.

1. The last neighborhood was unsuccessfully investigated, that is $k > n_{\max}$.

2. The CPU time exceeds a given threshold $t_{\max}$, typically 30 minutes (18K seconds).
3. The number of function evaluations exceeds a given threshold $\text{eval}_{\max}$, typically 10^5.

Main loop For each VNS phase, we apply the following steps.

1. Generate neighbors of x^k_{best}:
$$(z_1, z_2, \ldots, z_p) = \text{NEIGHBORS}(x^k_{\text{best}}, k). \tag{7}$$
2. The local search procedure is applied p times, starting from each generated neighbors, that is, for $j = 1, \ldots, p$,
$$(\text{SUCCESS}_j, y^*_j) \leftarrow \text{LS}(z_j, \ell_{\text{large}}, \mathcal{L}). \tag{8}$$
3. If all local search procedures have been interrupted, that is if $\text{SUCCESS}_j = \text{false}$, for $j = 1, \ldots, p$, we have two variants:

   Economical We set $k = k + 1$ and proceed to the next VNS phase.

   Conservative We apply the local search to convergence from the best point identified by the procedure, that is
$$(\text{SUCCESS}, y^*) \leftarrow \text{LS}(z^*, \ell_{\text{large}}, \emptyset), \tag{9}$$
   where z^* is such that $f(z^*) \leq f(y^*_j)$, $j = 1, \ldots, p$. If $\text{SUCCESS} = \text{true}$, we update the set of local optima: $\mathcal{L} = \mathcal{L} \cup \{y^*\}$.
4. Otherwise, we update the list of local minima, that is for each j such that $\text{SUCCESS}_j = \text{true}$,
$$\mathcal{L} = \mathcal{L} \cup \{y^*_j\}. \tag{10}$$

5. We define x_{best}^{k+1} as the best point in $\mathcal{L}$, that is $x_{\text{best}}^{k+1} \in \mathcal{L}$ and
$$f(x_{\text{best}}^{k+1}) \leq f(x), \text{ for each } x \in \mathcal{L}. \tag{11}$$
6. If $x_{\text{best}}^{k+1} = x_{\text{best}}^{k}$, then we could not improve the best solution during this VNS phase. We must investigate the next neighborhood. We set $k = k + 1$ and we proceed to the next VNS phase.
7. Otherwise, we have found a new candidate for the global optimum. The neighborhood structure is reset, that is, $x_{\text{best}}^{1} = x_{\text{best}}^{k+1}$, $k = 1$ and we proceed to the next VNS phase.

Output The output is the best solution found during the algorithm, that is x_{best}^{k}.

2.1 Local search

We now describe our local search procedure (2). It is based on a trust region framework (see Conn et al., 2000).

A trust-region algorithm is an iterative numerical procedure in which the objective function f is approximated in a suitable neighborhood of the current iterate (we call it the trust-region). More formally, it can be described as follows.

Initialization Initialize the radius of the trust region[1] Δ_1, the iteration counter $\ell = 1$ and $H_1 = I$, the identity matrix.

Model definition Define a quadratic model of f around y_ℓ, that is, for $s \in \mathbb{R}^n$,
$$m_\ell(y_\ell + s) = f(y_\ell) + \nabla f(y_\ell)^T s + \frac{1}{2} s^T H_\ell s. \tag{12}$$

[1]In our tests, we have used the procedure proposed by Sartenaer (1997), but any arbitrary, strictly positive, value can be used.

Step computation. Compute a tentative step s_ℓ within the trust region, that sufficiently reduces the model m_ℓ. This is obtained by solving (not necessarily to optimality) the so-called *trust-region subproblem*:

$$\begin{cases} \min_s m_\ell(y_\ell + s) \\ \text{s.t. } \|s\|_2 \leq \Delta_\ell. \end{cases} \tag{13}$$

We use the truncated conjugate gradient algorithm described by Conn et al. (2000, chap. 7) (see also Toint, 1981 and Steihaug, 1983).

Acceptance of the tentative step Compute $f(y_\ell + s_\ell)$ and define

$$\rho_\ell = \frac{f(y_\ell) - f(y_\ell + s_\ell)}{m_\ell(y_\ell) - m_\ell(y_\ell + s_\ell)}. \tag{14}$$

If $\rho_\ell \geq 0.1$, the tentative step is accepted, and we define $y_{\ell+1} = x_\ell + s_\ell$; otherwise it is rejected and $y_{\ell+1} = y_\ell$.

Trust-region radius update. Set

$$\Delta_{k+1} = \begin{cases} \max(2\|s_\ell\|_2, \Delta_\ell) & \text{if } \rho_\ell \geq 0.9, \\ \Delta_\ell & \text{if } \rho_\ell \in [0.1, 0.9), \\ 0.5\|s_\ell\|_2 & \text{if } \rho_\ell \in [0, 0.1). \end{cases}$$

If it happens that $\rho_\ell < 0$, the adequation between the model and the objective function is so poor that we consider it as a special case. In this case, we use the technique described in Conn et al. (2000, chap. 17).

Update the hessian approximation using the symmetric rank one (SR1) formula when the tentative step is accepted:

$$H_\ell = H_{\ell-1} + \frac{(g_{\ell-1} - H_{\ell-1}d_{\ell-1})(g_{\ell-1} - H_{\ell-1}d_{\ell-1})^T}{(g_{\ell-1} - H_{\ell-1}d_{\ell-1})^T d_{\ell-1}} \tag{15}$$

where $d_{\ell-1} = y_\ell - y_{\ell-1}$ and $g_{\ell-1} = \nabla f(y_\ell) - \nabla f(y_{\ell-1})$.

Note that H_ℓ is not necessarily positive definite when using SR1.

Stopping criteria The algorithm is interrupted in one of the below cases.

- If $\ell \geq \ell_{\max}$, the maximum number of iterations is reached. Set SUCCESS = false, $y^* = y_\ell$ and STOP.
- If $\|\nabla f(y_\ell)\| \leq 10^{-6}$, the local search has converged to a local minimum up to the desired precision, set SUCCESS = true, $y^* = y_\ell$ and STOP.
- If one of the tests described in Section 2.2 is verified, then it is preferable to prematurely interrupt the iterations. We set SUCCESS = false, $y^* = y_\ell$ and STOP.

2.2 Identification of unpromising convergence

A key feature of our approach is to spare computational time by prematurely interrupting the local search iterations if they do not look promising. Note that these tests are applied only if the set $\mathcal{L}$ in (2) is not empty.

In order to do so, we combine three criteria. First, we check if the algorithm does not get closer and closer to an already identified local minimum. Second, we check that the gradient norm is not too small when the value of the objective function is far from the value at the best iterate in $\mathcal{L}$. Third, we check if a significant reduction in the objective function is achieved.

More formally, we prematurely interrupt the local search iterations if one of the following conditions is verified when a tentative step is accepted.

- $\exists x \in \mathcal{L}$ such that $\|y_\ell - x\| \leq 1$,
- $\|\nabla f(y_\ell)\| \leq 10^{-3}$ and $f(y_\ell) - f_{\text{best}} \geq 3$, where f_{best} is the value of the objective function at the best iterate in $\mathcal{L}$,
- $f(y_\ell) > f(y_{\ell-1}) + 0.3 \nabla f(y_{\ell-1})^T s_{\ell-1}$ and $f(y_\ell) - f_{\text{best}} \geq 3$. This Armijo-like condition is supposed to be more demanding than the sufficient

reduction condition, based on (14).

Note that the threshold values presented above have been empirically selected based on various tests of the algorithm.

2.3 Generating neighborhoods

We present now the neighbors generating procedure (3). The key idea is to analyze the curvature of f at x through an analysis of the eigenstructure of H, the approximation of the second derivatives matrix of f at x.

Let $v_1, \dots, v_n$ be the (normalized) eigenvectors of H, and $\lambda_1, \dots, \lambda_n$ the corresponding eigenvalues. We compute them using a standard QR procedure (see, for instance, Golub and Loan, 1996).

Neighbors are generated in direction $w_1, \dots, w_{2n}$, where $w_i = v_i$ if $i \leq n$, and $w_i = -v_i$ otherwise. The size of the neighborhood is defined as a function of k as follows. If $k = 1$, then $d_k = d^{\text{INIT}}$. If $k > 1$, then $d_k = \gamma d_{k-1}$. We have adopted $d^{\text{INIT}} = 1$ and $\gamma = 1.5$ after various numerical tests. The p neighbors generated by this procedure are of the type

$$z_j = x + \alpha d_k w_i \tag{16}$$

where $j = 1, \dots, p$, α is randomly drawn using a uniform distribution between 0.75 and 1, and i is the index of a selected direction.

The indices i for the neighbor generation process are selected according to a sequence of random draws among the $2n$ possible values. We immediately note that the same direction can be selected more than once. In this case, the randomness of α practically guarantees that different neighbors are generated.

The idea is to assign more probability to directions such that the curvature of the function is larger. Indeed, it is hoped that moving in a direction of high curvature increases the chance to jump towards another valley. Taking directions associated with small curvature might cause the iterates to be stucked in a valley where significant improvement cannot be achieved. More formally, the probability for w_i to be selected is given by

$$P(w_i) = P(-w_i) = \frac{e^{\beta \frac{\lambda_i}{d_k}}}{2 \sum_{j=1}^{n} e^{\beta \frac{\lambda_j}{d_k}}}. \tag{17}$$

The probability distribution depends on β which can be viewed as a weight factor associated with the curvature. A value of $\beta = 0$ corresponds to a uniform distribution, where the curvature is actually ignored. A high value of β affects all the mass to the two directions with highest curvature, and zero probability elsewhere. In our tests, we have selected a value of $\beta = 0.05$. Note that the curvature of the function is a local information, which may not be relevant for large neighborhoods. The role of d_k in (17) is to decrease the impact of the curvature and to converge towards a uniform distribution as the size of the neighborhoods grows.

3 Numerical experiments

We have performed intensive numerical tests on a set of problems from the literature (see, for instance, Hedar and Fukushima, 2002, and Chelouah and Siarry, 2003). More precisely, we have used a total of 25 optimization problems corresponding to 15 different test functions described in Appendix A. The functions we use to challenge our algorithm exhibit very different and specific shapes. Most of the functions present several local minima. Some of them have many crowded local minima such as Shubert (SH) and Rastrigin (RT) functions. Easom (ES) function has its global minimum lying in a very narrow hole while the well-known Rosenbrock (R_n) presents a narrow valley. Smooth and more standard functions like De Joung (DJ) function or Zakharov (Z_n) function have also been used.

Note that for each test problem, a search space is provided in which initial points can be randomly selected.

3.1 Performance analysis

All variants of our algorithm and test functions have been implemented with the package Octave (see `www.octave.org` or Eaton, 1997) and computations have been done on a desktop equipped with 3GHz CPU, in double precision.

For each test problem, 100 runs have been performed with our VNS algorithm, except for larger size instances ($n \geq 50$) when only 20 trials have been made. A run is considered to be successful if the VNS algorithm finds a global minimum of the problem. The *warm start* procedure described in Section 2 has been used for all reported tests of our VNS. The random starting points involved in this procedure have been randomly selected in the search domain associated with the problem (see Appendix A).

We consider two measures of performance: the average percentage of success and the average number of function evaluations across successful runs. Performances of competiting algorithms have been obtained from the literature.

For the sake of fair comparison with competitors, the gradient of the objective function used in the local search is computed by using finite differences, requiring n additional function evaluations. Consequently, a single iteration of our local search algorithm requires $n + 1$ function evaluations. In practice, analytical gradients should be used to improve the efficiency of our algorithm.

We are presenting some results with the method proposed by Dolan and More (2002). If $f_{p,a}$ is the performance index (the average number of function evaluations across successful runs) of algorithm a on problem p, then the *performance ratio* is defined by

$$r_{p,a} = \frac{f_{p,a}}{\min_b\{f_{p,b}\}}. \tag{18}$$

For any given threshold π, the overall performance of algorithm a is given by

$$\rho_a(\pi) = \frac{1}{n_p}\Phi_a(\pi), \tag{19}$$

where n_p is the number of problems considered, and $\Phi_a(\pi)$ is the number of problems for which $r_{p,a} \leq \pi$. In particular, the value $\rho_a(1)$ gives the proportion of times that algorithm a wins over all other algorithms.

Note that the sum of $\rho_a(1)$ values for all algorithms a considered in a given profile may exceed 1 in the case that some algorithms performs exactly the same on some of the tested problems. Note also that $\rho_a(\pi)$ goes to 1 one as π grows. Methods such that $\rho_a(\pi)$ converges fast to 1 are considered more efficient.

3.2 Variants and competitors

We consider three variants of our algorithm described in Section 2:

1. the main method is called *VNS*. It uses the *economical* variant described in Section 2 and consequently never applies the local search without potentially interrupting it prematurely. Also, the probabilistic formula (17) is used with $\beta = 0.05$.

2. the conservative method is called VNS_a. It uses the *conservative* variant described in Section 2 in which the local search is applied without premature stop if all local searches have been stopped.

3. the third variant is called VNS_b. It sets $\beta = 0$ in (17), that is, it uses equal probabilities for the neighbors generation procedure.

We compare our method with the following methods from the literature:

1. Direct Search Simulated Annealing (DSSA), see Hedar and Fukushima, 2002.
2. Continuous Hybrid Algorithm (CHA), see Chelouah and Siarry, 2003.
3. Simulated Annealing Heuristic Pattern Search (SAHPS), see Hedar and Fukushima, 2004.
4. Directed Tabu Search (DTS), see Hedar and Fukushima, 2006.
5. General Variable Neighborhood Search (GVNS), see Mladenovic et al., 2006. Note that the authors report the number of function evaluations necessary to find the global minimum (the first one of them if several have been identified during the optimization process) and not the number of function evaluations performed before the algorithm stops. The measure of performance is thus slightly different.

The respective features of these algorithms have been described in Section 1. Note that we have chosen the DTS algorithm with Adaptative Pattern Search (APS), called $\mathrm{DTS}_{\mathrm{APS}}$, which was the best of two variants proposed by Hedar and Fukushima (2006).

3.3 Tests

In the tables presented in this section, some of the cells, corresponding to competitors, are empty when the information was not reported in the paper.

Table 1 gives the number of successes over the 100 runs for 25 problems. Note that we do not report results for variants of our algorithm, VNS_a and VNS_b, as the results are very similar.

Table 2 gives the average number of function evaluations for successful runs on the same 25 problems. In this table, results for all 3 variants of our VNS are included. The comparison between VNS and GVNS on their 10 common problems is available in a specific table (see Table 4) as the measure of performance is slightly different. While Tables 2 and 4 present absolute values for the average number of function evaluations, Tables 3 and 5 give the corresponding normalized values with respect to our *VNS* algorithm.

Finally, Table 6 provides a few results available in terms of CPU time for DTS heuristic on large size problems. CPU time comparison is always complicated. As DTS has been published in 2006, we believe that it illustrates well the good performance of our algorithm.

3.3.1 Comparison of VNS with competitors except GVNS

We first focus on the five first columns of Tables 1 and 2 for problems of small size (skipping the three last rows).

From Table 1, we can see that VNS is the most robust algorithm as it achieves a maximal success rate of 100% on almost all problems, actually 18 out of 20. The best challenger of VNS with regard to robustness is CHA as it is able to solve on each run 12 problems among 16. For instance, VNS in the only algorithm able to reach 100% of success on the Shekel $S_{4,n}$ functions.

Table 2 shows that VNS presents the lowest average number of function evaluations on the majority of the tested problems. One can also see that the efficiency of VNS on Rosenbrock (R_n) and Zakharov (Z_n) functions is

Problem	VNS	CHA	DSSA	DTS	SAHPS	GVNS
RC	100	100	100	100	100	100
ES	100	100	93	82	96	
RT	84	100	100		100	
SH	78	100	94	92	86	100
R_2	100	100	100	100	100	100
Z_2	100	100	100	100	100	
DJ	100	100	100	100	100	
$H_{3,4}$	100	100	100	100	95	100
$S_{4,5}$	100	85	81	75	48	100
$S_{4,7}$	100	85	84	65	57	
$S_{4,10}$	100	85	77	52	48	100
R_5	100	100	100	85	91	
Z_5	100	100	100	100	100	
$H_{6,4}$	100	100	92	83	72	100
R_{10}	100	83	100	85	87	100
Z_{10}	100	100	100	100	100	
HM	100		100			
GR_6	100		90			
GR_{10}	100					100
CV	100		100			
DX	100		100			
MG	100					100
R_{50}	100	79		100		
Z_{50}	100	100		0		
R_{100}	100	72		0		

Table 1: Percentage of success

Problem	VNS	CHA	DSSA	DTS	SAHPS	VNS_a	VNS_b
RC	153	295	118	212	318	179	165
ES	167	952	1442	223	432	249	237
RT	246	132	252		346	340	234
SH	366	345	457	274	450	630	424
DJ	104	371	273	446	398	104	104
$H_{3,4}$	249	492	572	438	517	292	268
$H_{6,4}$	735	930	1737	1787	997	1036	759
$S_{4,5}$	583	698	993	819	1073	769	589
$S_{4,7}$	596	620	932	812	1059	752	591
$S_{4,10}$	590	635	992	828	1035	898	664
R_2	556	459	306	254	357	847	618
Z_2	251	215	186	201	276	273	280
R_5	1120	3290	2685	1684	1104	2197	1157
Z_5	837	950	914	1003	716	866	831
R_{10}	2363	14563	16785	9037	4603	4503	2358
Z_{10}	1705	4291	12501	4032	2284	1842	1754
HM	335		225			388	359
GR_6	807		1830			1011	831
CV	854		1592			1346	782
DX	2148		6941			3057	2243
R_{50}	11934	55356		510505			
Z_{50}	17932	75520		177125^*			
R_{100}	30165	124302		3202879			

Superscript * means that DTS only obtains points close to the global minimum

Table 2: Average number of function evaluations

Problem	VNS	CHA	DSSA	DTS	SAHPS	VNS_a	VNS_b
RC	1	1.93	0.77	1.39	2.08	1.17	1.08
ES	1	5.70	8.63	1.34	2.59	1.49	1.42
RT	1	0.54	1.02		1.41	1.38	0.95
SH	1	0.94	1.25	0.75	1.23	1.72	1.16
DJ	1	3.57	2.62	4.29	3.83	1	1
$H_{3,4}$	1	1.98	2.30	1.76	2.08	1.17	1.08
$H_{6,4}$	1	1.27	2.36	2.43	1.36	1.41	1.03
$S_{4,5}$	1	1.20	1.70	1.40	1.84	1.32	1.01
$S_{4,7}$	1	1.04	1.56	1.36	1.78	1.26	0.99
$S_{4,10}$	1	1.08	1.68	1.4	1.75	1.52	1.13
R_2	1	0.83	0.55	0.46	0.64	1.52	1.11
Z_2	1	0.86	0.74	0.80	1.10	1.09	1.12
R_5	1	2.94	2.40	1.50	0.99	1.96	1.03
Z_5	1	1.14	1.09	1.20	0.86	1.03	0.99
R_{10}	1	6.16	7.10	3.82	1.95	1.91	0.99
Z_{10}	1	2.52	7.33	2.36	1.34	1.08	1.03
HM	1		0.67			1.16	1.07
GR_6	1		2.27			1.25	1.03
CV	1		1.86			1.58	0.92
DX	1		3.23			1.42	1.04
R_{50}	1	4.64		42.78			
Z_{50}	1	4.21		9.88^*			
R_{100}	1	4.12		106.18			

Superscript * means that DTS only obtains points close to the global minimum

Table 3: Normalization of average number of function evaluations

Problem	VNS	GVNS
RC	99	45
SH	305	623
R_2	176	274
R_{10}	1822	39062
GR_{10}	1320	1304
$H_{3,4}$	174	385
$H_{6,4}$	532	423
$S_{4,5}$	468	652
$S_{4,10}$	481	676
MG	17	73

Table 4: Average number of function evaluations - VNS against GVNS

Problem	VNS	GVNS
RC	1	0.45
SH	1	2.04
R_2	1	1.56
R_{10}	1	21.44
GR_{10}	1	0.99
$H_{3,4}$	1	2.21
$H_{6,4}$	1	0.80
$S_{4,5}$	1	1.39
$S_{4,10}$	1	1.41
MG	1	4.29

Table 5: Normalization of average number of function evaluations - VNS against GVNS

Problem	VNS	DTS
R_{50}	208	1080
Z_{50}	228	1043
R_{100}	1171	15270

Table 6: Average CPU time in seconds - Large size problems

becoming better and better when the dimension n of the problem increases from 2 to 10. In particular, VNS is able to significantly decrease (up to a factor 7 compared to some methods) the average number of evaluations of f on problems R_{10} and Z_{10}.

We now present the performance profiles of all 5 heuristics on 15 common problems (out of the 20 problems in Table 2) in Figure 1. A zoom for π between 1 and 5 is provided in Figure 2. The measure of performance is the average number of function evaluations (from Table 2). From Figure 2, we see that VNS is the best algorithm on 60% of the problems. Moreover, when VNS is not the best algorithm, it remains within a factor around 1.5 of the best method on 90% of the problems. Results are very satisfactory as VNS is the most robust but also the most efficient method among the 5 tested methods.

As we have already seen, our VNS behaves better and better on Rosenbrock and Zakharov problems when the dimension of the problem increases. This motivated us to perform several tests in larger dimension to see if our algorithm can achieve a significant gain in terms of number of function evaluations as well as CPU time. Table 1 shows the robustness of VNS compared to CHA and DTS methods on these 3 large size problems while Table 2 provides the average number of function evaluations. The performance with regard to the CPU time for VNS and DTS can be found in Table 6. VNS is the most robust as well as the most efficient on these 3

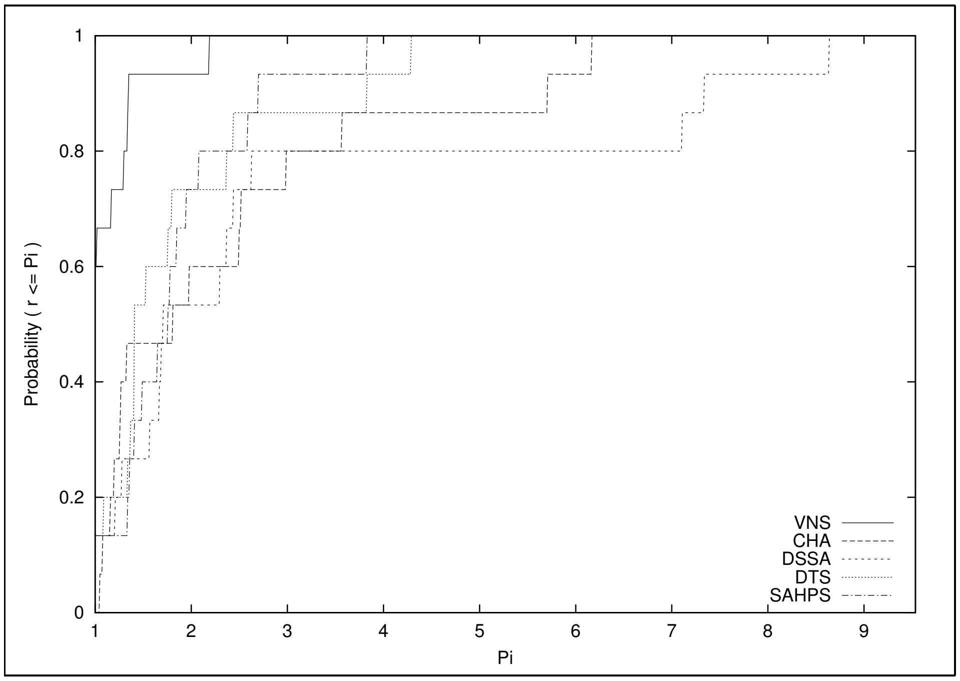


Figure 1: Average number of function evaluations

problems. Even if CHA is the best competitor, it requires up to 5 times more function evaluations.

These results are very encouraging for the future use of our algorithm to real applications. Comparing VNS with DTS, we clearly see that there is a computational overhead associated with our algorithm as the gain in CPU time (see Table 6) is less impressive compared to the gain in number of function evaluations (see Table 2). This is mainly due to our more costly local search and the QR-analysis discussed in Section 2.3. However, even if the tested functions are not cumbersome to compute, our method requires significantly less time to identify the global minimum, showing that the additional computational cost of our algorithm is compensated by its better efficiency. Given that, we are confident that the proposed algorithm will

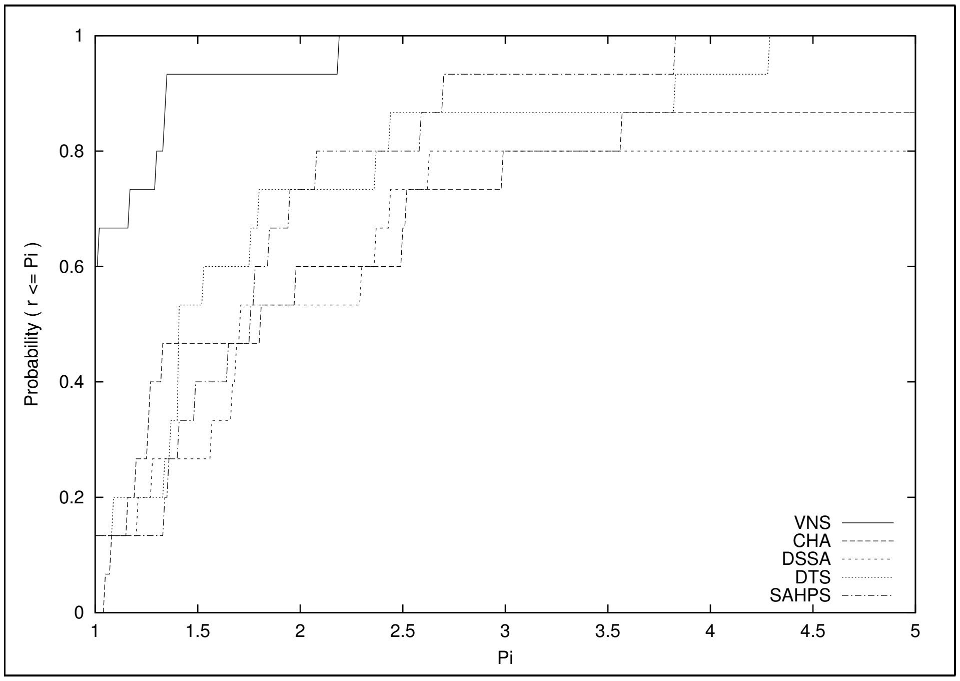


Figure 2: Zoom on the average number of function evaluations

reduce the CPU time for solving problems such that the evaluation of the objective function dominates all other computations of numerical algebra.

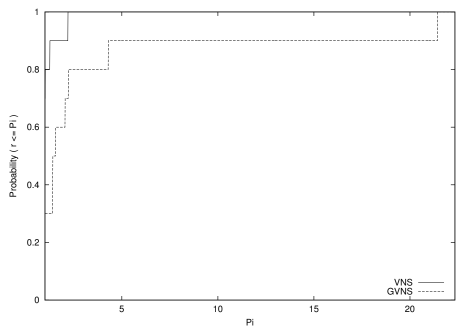
3.3.2 Comparison of VNS with GVNS

Now we consider the columns associated with VNS and GVNS in Table 1 as well as in Table 4. 10 common problems allow to challenge our VNS against the GVNS recently proposed by Mladenovic et al. (2006).

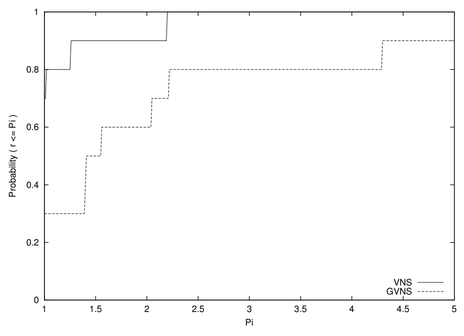
VNS and GVNS exhibit the same high level of robustness on the tested problems. GVNS always reaches the maximal rate of success while VNS attains 100% of success on all common problems, except one. Still, 78 runs on the Shubert (SH) function were successful.

From Table 4, we can note that VNS is the most efficient method on

7 out of the 10 problems. It is able to significantly decrease the number of function evaluations required to identify the global minimum of tested problems. When VNS is beaten, it remains within a reasonable factor of GVNS, requiring at worst the double of function evaluations on the Branin RCOS (RC) function. Contrarily to VNS, GVNS can be much slower on several problems. The performance profiles corresponding to Table 4 are provided in Figures 3(a) and 3(b).



(a) Full profile



(b) Zoom

Figure 3: Average number of function evaluations - VNS against GVNS

3.3.3 Comparison of the variants of VNS

We consider the three columns of Table 2 associated with the three variants of our VNS.

Comparing VNS with VNS_a, we clearly see that applying a full local search for each VNS iteration significantly increases the number of evaluations of f, up to a factor 2. As both algorithms are similar in terms of robustness (percentage of runs leading to the global minimum), it means that the tests proposed in Section 2.2 allow to reduce the number of function evaluations without deteriorating the capability of finding the global minimum. Moreover, we can argue that applying a classical VNS framework in which full local searches are applied to all neighbors generated within the VNS would be definitely too cumbersome from a computional point of view.

Comparison bewteen VNS and VNS_b shows that VNS is the best method for most of the 20 tested problems, with an important gain on some problems. From the related performance profile provided in Figure 4, it appears that VNS is the fastest method on about 75% of the problems. The gain obtained can be up to a factor of 1.5, meaning that using a purely random selection for search directions in order to compute neighbors may need 50% additional function evaluations compared to the strategy proposed in Section 2.3. This strategy prevents the algorithm from getting stucked in a given valley, where no significant improvement can be achieved, and gives the possibility to jump over valleys by using information on the curvature of f.

4 Conclusions and perspectives

The paper deals with nonlinear global optimization. We have proposed a new heuristic dedicated to identify the global minimum of an unconstrained nonlinear problem. Limiting the number of function evaluations is of major importance when the objective function is cumbersome to evaluate.

Within our VNS heuristic, we use an efficient nonlinear optimization algorithm using first and second order derivatives in order to quickly identify a local minimum of the problem. The ability to prematurely stop the local search allows to reduce the number of function evaluations (as well as the CPU time). Information about the function and its derivatives is also used

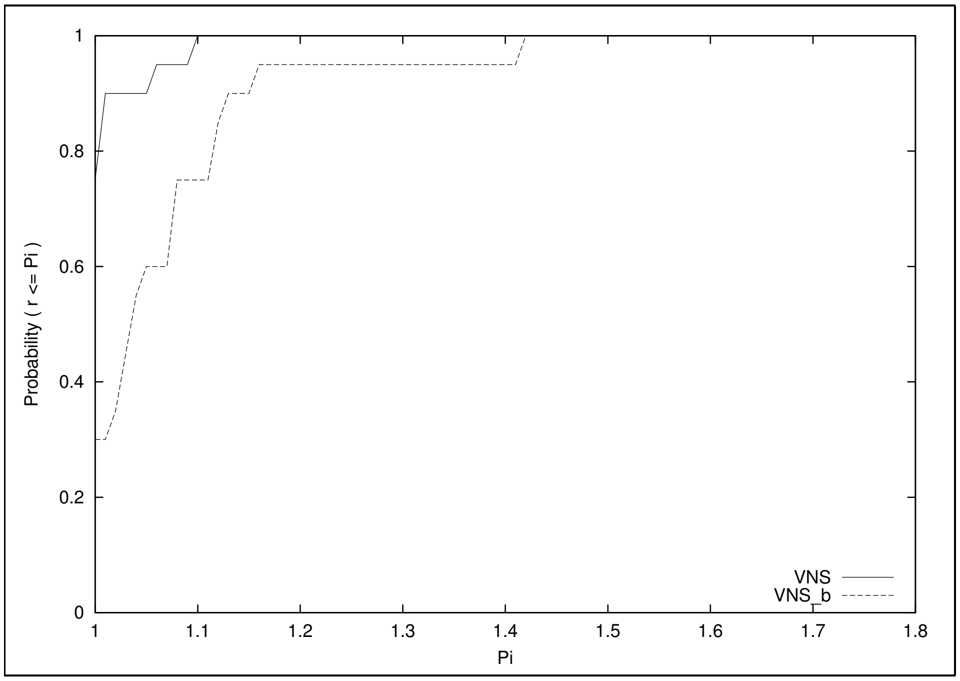


Figure 4: Average number of function evaluations for VNS and its variant

to compute the list of neighborhoods involved in the VNS and to select the associated neighbors. The better use of available and relevant information on f is determinant in the good behavior of the proposed method.

Numerical results obtained with our method are very satisfactory as VNS is the most robust but also the most efficient method on the problems we used in the experiments. The VNS framework makes the algorithm robust by its capability to explore and diversify the search space. The proposed algorithm significantly reduces the number of function evaluations compared to other efficient published methods. This makes the method particularly appealing for problems where the CPU time spent in function evaluations is dominant, such as those involving simulation. The results are consistent with the way the heuristic has been designed.

As conclusion, we could say that this paper represents a nice and profitable collaboration and interaction between nonlinear optimization and discrete optimization.

Several improvements should be investigated. We believe that we could have a better estimation of convergence basins of already encountered minima by stocking also other previous iterates and not only local minima. Also, defining p as dynamic from iteration to iteration of the VNS might also be interesting to investigate.

From a numerical point of view, we could implement a more efficient eigen-structure analysis in the VNS algorithm, aiming to reduce the computational cost of the overall method. Other stopping criteria could also be used to compare heuristics challenged in the paper. For instance, we could see how each method behaves with a given budget of CPU time or a given budget of function evaluations.

Another track of development would be to incorporate the VNS presented in this paper into an Adaptive Memory Method (AMM) framework (see Rochat and Taillard, 1995) in order to improve the diversification inside our algorithm.

Finally, we could investigate how the ideas presented in this paper could be tailored to constrained nonlinear global optimization.

References

Addis, B. and Leyffer, S. (2004). A trust-region algorithm for global optimization, *Technical report*, Mathematics and Computer Science Division, Argonne National Laboratory, USA.

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

437.

Androulakis, I. P., Maranas, C. D. and Floudas, C. A. (1995). αbb: A global optimization method for general constrained nonconvex problems, *Journal of Global Optimization* **7**(4): 337–363.

Battiti, R. and Tecchiolli, G. (1996). The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization, *Annals of Operations Research* **63**: 153–188.

Ben-Akiva, M. and Bierlaire, M. (2003). Discrete choice models with applications to departure time and route choice, *in* R. Hall (ed.), *Handbook of Transportation Science, Second Edition*, Kluwer Academic Publishers, Boston/Dordrecht/London, pp. 7–37.

Ben-Akiva, M. E. and Lerman, S. R. (1985). *Discrete Choice Analysis: Theory and Application to Travel Demand*, MIT Press, Cambridge, Ma.

Bertsekas, D. P. (1999). *Nonlinear Programming*, 2nd edn, Athena Scientific, Belmont.

Bierlaire, M. (2006). *Introduction à l'optimisation différentiable*, Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland. In press.

Chelouah, R. and Siarry, P. (1997). Enhanced simulated annealing for globally minimizing functions of many continuous variables, *ACM Transactions on Mathematical Software* **23**(2): 209–228.

Chelouah, R. and Siarry, P. (2000a). A continuous genetic algorithm designed for the global optimization of multimodal functions, *Journal of Heuristics* **6**: 191–213.

Chelouah, R. and Siarry, P. (2000b). Tabu search applied to global optimization, *European Journal of Operations Research* **123**: 256–270.

Chelouah, R. and Siarry, P. (2003). Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions, *European Journal of Operational Research* **148**: 335–348.

Conn, A. R., Gould, N. I. M. and Toint, P. L. (2000). *Trust-Region Methods*, Series on Optimization, MPS-SIAM, Philadelphia, USA.

Davis, L. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New-York.

Dekkers, A. and Aarts, E. (1991). Global optimization and simulated annealing, *Mathematical Programming* **50**: 367–393.

Dolan, E. D. and More, J. J. (2002). Benchmarking optimization software with performance profiles, *Mathematical Programming* **91**(2): 201–213.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey, *Theoretical Computer Science* **344(2-3)**: 243–278.

Dorsey, R. E. and Mayer, W. J. (1995). Genetic algorithms for estimation problems with multiple optima, nondifferentiability, and other irregular features, *Journal of Business & Economic Statistics* **13**(1): 53–66.

Eaton, J. W. (1997). GNU *Octave Manual*, Network Theory Limited, Bristol, United Kingdom.

Fletcher, R. and Leyffer, S. (2002). Nonlinear programming without a penalty function, *Mathematical Programming* **91**(2): 239–269.

Floudas, C., Pardalos, P., Adjiman, C., Esposito, W., Gumus, Z., Harding, S., Klepeis, J., Meyer, C. and Schweiger, C. (1999). *Handbook of Test Problems in Local and Global Optimization*, Kluwer Academic Publishers.

Franzè, F. and Speciale, N. (2001). Tabu search applied to global optimization, *International Journal for Numerical Methods in Engineering* **50**: 665–680.

Glover, F. (1986). Future paths for integer programming and linkage to artificial intelligence, *Computers & Operations Research* **13**: 533–549.

Glover, F. (1994). Tabu search for nonlinear and parametric optimization (with links to genetic algorithms), *Discrete Applied Mathematics* **49**: 231–255.

Glover, F. (1998). A template for scatter search and path relinking, *Lecture Notes on Computer Science* **1363**: 3 – 51.

Goffe, W. L., Ferrier, G. D. and Rogers, J. (1992). Simulated annealing: An initial application in econometrics, *Computer Science in Economics and Management* **5**: 133–146.

Goffe, W. L., Ferrier, G. D. and Rogers, J. (1994). Global optimization of statistical functions with simulated annealing, *Journal of Econometrics* **60**: 65–99.

Golub, G. H. and Loan, C. F. V. (1996). *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, USA.

Hansen, E. R. and Walster, G. W. (2003). *Global optimization using interval analysis*, CRC Press.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming, *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

Hedar, A. and Fukushima, M. (2002). Hybrid simulated annealing and direct search methods for nonlinear unconstrained global optimization, *Optimization Methods and Software* **17**: 891–912.

Hedar, A. and Fukushima, M. (2004). Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization, *Optimization Methods and Software* **19**: 291–308.

Hedar, A. and Fukushima, M. (2006). Tabu search directed by direct search methods for nonlinear global optimization, *European Journal of Operations Research* **170**: 329–349.

Henrion, D. and Lasserre, J.-B. (2003). Gloptipoly: Global optimization over polynomials with matlab and sedumi, *ACM Transactions on Mathematical Software* **29**(2): 165–194.

Hertz, A., Plumettaz, M. and Zufferey, N. (2006). Variable space search for graph coloring, *submitted for publication* .

Horst, R. and Thoai, N. V. (1999). Dc programming: Overview, *Journal of Optimization Theory and Applications* **103**(1): 1–43.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. (1983). Optimization by simulated annealing, *Science* **220**(5498): 671–680.

Klepeis, J. L., Pieja, M. J. and Floudas, C. A. (2003). Hybrid global optimization algorithms for protein structure prediction: Alternating hybrids, *Biophysical Journal* **84**: 869–882.

Lasserre, J. B. (2001). Global optimization with polynomials and the problem of moments, *SIAM Journal on Optimization* **11**(3): 796–817.

Lasserre, J. B. (2004). Solving nonconvex optimization problems, *IEEE Control Systems Magazine* **24**: 72–83.

Le, T. H. A. and Pham, D. T. (2005). The dc (difference of convex functions) programming and dca revisited with dc models of real world nonconvex optimization problems, *Annals of Operations Research* **133**: 23–46.

Lin, Y. and Stadtherr, M. A. (2004). Advances in interval methods for deterministic global optimization in chemical engineering, *Journal of Global Optimization* **29**(3): 281–296.

Locatelli, M. (2000). Simulated annealing algorithms for continuous global optimization: Convergence conditions, *Journal of Optimization Theory and Applications* **104**(1): 121–133.

Mladenovic, N., Drazic, M., Kovacevic-Vujcic, V. and Cangalovic, M. (2006). General variable neighborhood search for the continuous optimization, *Les Cahiers du GERAD* **48**.

Mladenovic, N. and Hansen, P. (1997). Variable neighborhood search, *Computers & Operations Research* **24**(11): 1097–1100.

Nocedal, J. and Wright, S. J. (1999). *Numerical optimization*, Operations Research, Springer Verlag, New-York.

Renders, J.-M. and Flasse, S. P. (1996). Hybrid methods using genetic algorithms for global optimization, *IEEE Transactions on Systems, Man and Cybernetics - Part B:Cybernetics* **26**(2): 243–258.

Rochat, Y. and Taillard, E. (1995). Probabilistic diversification and intensification in local search for vehicle routing, *Journal of Heuristics* **1**: 147–167.

Sandgren, E. and Cameron, T. M. (2002). Robust design optimization of structures through consideration of variation, *Computers & Structures* **80**(20-21): 1605–1613.

Sartenaer, A. (1997). Automatic determination of an initial trust region in nonlinear programming, *SIAM Journal on Scientific Computing* **18**(6): 1788–1803.

Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization, *SIAM Journal on Numerical Analysis* **20**(3): 626–637.

Teh, Y. S. and Rangaiah, G. P. (2003). Tabu search for global optimization of continuous functions with application to phase equilibrium calculations, *Computers and Chemical Engineering* **27**(11): 1665–1679.

Toint, P. L. (1981). Towards an efficient sparsity exploiting newton method for minimization, *in* I. S. Duff (ed.), *Sparse Matrices and Their Uses*, pp. 57–88.

Vaz, A. I. F. and Vicente, L. N. (to appear). A particle swarm pattern search method for bound constrained global optimization, *Journal of Global Optimization* . Accepted for publication.

A List of test functions

A.1 Branin RCOS Function (RC)

- 2 variables
- $RC(x_1, x_2) = (x_2 - (5.1/4\pi^2)x_1^2 + (5/\pi)x_1 - 6)^2 + 10(1 - (1/8\pi))\cos(x_1) + 10$
- Range of initial points: $-5 < x_1 < 10$, $0 < x_2 < 15$.
- No local minimum
- Global minima: $(x_1^*, x_2^*) = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$; $RC(x_1, x_2^*) = 0.397887$

A.2 Easom Function (ES)

- 2 variables
- $ES(x_1, x_2) = -\cos(x_1)\cos(x_2)e^{-(x_1-\pi)^2-(x_2-\pi)^2}$
- Range of initial points: $-10 < x_j < 10$, $j = 1, 2$.
- Several local minima
- Global minimum: $(x_1^*, x_2^*) = (\pi, \pi); ES(x_1, x_2^*) = -1$

A.3 Rastrigin Function (RT)

- 2 variables
- $RT(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
- Range of initial points: $-1 < x_j < 1$, $j = 1, 2$.
- Many local minima
- Global minimum: $(x_1^*, x_2^*) = (0, 0); RT(x_1, x_2^*) = 0$

A.4 Shubert Function (SH)

- 2 variables
- $SH(x_1, x_2) = (\sum_{j=1}^{5} j\cos((j+1)x_1 + j))(\sum_{j=1}^{5} j\cos((j+1)x_2 + j))$
- Range of initial points: $-10 < x_j < 10$, $j = 1, 2$.
- 760 local minima
- 18 global minima: $SH(x_1, x_2^*) = -186.7309$

A.5 De Joung Function (DJ)

- 3 variables
- $DJ(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$
- Range of initial points: $-5 < x_j < 5$, $j = 1, 2, 3$.
- No local minimum
- Global minimum: $(x_1^*, x_2^*, x_3^*) = (0, 0, 0); DJ(x_1, x_2^*, x_3^*) = 0$

A.6 Hartmann Function ($H_{3,4}$)

- 3 variables
- $H_{3,4}(x) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2}$
- Range of initial points: $0 < x_j < 1$, $j = 1, 2, 3$.
- 4 local minima
- Global minimum: $x* = (0.114614, 0.555649, 0.852547)$;
  $H_{3,4}(x*) = -3.86278$

i	a_{ij}			c_i	p_{ij}		
1	3.0	10.0	30.0	1.0	0.689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.0381	0.5743	0.8828

A.7 Hartmann Function ($H_{6,4}$)

- 6 variables
- $H_{6,4}(x) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2}$
- Range of initial points: $0 < x_j < 1$, $j = 1, \ldots, 6$.
- 6 local minima
- Global minimum: $x* = (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657300)$; $H_{6,4}(x*) = -3.32237$

i	a_{ij}						c_i
1	10.0	3.0	17.0	3.50	1.70	8.00	1.0
2	0.05	10.0	17.0	0.10	8.00	14.00	1.2
3	3.00	3.50	1.70	10.0	17.00	8.00	3.0
4	17.00	8.00	0.05	10.00	0.10	14.00	3.2

i	p_{ij}					
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

A.8 Shekel Functions ($S_{4,m}$)

- 4 variables
- $S_{4,m}(x) = -\sum_{i=1}^{m} (\sum_{j=1}^{4} (x_j - a_{ij})^2 + c(i))^{-1}$
- 3 functions are considered, namely: $S_{4,5}$, $S_{4,7}$ and $S_{4,10}$
- Range of initial points: $0 < x_j < 10$, $j = 1, \ldots, 4$.
- m local minima
- Global minimum: $x* = (4, 4, 4, 4)$; $S_{4,5}(x*) = -10.1532$, $S_{4,7}(x*) = -10.4029$ and $S_{4,10}(x*) = -10.5364$

i	a_{ij}				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

A.9 Rosenbrock Function (R_n)

- n variables with $n = 2, 5, 10, 50, 100$
- $R_n(x) = \sum\limits_{j=1}^{n-1} (100(x_j^2 - x_{j+1})^2 + (x_j - 1)^2)$
- Range of initial points: $-5 < x_j < 10$, $j = 1, 2, \ldots, n$
- No local minimum
- Global minimum: $x^* = (1, \ldots, 1)$, $R_n(x^*) = 0$

A.10 Zakharov Function (Z_n)

- n variables with $n = 2, 5, 10, 50$
- $Z_n(x) = \sum\limits_{j=1}^{n} x_j^2 + (\sum\limits_{j=1}^{n} 0.5jx_j)^2 + (\sum\limits_{j=1}^{n} 0.5jx_j)^4$
- Range of initial points: $-5 < x_j < 10$, $j = 1, 2, \ldots, n$
- No local minimum
- Global minimum: $x^* = (0, \ldots, 0)$, $R_n(x^*) = 0$

A.11 Hump Function (HM)

- 2 variables
- $HM(x_1, x_2) = 1.0316285 + 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
- Range of initial points: $-5 < x_j < 5$, $j = 1, 2$.
- No local minimum
- Global minima: $(x_1^*, x_2^*) = (0.0898, -0.7126).(-0.0898, 0.7126)$;
  $HM(x_1, x_2^*) = 0$

A.12 Griewank Function (GR_n)

- n variables with $n = 6, 10$
- $GR_n(x) = \sum\limits_{j=1}^{n} x_j^2/4000 - \prod\limits_{j=1}^{n} \cos(x_j/\sqrt{j}) + 1$
- Range of initial points: $-10 < x_j < 10$, $j = 1, 2, \ldots, n$
- Many local minima
- Global minimum: $x^* = (0, \ldots, 0)$, $GR_n(x^*) = 0$

A.13 Colville Function (CV)

- 4 variables
- $CV(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$
- Range of initial points: $-10 < x_j < 10$, $j = 1, \ldots, 4$
- No local minimum
- Global minimum: $x^* = (1, 1, 1, 1)$, $CV(x^*) = 0$

A.14 Dixon Function (DX)

- 10 variables
- $DX(x) = (1 - x_1)^2 + (1 - x_{10})^2 + \sum\limits_{j=1}^{9} (x_j^2 - x_{j+1})^2$
- Range of initial points: $-10 < x_j < 10$, $j = 1, \ldots, 10$

- No local minimum
- Global minimum: $x^* = (1, \ldots, 1)$, $DX(x^*) = 0$

A.15 Martin&Gaddy Function (MG)

- 2 variables
- $MG(x) = (x_1 - x_2)^2 + (\frac{x_1 + x_2 - 10}{3})^2$
- Range of initial points: $-20 < x_j < 20$, $j = 1, 2$
- No local minimum
- Global minimum: $x^* = (5, 5)$, $MG(x^*) = 0$