CONDOR, an new Parallel, Constrained extension of Powell's UOBYQA algorithm. Experimental results and comparison with the DFO algorithm.

Frank Vanden Berghen, Hugues Bersini

IRIDIA, Université Libre de Bruxelles
Avenue Franklin Roosevelt, 50 CP194/6
1050 Brussels, Belgium.

Abstract

This paper presents an algorithmic extension of Powell's UOBYQA algorithm ("*Unconstrained Optimization BY Quadratical Approximation*"). We start by summarizing the original algorithm of Powell and by presenting it in a more comprehensible form. Thereafter, we report comparative numerical results between UOBYQA, DFO and a parallel, constrained extension of UOBYQA that will be called in the paper CONDOR ("*COnstrained, Non-linear, Direct, parallel Optimization using trust Region method for high-computing load function*"). The experimental results are very encouraging and validate the approach. They open wide possibilities in the field of noisy and high-computing-load objective functions optimization (from two minutes to several days) like, for instance, industrial shape optimization based on CFD (Computation Fluid Dynamic) codes or PDE (partial differential equations) solvers. Finally, we present a new, easily comprehensible and fully stand-alone implementation in C++ of the parallel algorithm.

Key words: non-linear optimization, lagrange interpolation, trust region method, optimal shape design, parallel computing

1 Introduction

Powell's UOBYQA algorithm ([35] or [36]) is a new algorithm for unconstrained, direct optimization that take into account the curvature of the ob-

Email address: fvandenb@iridia.ulb.ac.be, Bersini@ulb.ac.be (Frank Vanden Berghen, Hugues Bersini).

jective function, leading to a high convergence speed. UOBYQA is the direct successor of COBYLA [32]. Classical quasi-newton methods also use curvature information ([28,2,1,15,18]) but they need explicit gradient information, usually obtained by finite difference. In the field of aerodynamical shape optimization, the objective functions are based on expensive simulation of CFD (computation fluid dynamic) codes (see [42,29,13,30,31]) or PDE (partial differential equations) solvers. For such applications, choosing an appropriate step size for approximating the derivatives by finite differences is quite delicate: function evaluation is expensive and can be very noisy. For such type of application, finite difference quasi-newton methods need to be avoided. Indeed, even if actual derivative information were available, quasi-Newton methods might be a poor choice because adversely affected by function inaccuracies (see [17]). Instead, direct optimization methods [16] are relatively insensitive to the noise. Unfortunately, they usually require a great amount of function evaluations.

UOBYQA and CONDOR sample the search space, making evaluations in a way that reduces the influence of the noise. They both construct a full quadratical model based on Lagrange Interpolation technique [14,38,8,43,44,34]. The curvature information is obtained from the quadratical model. This technique is less sensitive to the noise and leads to high quality local quadratical models which directly guide the search to the nearest local optimum. These quadratical models are built using the least number of evaluations (possibly reusing old evaluations).

DFO [12,11] is an algorithm by A.R.Conn, K. Scheinberg and Ph.L. Toint. It's very similar to UOBYQA and CONDOR. It has been specially designed for small dimensional problems and high-computing-load objective functions. In other words, it has been designed for the same kind of problems that CONDOR. DFO also uses a model build by interpolation. It is using a Newton polynomial instead of a Lagrange polynomial. When the DFO algorithm starts, it builds a linear model (using only $n + 1$ evaluations of the objective function; n is the dimension of the search space) and then directly uses this simple model to guide the research into the space. In DFO, when a point is "too far" from the current position, the model could be *invalid* and could not represent correctly the local shape of the objective function. This "far point" is rejected and replaced by a closer point. This operation unfortunately requires an evaluation of the objective function. Thus, in some situation, it is preferable to lower the degree of the polynomial which is used as local model (and drop the "far" point), to avoid this evaluation. Therefore, DFO is using a polynomial of degree oscillating between 1 and a "full" 2. In UOBYQA and CONDOR, we use the Moré and Sorenson algorithm [27,9] for the computation of the trust region step. It is very stable numerically and give *very high* precision results. On the other hand, DFO uses a general purpose tool (NPSOL [20]) which gives high quality results *but* that cannot be compared to the Moré

and Sorenson algorithm when precision is critical. An other critical difference between DFO and CONDOR/UOBYQA is the formula used to update the local model. In DFO, the quadratical model built at each iteration is not defined uniquely. For a unique quadratical model in n variables one needs at least $\frac{1}{2}(n+1)(n+2) = N$ points and their function values. "In DFO, models are often build using many fewer points and such models are not uniquely defined" (citation from [11]). The strategy used inside DFO is to select the model with the smallest Frobenius norm of the Hessian matrix. This update is highly numerically instable [37]. Some recent research at this subject have maybe found a solution [37] but this is still "work in progress". The model DFO is using can thus be very inaccurate.

In contrast to UOBYQA and CONDOR, DFO uses linear or quadratical models to guide the search, thus requiring less function evaluations to build the local models. Based on our experimental results, we surprisingly discovered that CONDOR used less function evaluations than DFO to reach an optimum point, despite the fact that the cost to build a local model is higher (see section 5 presenting numerical results). This is most certainly due to an heuristic (see section 5.1 at this subject) used inside UOBYQA and CONDOR which allows to build quadratical models at very "low price".

The algorithm used inside UOBYQA is thus a good choice to reduce the number of function evaluations in the presence of noisy and high computing load objective functions. Since description of this algorithm in the literature is hard to find and rather unclear, a first objective of the paper is to provide an updated and more accessible version of it.

When concerned with CPU time to reach the local optimum, computer parallelization of the function evaluations is always an interesting road to pursue. Indeed, PDS (Parallel direct search) largely exploits this parallelization to reduce the optimization time. We take a similar road by proposing an extension of the original UOBYQA that can use several CPU's in parallel: CONDOR. Our experimental results show that this addition makes CONDOR the fastest available algorithm for noisy, high computing load objective functions (fastest in terms of number of function evaluations).

In substance, this paper proposes a new, simpler and clearer, parallel implementation in C++ of UOBYQA: the CONDOR optimizer. A version of CONDOR allowing constraints is discussed in [41].

The paper is structured in the following way:

- **Section 1:** The introduction.
- **Section 2:** Basic description of the UOBYQA algorithm with hints to possible parallelization.
- **Section 3:** New, more in depth, comprehensible presentation of UOBYQA

with a more precise description of the parallel extension.
- **Section 4:** In depth description of this parallel extension.
- **Section 5:** Experimental results: comparison between CONDOR, the original Powell's UOBYQA, DFO, LANCELOT, COBYLA, PDS.
- **Section 5:** How to get the code and conclusions.

2 Basic description of Powell's UOBYQA algorithm

Let n be the dimension of the search space. Let $f(x)$ be the objective function to minimize. We want to find $x^* \in \Re^n$ which satisfies:

$$f(x^*) = \min_x f(x) \tag{1}$$

In the following algorithm, ρ is the usual trust region radius. We do not allow ρ to increase because this would necessitate expensive decrease later. We will introduce Δ, another trust region radius that satisfies $\Delta \geq \rho$. The advantage of Δ is to allow the length of the steps to exceed ρ and to increase the efficiency of the algorithm.

Let x_{start} be the starting point of the algorithm. Let ρ_{start} and ρ_{end} be the initial and final value of the trust region radius ρ.

Definition 1 *The local approximation $q_k(s)$ of $f(x)$ is* valid *in $B_k(\rho)$ (a ball of radius ρ around x_k) when $|f(x_k + s) - q_k(s)| \leq \kappa\rho^2 \quad \forall \|s\| \leq \rho$ where κ is a given constant independent of x.*

Basically, Powell's UOBYQA algorithm does the following (for a more detailed explanation, see section 3 or [35]):

(1) Create an interpolation polynomial $q_0(s)$ of degree 2 which interpolates the objective function around x_{start}. All the points in the interpolation set Y (used to build $q(x)$) are separated by a distance of approximatively ρ_{start}. Set $x_k =$ *the best point of the objective function known so far*. Set $\rho_0 = \rho_{start}$. In the following algorithm, $q_k(s)$ is the quadratical approximation of $f(x)$ around x_k: $q_k(s) = f(x_k) + g_k^t s + s^t H_k s$ where g_k is an approximation of the gradient of $f(x)$ evaluated at x_k and H_k is an approximation of the Hessian matrix of $f(x)$ evaluated at x_k.
(2) Set $\Delta_k = \rho_k$
(3) Inner loop: solve the problem for a given precision of ρ_k.
  (a) (i)

$$\text{Solve } s_k = \min_{s \in \Re^n} q_k(s) \text{ subject to } \|s\|_2 < \Delta_k \tag{2}$$

  (ii) If $\|s_k\| < \frac{1}{2}\rho_k$, then break and go to step 3(b) because, in order to do such a small step, we need to be sure that the model is

valid.

(iii) Evaluate the function $f(x)$ at the new position $x_k + s_k$. Update (like described in next section, 4(a)viii. to 4(a)x.) the trust region radius Δ_k and the current best point x_k using classical trust region technique. Include the new x_k inside the interpolation set Y. Update $q_k(s)$ to interpolate on the new Y.

(iv) If some progress has been achieved (for example, $\|s_k\| > 2\rho$ or there was a reduction $f(x_{k+1}) < f(x_k)$), increment k and go back to step 3(a)**i**, otherwise continue.

(b) Test the validity of $q_k(x)$ in $B_k(\rho)$, like described in [35].

- **Model is invalid:**
  Improve the quality of the model $q(x)$: Remove the worst point of the interpolation set Y and replace it (one evaluation required!) with a new point x_{new} such that: $\|x_{new} - x_k\| < \rho$ and the precision of $q_k(s)$ is substantially increased.
- **Model is valid:**
  If $\|s_k\| > \rho_k$ go back to step 3(a), otherwise continue.

(4) **Reduce** ρ since the optimization steps s_k are becoming very small, the accuracy needs to be raised.

(5) If $\rho = \rho_{end}$ stop, otherwise increment k and go back to step **2**.

Basically, ρ is the distance (Euclidian distance) which separates the points where the function is sampled. When the iterations are unsuccessful, the trust region radius Δ_k decreases, preventing the algorithm to achieve more progress. At this point, loop 3(a)i to 3(a)iv is exited and a function evaluation is required to increase the quality of the model (step 3(b)). When the algorithm comes close to an optimum, the step size becomes small. Thus, the inner loop (steps 3(a)i. to 3(a)iv.) is usually exited from step 3(a)ii, allowing to skip step 3(b) (hoping the model is *valid*), and directly reducing ρ in step 4.

The most inner loop (steps 3(a)i. to 3(a)iv.) tries to get from $q_k(s)$ good search directions without doing any extra evaluation to maintain the quality of $q_k(s)$ (The evaluations that are performed on step 3(a)i) have another goal). Only inside step 3(b), evaluations are performed to increase this quality (called a "model step") and only at the condition that the model has been proven to be invalid (to spare evaluations!).

Notice the update mechanism of ρ in step 4. This update occurs only when the model has been validated in the trust region $B_k(\rho)$ (when the loop 3(a) to 3(b) is exited). The function cannot be sampled at point too close to the current point x_k without being assured that the model is *valid* in $B_k(\rho)$. This mechanism protects us against noise.

The different evaluations of $f(x)$ are used to:

(a) guide the search to the minimum of $f(x)$ (see inner loop in the steps 3(a)i. to 3(a)iv.). To guide the search, the information gathered until now and available in $q_k(s)$ is *exploited*.
(b) increase the quality of the approximator $q_k(x)$ (see step 3(b)). To avoid the degeneration of $q_k(s)$, the search space needs to be additionally *explored*.

(a) and (b) are antagonist objectives like frequently encountered in the *exploitation/exploration* paradigm. The main idea of the parallelization of the algorithm is to perform the *exploration* on distributed CPU's. Consequently, the algorithm will have better models $q_k(s)$ of $f(x)$ available and choose better search direction, leading to a faster convergence.

Let's assume that the current minimization step s_k pushes CONDOR to enter into the infeasible space. We then activate all the box and linear constraints which have been violated and we re-compute a solution of equation 2 in the Reduced-Space of the Active Box and Linear Constraints (RSABLC) to obtain a new s_k. A basis of the RSABLC is needed and is built using a QR factorisation with pivoting [18,22]. If some non-linear constraints are active, an SQP algorithm [2] performed inside the RSABLC is used to compute the new . The decision to remove a constraint J out of the active set of the constraints is mainly based on the value of the Lagrangian variable (also called dual variable) associated to the constraint J. For in depth explanation of the constrained step inside CONDOR, see [41].

UOBYQA and CONDOR are inside the class of algorithm which are proven to be globally convergent to a local (maybe global) optimum: They are both using conditional models as described in [12,8].

3 The UOBYQA algorithm in depth

We will now detail the UOBYQA algorithm [35] and a part of its parallel extension. As a result of this parallel extension, the points 3, 4(a)i, 4(b), 9 constitute an original contribution of the authors. When only one CPU is available, these points are simply skipped. The point 4(a)v is also original and has been added to make the algorithm more robust against noise in the evaluation of the objective function. These points will be detailed in the next section. The other points of the algorithm belong to the original UOBYQA.

Let $noise_a$ and $noise_r$, be the absolute and relative error on the evaluation of the objective function. These constants are given by the user. By default, they are null.

(1) Set $\Delta = \rho$, $\rho = \rho_{start}$ and generate a first interpolation set $Y = \{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(N)}\}$

around x_{start} (with $N = (n+1)(n+2)/2$). This set is "poised", meaning that the Vandermonde determinant of Y is non-null (see [14,38]). The set Y is generated using the algorithm described in [35].

(2) In what follows, the index k is always the index of the best point of the set $Y = \{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(N)}\}$. The points in Y will be noted in **bold** with parenthesis around their subscript. Let $x_{(base)} := \boldsymbol{x}_{(k)}$. Set $F_{old} := f(x_{(base)})$. Apply a translation of $-x_{(base)}$ to all the dataset $\{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(N)}\}$ and generate the quadratical polynomial $q(x)$, which intercepts all the points in the dataset Y. The translation is achieved to increase the quality of the interpolation. $q_k(s)$ is built using Multivariate Lagrange Interpolation. It means that $q_k(s) = \sum_{i=1}^{N} f(\boldsymbol{x}_{(i)}) P_i(s)$ where the $P_i(s)$ are the Lagrange polynomials associated to the dataset Y. The $P_i(s)$ have the following property: $P_i(\boldsymbol{x}_{(j)}) = \delta_{(i,j)}$ where $\delta_{(i,j)}$ is the Kronecker delta (see [14,38] about multivariate Lagrange polynomial interpolation). The complete procedure is given in [35].

(3) *Parallel extension:* Start the "parallel computations" on the different computer nodes. See next section for more details.

(4) (a) (i) *Parallel extension:* Check the results of the parallel computation and use them to increase the quality of $q_k(s)$. See next section for more details.

(ii) Calculate the "Trust region step" s^*: s^* is the solution of:

$$\min_{s \in \Re^n} q(\boldsymbol{x}_{(k)} + s) = \min_{s \in \Re^n} q_k(s) \text{ subject to } \|s\|_2 < \Delta$$

This is a quadratic program with a non-linear constraint. It's solved using Moré and Sorenson algorithm (see [27,9]). The original implementation of the UOBYQA algorithm uses a special tri-diagonal decomposition of the Hessian to obtain high speed (see [33]). CONDOR uses a direct, simpler, implementation of the Moré and Sorenson algorithm.

(iii) If $\|s\| < \dfrac{\rho}{2}$, then break and go to step 4(b): the model needs to be validated before doing such a small step.

(iv) Let $R := q(\boldsymbol{x}_{(k)}) - q(\boldsymbol{x}_{(k)} + s^*) \geq 0$, the predicted reduction of the objective function.

(v) One original addition to the algorithm is the following:
Let $noise := \frac{1}{2} \max[noise_a * (1 + noise_r), noise_r |f(\boldsymbol{x}_{(k)})|]$.
If $(R < noise)$, break and go to step 4(b).

(vi) Evaluate the objective function $f(x)$ at point $x = x_{(base)} + \boldsymbol{x}_{(k)} + s^*$. The result of this evaluation is stored in the variable F_{new}.

(vii) Compute the agreement r between $f(x)$ and the model $q(x)$:

$$r = \frac{F_{old} - F_{new}}{R}$$

(viii) Update the trust region radius Δ:

$$\begin{cases} \max[\Delta, \frac{5}{4}\|s\|, \rho + \|s\|] & \text{if } 0.7 \leq r, \\ \max[\frac{1}{2}\Delta, \|s\|] & \text{if } 0.1 < r < 0.7, \\ \frac{1}{2}\|s\| & \text{if } r \leq 0.1 \end{cases}$$

If $(\Delta < 1.5\rho)$, set $\Delta := \rho$.

(ix) Store $\boldsymbol{x}_{(k)} + s^*$ inside the interpolation dataset Y. To do so, first, choose the worst point $\boldsymbol{x}_{(t)}$ of the dataset (The exact, detailed algorithm, is given in [35]). This is the point which gives the highest contribution to following bound on the interpolation error [34]:

$$\begin{matrix} \textit{Interpolation} \\ \textit{error at point } y \end{matrix} = |q_k(y) - f(y)| < \frac{M}{6} \sum_{j=1}^{N} |P_j(y)| \|y - \boldsymbol{x}_{(j)}\|^3 \tag{3}$$

Where M is a bound on the third derivative of $f(x)$: $|\phi'''(\alpha)| \leq M$ where $\phi(\alpha) = f(y + \alpha\bar{d}), \quad \alpha \in \Re, \bar{d} \in \Re^n$ and $\|\bar{d}\| = 1$, and where $P_j(y)$ are the Lagrange Polynomials used to construct $q_k(y)$.(see [14,38] about multivariate Lagrange polynomial interpolation).

Secondly, replace the point $\boldsymbol{x}_{(t)}$ by $\boldsymbol{x}_{(k)} + s^*$ and recalculate the new quadratic $q_k(s)$ which interpolates the new dataset.

Definition 2 *The ModelStep is* $\|\boldsymbol{x}_{(t)} - (\boldsymbol{x}_{(k)} + s^*)\|$

(x) Update the index k of the best point in the dataset.
Set $F_{new} := \min[F_{old}, F_{new}]$.

(xi) Update the value M (the bound on the third derivative of $f(x)$) using:

$$M_{new} = \max\left[M_{old}, \frac{|q_k(x) - f(x)|}{\frac{1}{6}\sum_{j=1}^{N} |P_j(x)| \|x - \boldsymbol{x}_{(j)}\|^3}\right] \tag{4}$$

(xii) If there is an improvement in the quality of the solution ($F_{new} < F_{old}$) OR if ($\|s^*\| > 2\rho$) OR if $ModelStep > 2\rho$ then go back to point 4(a)i, otherwise, continue.

(b) *Parallel extension:* Check the results of the parallel computation and use them to increase the quality of $q_k(s)$. See next section for more details.

(c) The validity of our model in $B_k(\rho_k)$, a ball of radius ρ_k around $\boldsymbol{x}_{(k)}$ now needs to be checked based on equations (7) and (8).

- **Model is invalid:**
  Improve the quality of our model $q(x)$. This is called a "model improvement step". Remove the worst point $\boldsymbol{x}_{(j)}$ of the dataset and replace it by a better point. This better point is computed

using an algorithm described in [35]. If a new function evaluation has been made, the value of M must also be updated. Possibly, an update of the index k of the best point in the dataset Y and F_{old} is required. Once this is finished, go back to step 4(a).

- **Model is valid:**
  If $\|s^*\| > \rho$ go back to step 4(a), otherwise continue.

(5) If $\rho = \rho_{end}$, the algorithm is nearly finished. Go to step 8, otherwise continue to the next step.

(6) Update of trust region radius ρ.

$$\rho_{new} = \begin{cases} \rho_{end} & \text{if } \rho_{end} < \rho \leq 16\rho_{end} \\ \sqrt{\rho_{end}\,\rho} & \text{if } 16\rho_{end} < \rho \leq 250\rho_{end} \\ 0.1\rho & \text{if } 250\rho_{end} < \rho \end{cases} \tag{5}$$

Set $\Delta := \max[\frac{\rho}{2}, \rho_{new}]$. Set $\rho := \rho_{new}$.

(7) Set $x_{(base)} := x_{(base)} + \boldsymbol{x}_{(k)}$. Apply a translation of $-\boldsymbol{x}_{(k)}$ to $q_k(s)$, to the set of Newton polynomials P_i which defines $q_k(s)$ and to the whole dataset $Y = \{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(N)}\}$. Go back to step 4.

(8) The iterations are now complete but one more value of $f(x)$ may be required before termination. Indeed, it is known from step 4(a)iii and step 4(a)v of the algorithm that the value of $f(x_{(base)} + \boldsymbol{x}_{(k)} + s^*)$ could not have been computed. Compute $F_{new} := f(x_{(base)} + \boldsymbol{x}_{(k)} + s^*)$.

- if $F_{new} < F_{old}$, the solution of the optimization problem is $x_{(base)} + \boldsymbol{x}_{(k)} + s^*$ and the value of f at this point is F_{new}.
- if $F_{new} > F_{old}$, the solution of the optimization problem is $x_{(base)} + \boldsymbol{x}_{(k)}$ and the value of f at this point is F_{old}.

(9) *Parallel extension:* Stop the parallel computations if necessary.

The aim of the parallelization is to evaluate $f(x)$ at positions which could substantially increase the quality of the approximator $q_k(s)$. The way to choose such positions is explained in section 4.

4 The parallel extension of UOBYQA

We will use a client-server approach. The main node, the server, will run two concurrent processes:

- The **main process** on the main computer is the classical non-parallelized version of the algorithm, described in the previous section. There is an exchange of information with the second/parallel process on steps 4(a)i and 4(b) of the original algorithm.
- The goal of the **second/parallel process** on the main computer is to

increase the quality of the model $q_k(s)$ by using client computers to sample $f(x)$ at specific interpolation sites.

In an ideal scenario:

- The **main process** will always stay inside the most inner loop 4(a)i to 4(a)xii. Hoping that the evaluation on the client computers always provide a *valid* local model $q_k(s)$, progress will constantly be achieved.
- The **main process** exits the inner loop at step 4(a)iii: Near an optimum, the model is ideally *valid* and ρ can be decreased.

The client nodes are performing the following:

(1) Wait to receive from the second/parallel process on the server a sampling site (a point).
(2) Evaluate the objective function at this site and return immediately the result to the server.
(3) Go to step 1.

Several strategies have been tried to select good sampling sites. We describe here the most promising one. The second/parallel task is the following:

A. Make a local copy $q_{(copy)}(s)$ of $q_k(s)$ (and of the associated Lagrange Polynomials $P_j(x)$)
B. Make a local copy $J_{(copy)}$ of the dataset $J = \{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(N)}\}$.
C. Find the index j of the point inside $J_{(copy)}$ the further away from $\boldsymbol{x}_{(k)}$.
D. Replace $\boldsymbol{x}_{(j)}$ by a better point $\boldsymbol{x}_{(j)} + d$ which will increase the quality of the approximation of $f(x)$. The computation of this point is detailed below.
E. Ask for an evaluation of the objective function at point $\boldsymbol{x}_{(j)} + d$ using a free client computer to perform the evaluation. If there is still a client idle, go back to step **C**.
F. Wait for a node to finish its evaluation of the objective function $f(x)$. Most of the time, the second/parallel task will be blocked here without consuming any resources.
G. Update $q_{(copy)}(x)$ using the newly received evaluation. Update $J_{(copy)}$. go to step **C**.

In the parallel/second process we are always working on a copy of $q_k(x)$, J and $P_{j,(copy)}(x)$ to avoid any side effect with the main process which is guiding the search. The communication and exchange of information between these two processes are done only at steps 4(a)i and 4(b) of the **main process** described in the previous section. Each time the main process checks the results of the parallel computations the following is done:

i. Wait for the parallel/second task to enter the step **F** described above and

block the parallel task inside this step **F** for the time needed to perform the points **ii** and **iii** below.

ii. Update of $q_k(s)$ using all the points calculated in parallel, discarding the points that are too far away from $\boldsymbol{x}_{(k)}$ (at a distance greater than ρ)(The points are inside $J_{(copy)}$). This update is performed using technique described in [35]. We will possibly have to update the index k of the best point in the dataset J and F_{old}.

iii. Perform operations described in point **A** & **B** of the parallel/second task algorithm above: "Copy $q_{(copy)}(x)$ from $q_k(x)$.
Copy $J_{(copy)}$ from $J = \{\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(N)}\}$".

In step **D.** of the parallel algorithm, we must find a point which increase substantially the quality of the local approximation $q_{(copy)}(x)$ of $f(x)$. In the following, the discovery of this better point is explained. The equation (3) is used. We will restate it here for clarity:

$$\begin{matrix}\textit{Interpolation Error}\\ \textit{of } q_{(copy)} \textit{ at point } y\end{matrix} = |q_{(copy)}(y) - f(y)| < \frac{M}{6}\sum_{j=1}^{N}|P_{j,(copy)}(y)|\|y - \boldsymbol{x}_{(j),(copy)}\|^3$$

where M and $P_{j,(copy)}$ have the same signification as for equation (3). Note also that we are working on a copy of $q_k(x)$, J and $P_j(x)$. In the remaining of the current section, we will drop the $_{(copy)}$ subscript for easier notation.

This equation has a special structure. The contribution to the interpolation error of the point $\boldsymbol{x}_{(j)}$ to be dropped is easily separable from the contribution of the other points of the dataset J, it is:

$$\textit{error due to } \boldsymbol{x}_{(j)} = \frac{1}{6}M|P_j(y)|\|y - \boldsymbol{x}_{(j)}\|^3 \tag{6}$$

If y is inside the ball of radius ρ around x_k (x_k is the best point found until now in the second/parallel task), then an upper bound of equation (6) can be found:

$$\begin{aligned}\frac{1}{6}M\max_y\{|P_j(y)|\|y - x_k\|^3 : \|y - x_k\| \leq \rho\}\\ \approx \frac{1}{6}M\|\boldsymbol{x}_{(j)} - x_k\|^3\max_d\{|P_j(x_k + d)| : \|d\| \leq \rho\}\end{aligned}$$

We are ignoring the dependence of the other Newton polynomials in the hope of finding a useful technique and cheap to implement. $\boldsymbol{x}_{(j)}$ is thus replaced in $J_{(copy)}$ by $x_k + d$ where d is the solution of the following problem:

$$\max_d\{|P_j(x_k + d)| : \|d\| \leq \rho\}$$

The algorithm used to solve this problem is described in [35].

5 Numerical Results

5.1 Results on one CPU

We will now compare CONDOR with UOBYQA [35], DFO [12,11], PDS [16], LANCELOT [7] and COBYLA [32] on a part of the Hock and Schittkowski test set [24]. The test functions and the starting points are extracted from SIF files obtained from CUTEr, a standard test problem database for non-linear optimization (see [23]). We are thus in perfect standard conditions. The tests problems are arbitrary and have been chosen by A.R.Conn, K. Scheinberg and Ph.L. Toint. to test their DFO algorithm. The performances of DFO are thus expected to be, at least, good. We list the number of function evaluations that each algorithm took to solve the problem. We also list the final function values that each algorithm achieved. We do not list the CPU time, since it is not relevant in our context. The "*" indicates that an algorithm terminated early because the limit on the number of iterations was reached. The default values for all the parameters of each algorithm is used. The stopping tolerance of DFO was set to 10^{-4}, for the other algorithms the tolerance was set to appropriate comparable default values. The comparison between the algorithms is based on the number of function evaluations needed to reach the SAME precision. For the most fair comparison with DFO, the stopping criteria (ρ_{end}) of CONDOR has been chosen so that CONDOR is always stopping with a little more precision on the result than DFO. This precision is some time insufficient to reach the true optima of the objective function. In particular, in the case of the problems GROWTHLS and HEART6LS, the CONDOR algorithm can find a better optimum after some more evaluations (for a smaller ρ_{end}). All algorithms were implemented in Fortran 77 in double precision except COBYLA which is implemented in Fortran 77 in single precision and CONDOR which is written in C++ (in double precision). The trust region minimization subproblem of the DFO algorithm is solved by NPSOL [20], a fortran 77 non-linear optimization package that uses an SQP approach. For CONDOR, the number in parenthesis indicates the number of function evaluation needed to reach the optimum without being assured that the value found is the real optimum of the function. For example, for the WATSON problem, we find the optimum after (580) evaluations. CONDOR still continues to sample the objective function, searching for a better point. It's loosing 87 evaluations in this search. The total number of evaluation (reported in the first column) is thus 580+87=667.

CONDOR and UOBYQA are both based on the same algorithm and have nearly the same behavior.

PDS stands for "*Parallel Direct Search*" [16]. The number of function evalu-

Name	Dim	Number of Function Evaluation						final function value					
		CONDOR		UOB.	DFO	PDS	LAN.	COB.	CONDOR	UOBYQA	DFO	PDS	LANCELOT	COBYLA
ROSENBR	2	82	(80)	87	81	2307	94	8000	2.0833e-08	4.8316e-08	1.9716e-07	1.2265e-07	5.3797e-13	4.6102e+04*
SNAIL	2	316	(313)	306	246	2563	715	8000	9.3109e-11	1.8656e-10	1.2661e-08	2.6057e-10	4.8608e+00	7.2914e+00*
SISSER	2	40	(40)	31	27	1795	33	46	8.7810e-07	2.5398e-07	1.2473e-06	9.3625e-20	1.3077e-08	1.1516e-20
CLIFF	2	145	(81)	127	75	3075	84	36	1.9978e-01	1.9978e-01	1.9979e-01	1.9979e-01	1.9979e-01	2.0099e-01
HAIRY	2	47	(47)	305	51	2563	357	3226	2.0000e+01	2.0000e+01	2.0000e+01	2.0000e+01	2.0000e+01	2.0000e+01
PFIT1LS	3	153	(144)	158	180	5124	216	8000	2.9262e-04	1.5208e-04	4.2637e-04	3.9727e-06	1.1969e+00	2.8891e-02*
HATFLDE	3	96	(89)	69	95	35844	66	8000	5.6338e-07	6.3861e-07	3.8660e-06	1.7398e-05	5.1207e-07	3.5668e-04*
SCHMVETT	3	32	(31)	39	53	2564	32	213	-3.0000e+00	3.0000e+00	-3.0000e+00	-3.0000e+00	-3.0000e+00	-3.0000e+00
GROWTHLS	3	104	(103)	114	243	2308	652	6529	1.2437e+01	1.2446e+01	1.2396e+01	1.2412e+01	1.0040e+00	1.2504e+01
GULF	3	170	(160)	207	411	75780	148	8000	2.6689e-09	3.8563e-08	1.4075e-03	3.9483e-02	7.0987e-17	6.1563e+00*
BROWNDEN	4	91	(87)	107	110	5381	281	540	8.5822e+04	8.5822e+04	8.5822e+04	8.5822e+04	8.5822e+04	8.5822e+04
EIGENALS	6	123	(118)	119	211	5895	35	1031	3.8746e-09	2.4623e-07	9.9164e-07	1.1905e-05	2.0612e-16	7.5428e-08
HEART6LS	6	346	(333)	441	1350	37383	6652	8000	4.3601e-01	4.0665e-01	4.3167e-01	1.6566e+00	4.1859e-01	4.1839e+00*
BIGGS6	6	284	(275)	370	1364	31239	802	8000	1.1913e-05	7.7292e-09	1.7195e-05	7.5488e-05	8.4384e-12	8.3687e-04*
HART6	6	64	(64)	64	119	6151	57	124	-3.3142e+00	-3.2605e+00	-3.3229e+00	-3.3229e+00	-3.3229e+00	-3.3229e+00
CRAGGLVY	10	545	(540)	710	1026	13323	77	1663	1.8871e+00	1.8865e+00	1.8866e+00	1.8866e+00	1.8866e+00	1.8866e+00
VARDIM	10	686	(446)	880	2061	33035	165	4115	8.7610e-13	1.1750e-11	2.6730e-07	8.5690e-05	1.8092e-26	4.2233e-06
MANCINO	10	184	(150)	143	276	11275	88	249	3.7528e-09	6.1401e-08	1.5268e-07	2.9906e-04	2.2874e-16	2.4312e-06
POWER	10	550	(494)	587	206	13067	187	368	9.5433e-07	2.0582e-07	2.6064e-06	1.6596e-13	8.0462e-09	6.8388e-18
MOREBV	10	110	(109)	113	476	75787	8000	8000	1.0100e-07	1.6821e-05	6.0560e-07	1.0465e-05	1.9367e-13	2.2882e-06*
BRYBND	10	505	(430)	418	528	128011	8000	8000	4.4280e-08	1.2695e-05	9.9818e-08	1.9679e-02	7.5942e-15	8.2470e-03*
BROWNAL	10	331	(243)	258	837	14603	66	103	4.6269e-09	4.1225e-08	9.2867e-07	1.3415e-03	1.1916e-11	9.3470e-09
DQDRTIC	10	201	(79)	80	403	74507	33	7223	2.0929e-18	1.1197e-20	1.6263e-20	1.1022e-04	1.6602e-23	3.8218e-06
WATSON	12	667	(580)	590	1919	76813	200	8000	7.9451e-07	2.1357e-05	4.3239e-05	2.5354e-05	2.0575e-07	7.3476e-04*
DIXMAANK	15	964	(961)	1384	1118	63504	2006	2006	1.0000e+00	1.0000e+00	1.0000e+00	1.0000e+00	1.0000e+00	1.0001e+00
FMINSURF	16	695	(615)	713	1210	21265	224	654	1.0000e+00	1.0000e+00	1.0000e+00	1.0000e+00	1.0000e+00	1.0000e+00
Total Number of Function Evaluation		7531	(6612)	8420	14676	> 20000								

Fig. 1. Comparative results between CONDOR, UOBYQA, DFO, PDS, LANCELOT and COBYLA on one CPU.

ations is high and so the method doesn't seem to be very attractive. On the other hand, these evaluations can be performed on several CPU's reducing considerably the computation time.

Lancelot [7] is a code for large scale optimization when the number of variable is $n > 10000$ and the objective function is easy to evaluate (less than $1ms.$). Its model is build using finite differences and BFGS update. This algorithm has not been design for the kind of application we are interested in and is thus performing accordingly.

COBYLA [32] stands for "*Constrained Optimization by Linear Approximation*" by Powell. It is, once again, a code designed for large scale optimization. It is a derivative free method, which uses linear polynomial interpolation of the objective function.

DFO [12,11] is an algorithm by A.R.Conn, K. Scheinberg and Ph.L. Toint. It has already been described in section 1. In CONDOR and in UOBYQA the *validity* of the model is checked using two equations:

$$\begin{array}{l} \textit{All the interpolation points must} \\ \textit{be close to the current point } \boldsymbol{x}_{(k)} \end{array} : \|\boldsymbol{x}_{(j)} - \boldsymbol{x}_{(k)}\| \leq 2\rho \qquad j = 1, \ldots, N \tag{7}$$

$$\begin{array}{l} \textit{Powell's} \\ \textit{heuristic} \end{array} : \frac{M}{6}\|\boldsymbol{x}_{(j)} - \boldsymbol{x}_{(k)}\|^3 \max_d\{|P_j(\boldsymbol{x}_{(k)} + d)| : \|d\| \leq \rho\} \leq \epsilon \quad j = 1, \ldots, N \tag{8}$$

using notation of section 3. See [35] to know how to compute ϵ. The first equation (7) is also used in DFO. The second equation (8) (which is similar to equation (3)) is NOT used in DFO. This last equation allows us to "keep far points" inside the model, still being assured that it is valid. It allows us to have a "full" polynomial of second degree for a "cheap price". The DFO algorithm cannot use equation 8 to check the validity of its model because the variable ϵ (which is computed in UOBYQA and in CONDOR as a by-product of the computation of the "Moré and Sorenson Trust Region Step") is not cheaply available. In DFO, the trust region step is calculated using an external tool: NPSOL [20]. ϵ is difficult to obtain and is not used.

UOBYQA and CONDOR are always using a full quadratic model. This enables us to compute Newton's steps. The Newton's steps have a proven quadratical convergence speed [15]. Unfortunately, some evaluations of the objective function are lost to build the quadratical model. So, we only obtain *near* quadratic speed of convergence. We have Q-superlinear convergence (see original paper of Powell [35]). (In fact the convergence speed is often directly proportional to the quality of the approximation H_k of the real Hessian matrix of $f(x)$). Usually, the price (in terms of number of function evaluations) to construct a good quadratical model is very high but using equation (8), UOBYQA and CONDOR are able to use very few function evaluations to

update the local quadratical model.

When the dimension of the search space is greater than 25, the time needed to start, building the first quadratic, is so important (N evaluations) that DFO may becomes attractive again. Especially, if you don't want the optimum of the function but only a small improvement in a small time. If several CPU's are available, then CONDOR once again imposes itself. The function evaluations needed to build the first quadratic are parallelized on all the CPU's without any loss of efficiency when the number of CPU increases (the maximum number of CPU is $N + 1$). This first construction phase has a great parallel efficiency, as opposed to the rest of the optimization algorithm where the efficiency becomes soon very low (with the number of CPU increasing). In contrast to CONDOR, the DFO algorithm has a very short initialization phase and a long research phase. This last phase can't be parallelized very well. Thus, when the number of CPU's is high, the most promising algorithm for parallelization is CONDOR. A parallel version of CONDOR has been implemented. Very encouraging experimental results on the parallel code are given in the next section.

When the local model is not convex, no second order convergence proof (see [10]) is available. It means that, when using a linear model, the optimization process can prematurely stop. This phenomenon *can* occur with DFO which uses from time to time a simple linear model. CONDOR is very robust and always converges to a local optimum (extensive numerical tests have been made [41]).

5.2 Parallel results

We are using the same test conditions as for the previous section (standard objective functions with standard starting points).

Since the objective function is assumed to be time-expensive to evaluate, we can neglect the time spent inside the optimizer and inside the network transmissions. To be able to make this last assumption (negligible network transmissions times), a wait loop of 1 second is embedded inside the code used to evaluate the objective function (only 1 second: to be in the worst case possible).

Table 2 indicates the number of function evaluations performed on the master CPU (to obtain approximatively the total number of function evaluations cumulated over the master and all the slaves, multiply the given number on the list by the number of CPU's). The CPU time is thus directly proportional to the numbers listed in columns 3 to 5 of the table 2.

Name	Dim	Number of Function Evaluations on the main node			final function value		
		1CPU	2CPU	3CPU	1 CPU	2 CPU	3 CPU
ROSENBR	2	82	81	70	2.0833e-08	5.5373e-09	3.0369e-07
SNAIL	2	314	284	272	9.3109e-11	4.4405e-13	6.4938e-09
SISSER	2	40	35	40	8.7810e-07	6.7290e-10	2.3222e-12
CLIFF	2	145	87	69	1.9978e-01	1.9978e-01	1.9978e-01
HAIRY	2	47	35	36	2.0000e+01	2.0000e+01	2.0000e+01
PFIT1LS	3	153	91	91	2.9262e-04	1.7976e-04	2.1033e-04
HATFLDE	3	96	83	70	5.6338e-07	1.0541e-06	3.2045e-06
SCHMVETT	3	32	17	17	-3.0000e+00	-3.0000e+00	-3.0000e+00
GROWTHLS	3	104	85	87	1.2437e+01	1.2456e+01	1.2430e+01
GULF	3	170	170	122	2.6689e-09	5.7432e-04	1.1712e-02
BROWNDEN	4	91	60	63	8.5822e+04	8.5826e+04	8.5822e+04
EIGENALS	6	123	77	71	3.8746e-09	1.1597e-07	1.5417e-07
HEART6LS	6	346	362	300	4.3601e-01	4.1667e-01	4.1806e-01
BIGGS6	6	284	232	245	1.1913e-05	1.7741e-06	4.0690e-07
HART6	6	64	31	17	-3.3142e+00	-3.3184e+00	-2.8911e+00
CRAGGLVY	10	545	408	339	1.8871e+00	1.8865e+00	1.8865e+00
VARDIM	10	686	417	374	8.7610e-13	3.2050e-12	1.9051e-11
MANCINO	10	184	79	69	3.7528e-09	9.7042e-09	3.4434e-08
POWER	10	550	294	223	9.5433e-07	3.9203e-07	4.7188e-07
MOREBV	10	110	52	43	1.0100e-07	8.0839e-08	9.8492e-08
BRYBND	10	505	298	198	4.4280e-08	3.0784e-08	1.7790e-08
BROWNAL	10	331	187	132	4.6269e-09	1.2322e-08	6.1906e-09
DQDRTIC	10	201	59	43	2.0929e-18	2.0728e-31	3.6499e-29
WATSON	12	667	339	213	7.9451e-07	1.1484e-05	1.4885e-04
DIXMAANK	15	964	414	410	1.0000e+00	1.0000e+00	1.0000e+00
FMINSURF	16	695	455	333	1.0000e+00	1.0000e+00	1.0000e+00
Total Number of Function Evaluation		7531	4732	3947			

Fig. 2. Improvement due to parallelism

Suppose a function evaluation takes 1 hour. The parallel/second process on the main computer has asked 59 minutes ago to a client to perform one such evaluation. We are at step 4(a)i of the main algorithm. We see that there are no new evaluation available from the client computers. Should we go directly to step 4(a)ii and use later this new information, or wait 1 minute? The response is clear: wait a little. This bad situation occurs very often in our test examples since every function evaluation takes exactly the same time (1 second). But what's the best strategy when the objective function is computing, randomly, from 40 to 80 minutes at each evaluation (this is for instance the case for objective functions which are calculated using CFD techniques)? The response is still to investigate. Currently, the implemented strategy is: never

wait. Despite, this simple strategy, the current algorithm gives already some non-negligible improvements.

5.3 Noisy optimization

We will assume that objective functions derived from CFD codes have usually a simple shape but are subject to high-frequency, low amplitude noise. This noise prevents us to use simple finite-differences gradient-based algorithms. Finite-difference is highly sensitive to the noise. Simple Finite-difference quasi-Newton algorithms behave so badly because of the noise, that most researchers choose to use optimization techniques based on GA,NN,... [42,13,**?**,**?**]. The poor performances of finite-differences gradient-based algorithms are either due to the difficulty in choosing finite-difference step sizes for such a rough function, or the often cited tendency of derivative-based methods to converge to a local optimum [4]. Gradient-based algorithms can still be applied but a clever way to retrieve the derivative information must be used. One such algorithm is DIRECT [21,25,5] which is using a technique called implicit filtering. This algorithm makes the same assumption about the noise (low amplitude, high frequency) and has been successful in many cases [5,6,39]. For example, this optimizer has been used to optimize the cost of fuel and/or electric power for the compressor stations in a gas pipeline network [6]. This is a two-design-variables optimization problem. You can see in the right of figure 5 a plot of the objective function. Notice the simple shape of the objective function and the small amplitude, high frequency noise. Another family of optimizers is based on interpolation techniques. DFO, UOBYQA and CONDOR belongs to this last family. DFO has been used to optimize (minimize) a measure of the vibration of a helicopter rotor blade [4]. This problem is part of the Boeing problems set [3]. The blade are characterized by 31 design variables. CONDOR will soon be used in industry on a daily basis to optimize the shape of the blade of a centrifugal impeller [29]. All these problems (gas pipeline, rotor blade and impeller blade) have an objective function based on CFD code and are both solved using gradient-based techniques. In particular, on the rotor blade design, a comparative study between DFO and other approaches like GA, NN,... has demonstrated the clear superiority of gradient-based techniques approach combined with interpolation techniques [4].

We will now illustrate the performances of CONDOR in two simple cases which have sensibly the same characteristics as the objective functions encountered in optimization based on CFD codes. The functions, the amplitude of the artificial noise applied to the objective functions (uniform noise distribution) and all the parameters of the tests are summarized in table 5.3. In this table "NFE" stands for *Number of Function Evaluations*. Each columns represents 50 runs of the optimizer.

Objective function	Rosenbrock	A simple quadratic: $\sum_{i=1}^{4}(x_i-2)$				
starting point	$(-1.2\ \ 1)^t$	$(0\ 0\ 0\ 0)^t$				
ρ_{start}	1					
ρ_{end}	1e-4					
average NFE	96.28 (88.02)	82.04 (53.6)	89.1 (62.20)	90.7 (64.56)	99.4 (66.84)	105.36 (68.46)
max NFE	105	117	116	113	129	124
min NFE	86	58	74	77	80	91
average best val	2.21e-5	6.5369e-7	3.8567e-6	8.42271e-5	8.3758e-4	1.2699e-2
noise	1e-4	1e-5	1e-4	1e-3	1e-2	1e-1

Fig. 3. Noisy optimization.

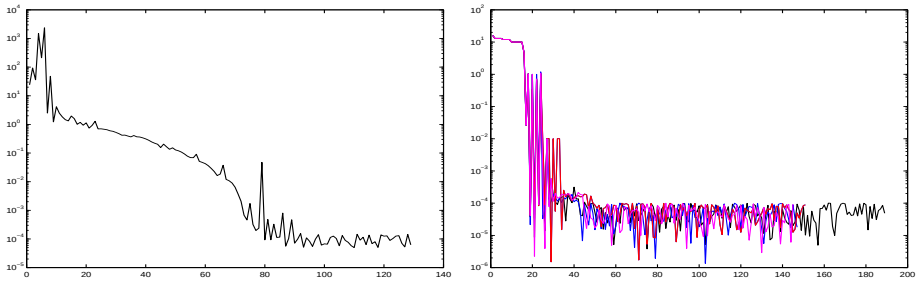

Fig. 4. On the left: A typical run for the optimization of the noisy Rosenbrock function. On the right:Four typical runs for the optimization of the simple noisy quadratic (noise=1e-4).

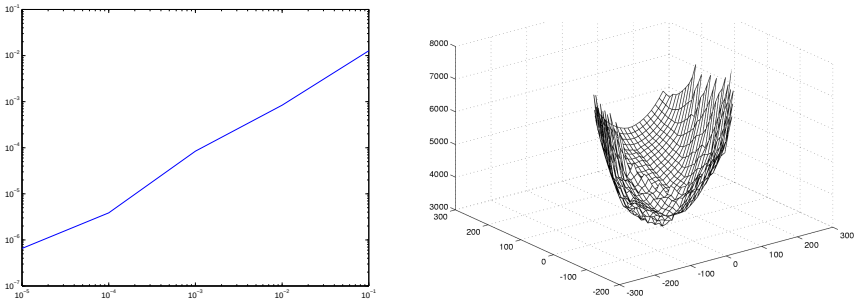

Fig. 5. On the left: The relation between the noise (X axis) and the average best value found by the optimizer (Y axis). On the right: Typical shape of objective function derived from CFD analysis.

A typical run for the optimization of the noisy Rosenbrock function is given in the left of figure 4. Four typical runs for the optimization of the simple noisy quadratic in four dimension are given in the right of figure 4. The noise on this four runs has an amplitude of 1e-4. In these conditions, CONDOR stops in average after 100 evaluations of the objective function but we can see in figure 4 that we usually already have found a quasi-optimum solution after only 45 evaluations.

As expected, there is a clear relationship between the noise applied on the objective function and the average best value found by the optimizer. This relationship is illustrated in the left of figure 4. From this figure and from the table 5.3 we can see the following: When you have a noise of 10^{n+2}, the difference between the best value of the objective function found by the optimizer AND the real value of the objective function at the optimum is around 10^n. In other words, in our case, if you apply a noise of 10^{-2}, you will get a final value of the objective function around 10^{-4}. Obviously, this strange result only holds for this simple objective function (the simple quadratic) and these particular testing conditions. Nevertheless, the robustness against noise is impressive.

If this result can be generalized, it will have a great impact in the field of CFD shape optimization. This simply means that if you want a gain of magnitude 10^n in the value of the objective function, you have to compute your objective function with a precision of at least 10^{n+2}. This gives you an estimate of the precision at which you have to calculate your objective function. Usually, the more precision, the longer the evaluations are running. We are always tempted to lower the precision to gain in time. If this strange result can be generalized, we will be able to adjust tightly the precision and we will thus gain a precious time.

6 Conclusions

Given the search space comprised between 2 and 20 and given some noise of small amplitude and high frequency on the objective function evaluation, among the best optimizer available are UOBYQA and its parallel, constrained extension CONDOR. When several CPU's are used, the experimental results tend to show that CONDOR becomes the fastest optimizer in its category(fastest in terms of number of function evaluations).

Some improvements are still possible:

- Add the possibility to start with a linear model, using a stable update inspired by [37].
- Use a better strategy for the parallel case (see end of section 5.2)
- Currently the trust region is a simple ball (this is linked to the L2-norm $\|s\|_2$ used in step 4(a)ii of the algorithm). It would be interesting to have a trust region which reflects the underlying geometry of the model and not give undeserved weight to certain directions (for example, using a H-norm) (see [9]). This improvement will have a small effect provided the variables have already been correctly normalized.

Some research can also be made in the field of kriging models (see [4]). These

models need very few "model improvement steps" to obtain a good *validity*. The *validity* of the approximation can also easily be checked.

The code of the optimizer is a complete C/C++ **stand-alone** package written in pure structural programmation style. There is no call to fortran, external, unavailable, copyrighted, expensive libraries. You can compile it under UNIX or Windows. The only library needed is the standard TCP/IP network transmission library based on sockets (only in the case of the parallel version of the code) which is available on almost every platform. You don't have to install any special library such as MPI or PVM to build the executables. The client on different platforms/OS'es can be mixed together to deliver a huge computing power. The full description of the algorithm code can be found in [40].

The code has been highly optimized (with extended use of memcpy function, special fast matrix manipulation, fast pointer arithmetics, and so on...). However, BLAS libraries [26] have not been used to allow a full Object-Oriented approach. Anyway, the dimension of the problems is rather low so BLAS is nearly un-useful. OO style programming allows a better comprehension of the code for the possible reader.

A small C++ SIF-file reader has also been implemented (to be able to use the problems coded in SIF from the CUTEr database, [34]). An AMPL interface [19] has also been implemented.

The fully stand-alone code is currently available at the homepage of the first author: http://iridia.ulb.ac.be/∼fvandenb/

References

[1] Aemdesign. URL: http://www.aemdesign.com/FSQPapplref.htm.

[2] Paul T. Boggs and Jon W. Tolle. Sequential Quadratic Programming. *Acta Numerica*, pages 1–000, 1996.

[3] Andrew J. Booker, A.R. Conn, J.E. Dennis Jr., Paul D. Frank, Michael Trosset, Virginia Torczon, and Michael W. Trosset. Global modeling for optimization: Boeing/ibm/rice collaborative project 1995 final report. Technical Report ISSTECH-95-032, Boeing Information Support Services, Research and technology, Box 3707, M/S 7L-68, Seattle, Washington 98124, December 1995.

[4] Andrew J. Booker, J.E. Dennis Jr., Paul D. Frank, David B. Serafini, Virginia Torczon, and Michael W. Trosset. Optimization using surrogate objectives on a helicopter test example. *Computational Methods in Optimal Design and Control*, pages 49–58, 1998.

[5] D. M. Bortz and C. T. Kelley. The Simplex Gradient and Noisy Optimization Problems. Technical Report CRSC-TR97-27, North Carolina State University, Department of Mathematics, Center for Research in Scientific Computation Box 8205, Raleigh, N. C. 27695-8205, September 1997.

[6] R. G. Carter, J. M. Gablonsky, A. Patrick, C. T. Kelley, and O. J. Eslinger. Algorithms for Noisy Problems in Gas Transmission Pipeline Optimization. *Optimization and Engineering*, 2:139–157, 2001.

[7] Andrew R. Conn, Nicholas I.M. Gould, and Philippe L. Toint. *LANCELOT: a Fortran package for large-scale non-linear optimization (Release A)*. Springer Verlag, HeidelBerg, Berlin, New York, springer series in computational mathematics edition, 1992.

[8] Andrew R. Conn, Nicholas I.M. Gould, and Philippe L. Toint. *Trust-region Methods*. SIAM Society for Industrial & Applied Mathematics, Englewood Cliffs, New Jersey, mps-siam series on optimization edition, 2000. Chapter 9: conditional model, pp. 307–323.

[9] Andrew R. Conn, Nicholas I.M. Gould, and Philippe L. Toint. *Trust-region Methods*. SIAM Society for Industrial & Applied Mathematics, Englewood Cliffs, New Jersey, mps-siam series on optimization edition, 2000. The ideal Trust Region: pp. 236–237.

[10] Andrew R. Conn, Nicholas I.M. Gould, and Philippe L. Toint. *Trust-region Methods*. SIAM Society for Industrial & Applied Mathematics, Englewood Cliffs, New Jersey, mps-siam series on optimization edition, 2000. Note on convex models, pp. 324–337.

[11] Andrew R. Conn, Nicholas I.M. Gould, and Philippe L. Toint. A Derivative Free Optimization Algorithm in Practice. Technical report, Department of Mathematics, University of Namur, Belgium, 98. Report No. 98/11.

[12] Andrew R. Conn, K. Scheinberg, and Philippe L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79:397–414, 1997.

[13] R. Cosentino, Z. Alsalihi, and R. Van Den Braembussche. Expert System for Radial Impeller Optimisation. In *Fourth European Conference on Turbomachinery, ATI-CST-039/01*, Florence,Italy, 2001.

[14] Carl De Boor and A. A Ron. On multivariate polynomial interpolation. *Constr. Approx.*, 6:287–302, 1990.

[15] J.E. Dennis Jr. and Robert B. Schnabel. *Numerical Methods for unconstrained Optimization and nonlinear Equations*. SIAM Society for Industrial & Applied Mathematics, Englewood Cliffs, New Jersey, classics in applied mathematics, 16 edition, 1996.

[16] J.E. Dennis Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM J. Optimization*, 1(4):448–474, 1991.

[17] J.E. Dennis Jr. and H.F. Welaker. Inaccurracy in quasi-Newton methods: local improvement theroems. *Mathematical programming Study*, 22:70–85, 1984.

[18] R. Fletcher. *Practical Methods of optimization.* a Wiley-Interscience publication, Great Britain, 1987.

[19] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press / Brooks/Cole Publishing Company, 2002.

[20] P.E. Gill, W. Murray, M.A. Saunders, and Wright M.H. Users's guide for npsol (version 4.0): A fortran package for non-linear programming. Technical report, Department of Operations Research, Stanford University, Stanford, CA94305, USA, 1986. Report SOL 862.

[21] P. Gilmore and C. T. Kelley. An implicit filtering algorithm for optimization of functions with many local minima. *SIAM Journal of Optimization*, 5:269–285, 1995.

[22] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, third edition.* Johns Hopkins University Press, Baltimore, USA, 1996.

[23] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEr (and SifDec), a Constrained and Unconstrained Testing Environment, revisited*. Technical report, Cerfacs, 2001. Report No. TR/PA/01/04.

[24] W. Hock and K. Schittkowski. Test Examples for Nonlinear Programming Codes. *Lecture Notes en Economics and Mathematical Systems*, 187, 1981.

[25] C. T. Kelley. *Iterative Methods for Optimization*, volume 18 of *Frontiers in Applied Mathematics.* 1999.

[26] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Soft.*, 5:308–323, 1979.

[27] J.J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM journal on scientif and statistical Computing*, 4(3):553–572, 1983.

[28] Eliane R. Panier and André L. Tits. On combining feasibility, Descent and Superlinear Convergence in Inequality Contrained Optimization. *Mathematical Programming*, 59:261–276, 1995.

[29] S. Pazzi, F. Martelli, V. Michelassi, Frank Vanden Berghen, and Hugues Bersini. Intelligent Performance CFD Optimisation of a Centrifugal Impeller. In *Fifth European Conference on Turbomachinery*, Prague, CZ, March 2003.

[30] Stphane Pierret and Ren Van den Braembussche. Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. *Journal of Turbomachinery*, ASME 98-GT-4, 1998. publication in the transactions of the ASME: " Journal of Turbomachinery ".

[31] C. Poloni. Multi Objective Optimisation Examples: Design of a Laminar Airfoil and of a Composite Rectangular Wing. *Genetic Algorithms for Optimisation in Aeronautics and Turbomachinery*, 2000. von Karman Institute for Fluid Dynamics.

[32] M.J.D. Powell. A direct search optimization method that models the objective and constraint functions by linar interpolation. In *Advances in Optimization and Numerical Analysis, Proceedings of the sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275, pages 51–67, Dordrecht, NL, 1994. Kluwer Academic Publishers.

[33] M.J.D. Powell. The use of band matrices for second derivative approximations in trust region algorithms. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1997. Report No. DAMTP1997/NA12.

[34] M.J.D. Powell. On the Lagrange function of quadratic models that are defined by interpolation. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 2000. Report No. DAMTP2000/10.

[35] M.J.D. Powell. UOBYQA: Unconstrained Optimization By Quadratic Approximation. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 2000. Report No. DAMTP2000/14.

[36] M.J.D. Powell. UOBYQA: Unconstrained Optimization By Quadratic Approximation. *Mathematical Programming*, B92:555–582, 2002.

[37] M.J.D. Powell. On updating the inverse of a KKT matrix. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 2004. Report No. DAMTP2004/01.

[38] Thomas Sauer and Yuan Xu. On multivariate lagrange interpolation. *Math. Comp.*, 64:1147–1170, 1995.

[39] D. E. Stoneking, G. L. Bilbro, R. J. Trew, P. Gilmore, and C. T. Kelley. Yield optimization Using a gaAs Process Simulator Coupled to a Physical Device Model. *IEEE Transactions on Microwave Theory and Techniques*, 40:1353–1363, 1992.

[40] Frank Vanden Berghen. Intermediate Report on the development of an optimization code for smooth, continuous objective functions when derivatives are not available. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium, 2003. Available at http://iridia.ulb.ac.be/∼fvandenb/work/dea/.

[41] Frank Vanden Berghen. Optimization algorithm for Non-Linear, Constrained, Derivative-free optimization of Continuous, High-computing-load Functions. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium, 2004. Available at http://iridia.ulb.ac.be/∼fvandenb/work/thesis/.

[42] J. F. Wanga, J. Periaux, and Sefriouib M. Parallel evolutionary algorithms for optimization problems in aerospace engineering. *Journal of Computational and Applied Mathematics*, 149, issue 1:155–169, December 2002.

[43] D. Winfield. *Function and functional optimization by interpolation in data tables.* PhD thesis, Harvard University, Cambridge, USA, 1969.

[44] D. Winfield. Function minimization by interpolation in a data table. *Journal of the Institute of Mathematics and its Applications*, 12:339–347, 1973.