Condor, a parallel, direct, constrained optimizer for high-computing-load, black-box objective functions

F. Vanden Berghen*

Université Libre de Bruxelles, IRIDIA laboratory, Brussels, 1050 Belgium

Abstract

This paper presents CONDOR, an algorithmic extension of Powell's UOBYQA algorithm ('Unconstrained Optimization BY Quadratical Approximation'). CONDOR stands for 'COnstrained, Non-linear, Direct, parallel Optimization using trust Region method for high-computing load noisy objective functions'. CONDOR opens new possibilities in the field of industrial shape optimization based on CFD (Computation Fluid Dynamic) codes or PDE (partial differential equations) solvers where the evaluations of the objective functions are very CPU intensive and very noisy. We also report comparative numerical results between UOBYQA, DFO and CONDOR. The experimental results are very encouraging and validate the approach. Finally, we present a new, free, easily comprehensible and fully stand-alone implementation in C++ of CONDOR.

Keywords: Non-linear optimization; Lagrange interpolation; Trust region method; Parallel optimization; Constrained optimization; Noisy optimization; High-computing-load optimization

1. Introduction

The aim of CONDOR is to find the minimum $x^* \in \Re^n$ of an objective function $f(x) \in \Re$ using the least number of function evaluations. It is assumed that the dominant computing cost of the optimization process is the time needed to evaluate the objective function $f(x)$. CONDOR is a *direct* optimization tool (i.e. the derivatives of $f(x)$ are not required). The only information needed about the objective function is a simple function (written in Fortran, C++, etc.) or an executable which can evaluate the objective function $f(x)$ at a given point x. The algorithm has been specially developed to be very robust against noise inside the evaluation of the objective function $f(x)$.

CONDOR has been successfully used during the European LTR project NNE5 1999 20130 named METHOD. The goal of this project is to optimize the shape of the blades inside a centrifugal compressor. Each evaluation of the objective function involves a long CFD computation (more than one hour).

The algorithms used inside CONDOR are part of the gradient-based optimization family. Usually, classical quasi-Newton gradient-based optimizers are seen as a poor choice because they are adversely affected by function inaccuracies [1]. This poor behaviour is mainly due to the fact that these kinds of optimizers need explicit gradient information, usually obtained by the classical finite difference equation:

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + h_i e_i) - f(x)}{h_i}$$

In the field of aerodynamic shape optimization, the objective functions are based on expensive simulation of CFD codes [2,3,4,5] or PDE solvers. For such applications, choosing an appropriate step size h_i for approximating the derivatives by finite differences is quite delicate: function evaluation is expensive and can be very noisy.

In opposition, direct optimization methods [6] are relatively insensitive to the noise. Unfortunately, they usually require a great number of function evaluations.

Gradient-based algorithms can still be applied but a clever way to retrieve the derivative information must be used. One such algorithm is DIRECT [7,8,9], which uses a technique called implicit filtering. CONDOR uses a technique called multivariate Lagrange interpolation, making evaluations of $f(x)$ in a way that reduces the influence of the noise. Since CONDOR is using the

* Tel.: +32 (479) 992768; E-mail: fvandenb@iridia.ulb.ac.be

classical gradient-based optimization theory (Restricted Newton's Steps on quadratical local model of the objective function) it can easily obtain fast convergence speed.

Some of the advantages of CONDOR over UOBYQA [10] are:

1. CONDOR can handle box, linear and non-linear constraints. The algorithms used are part of the active set family [11]. For in depth explanation of the constrained part of CONDOR, see [12].
2. CONDOR is able to make simultaneous evaluations of the objective function $f(x)$ using several CPUs in a cluster of computers to speed up the optimization process.

This paper is structured in the following way:

1. The introduction.
2. Basic description of the CONDOR algorithm.
3. Experimental results.
4. How to get the code and conclusions.

2. Basic description of the CONDOR algorithm

A very basic description of the CONDOR algorithm is:

1. Build an approximation (also called a 'local model') $Q_k(\delta)$ of the objective function $f(x)$ around the current point x_k. $Q_k(\delta)$ is a polynomial of degree two built using multivariate Lagrange interpolation technique [13,14].
2. Find the minimum δ_k of the local model at $Q_k(\delta_k)$. δ_k is the minimization step.

$$Q_k(\delta_k) = \min_{\delta} Q_k(\delta) \text{ subject to } \|\delta\| \leq \Delta_k \tag{1}$$

   Δ_k is called the 'trust region radius'. It is the extent to which we can 'trust' the local model $Q_k(\delta)$ of $f(x)$. Equation (1) is solved using Moré and Sorensen's algorithm [15]. Evaluate the objective function at the new point: $f(x_k + \delta_k) = y$.
3. Compute the 'degree of agreement' τ_k between $f(x)$ and $Q_k(\delta)$:

$$\tau_k = \frac{f(x_k) - f(x_k + \delta_k)}{Q_k(0) - Q_k(\delta_k)}$$

   Update x_k and Δ_k using the classical trust region update mechanism:

$\tau_k < 0.01$ (bad iteration)	$0.01 \leq \tau_k < 0.9$ (good iteration)	$0.9 \leq \tau_k$ (very good iteration)
$x_{k+1} = x_k$ $\Delta_{k+1} = 0.5\,\Delta_k$	$x_{k+1} = x_k + \delta_k$ $\Delta_{k+1} = \Delta_k$	$x_{k+1} = x_k + \delta_k$ $\Delta_{k+1} = 2\,\Delta_k$

   The main idea of step 3 is to only increase the trust region radius Δ_k when the local model $Q_k(\delta)$ reflects well the real function $f(x)$ (and gives us good directions).
4. Use y to build the new 'local model' $Q_{k+1}(\delta)$, possibly performing more function evaluations to ensure a non-degenerate local model. Increase k and go back to step 2.

CONDOR is inside the class of algorithm that is proven to be globally convergent to a local (maybe global) optimum. It uses conditional models as described in [16,17].

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

(a) Guide the search to the minimum of $f(x)$ (see evaluation performed at step 2). To guide the search, the information gathered until now and available in $Q_k(\delta)$ is *exploited.*

(b) Increase the quality of the local model $Q_k(\delta)$ (see evaluations performed at step 4). To avoid the degeneracy of $Q_k(\delta)$, the search space needs to be additionally *explored.*

Points (a) and (b) are antagonistic objectives. The main idea of the parallelization of the algorithm is to perform the *exploration* on distributed CPUs. Consequently, the algorithm will have better models $Q_k(\delta)$ of $f(x)$ at its disposal and choose better search directions, leading to a faster convergence. The parallel version of CONDOR uses a client-server approach. The server is moving inside the search space using the algorithm described above. Only the server is updating the current position x_k. The client computers are 'sampling' the search space around the current point x_k in order to always have inside the server non-degenerate $Q_k(\delta)$.

If the local model $Q_k(\delta)$ has been found degenerate, some evaluations are performed during the *exploration step* (step (b) or step 4). The test of degeneracy of $Q_k(\delta)$ involves the following heuristic due to Powell:

$$\frac{1}{6} M \|x_k - P_i\|^3 \max_d \{|L_i(x_k + d)| : \|d\| \leq \rho\} \leq \varepsilon$$

where M is an upper bound on the third derivative of $f(x)$. For more in-depth explanation of this formula, see [12].

The evaluations performed during the *exploration step* (step 4) and performed inside the client computers are placed in the search space such that:

1. The quality of the updated local model $Q_k(\delta)$ is maximized.
2. The influence of the noise that is present at each evaluation of $f(x)$ is reduced.

This implies an inner optimization process. The full procedure is described in [10,12]. The importance of the exploration steps is clearly stated in the excellent 'optimization by surrogate theory' [18]. Any algorithm that does not check the validity of its local model is subject to failure and early stops. Unfortunately some algorithms based on neural networks and genetic algorithms do not check if their local model is degenerate and thus exhibit slow and uncertain convergence (see, for example, [19]).

Let us assume that the current minimization step δ_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 Eq. (1) in the Reduced-Space of the Active Box and Linear Constraints (RSABLC) to obtain a new δ_k. A basis of the RSABLC is needed and is built using a QR factorization with pivoting [11,20]. If some non-linear constraints are active, an SQP algorithm [21] performed inside the RSABLC is used to compute the new δ_k. 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 λ_J (also called dual variable) associated with the constraint J. For in-depth explanation of the constrained step inside CONDOR, see [12].

3. Numerical results

We will now compare CONDOR with UOBYQA [19,22] and DFO [16,23] on a part of the Hock and Schittkowski test set [24]. More numerical results can be found in [12]. The test functions and the starting points are extracted from SIF files obtained from CUTEr, a standard test problem database for non-linear optimization [25]. We are thus in perfect standard conditions. The test problems are arbitrary and have been chosen by Conn et al. [23] to test their DFO algorithm. The performances of DFO are thus expected to be, at least, good. DFO has been designed for the same kind of problems as CONDOR: derivate-free, noisy optimization. DFO also uses a model built by interpolation. 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 default values for all the parameters of each algorithm are used. The stopping tolerance of DFO was set

Table 1
Numerical results for comparison between CONDOR 1CPU, 2 CPU, 3 CPU, and UOBYQA and DFO

Name	dim	Number of function evaluations on main node								Final function value				
		Condor						UOB.	DFO	CONDOR			UOBYQA	DFO
		1 CPU		2 CPU		3 CPU				1 CPU	2 CPU	3 CPU		
ROSENBR	2	82	(80)	81	(1.2%)	70	(14.6%)	87	81	2,0833E−04	5,5373E−05	3,0369E−03	4,8316E−04	1,9716E−03
SNAIL	2	316	(313)	284	(9.6%)	272	(13.4%)	306	246	9,3109E−07	4,4405E−09	6,4938E−05	1,8656E−06	1,2661E−04
SISSER	2	40	(40)	35	(12.5%)	40	(0.0%)	31	27	8,7810E−03	6,7290E−06	2,3222E−08	2,5398E−03	1,2473E−02
CLIFF	2	145	(81)	87	(40.0%)	69	(52.4%)	127	75	1,9978E + 03	1,9978E + 03	1,9978E + 03	1,9978E + 03	1,9979E + 03
HAIRY	2	47	(47)	35	(25.5%)	36	(23.4%)	305	51	2,0000E + 05	2,0000E + 05	2,0000E + 05	2,0000E + 05	2,0000E + 05
PFIT1LS	3	153	(144)	91	(40.5%)	91	(40.5%)	158	180	2,9262E + 00	1,7976E + 00	2,1033E + 00	1,5208E + 00	4,2637E + 00
HATFLDE	3	96	(89)	83	(13.5%)	70	(27.1%)	69	95	5,6338E−03	1,0541E−02	3,2045E−02	6,3861E−03	3,8660E−02
SCHMVETT	3	32	(31)	17	(46.9%)	17	(46.9%)	39	53	−3,0000E + 04	−3,0000E + 04	−3,0000E + 04	3,0000E + 04	−3,0000E + 04
GULF	3	170	(160)	170	(0.0%)	122	(28.2%)	207	411	2,6689E−05	5,7432E + 00	1,1712E + 02	3,8563E−04	1,4075E + 01
BROWNDEN	4	91	(87)	60	(34.1%)	63	(30.8%)	107	110	8,5822E + 08	8,5826E + 08	8,5822E + 08	8,5822E + 08	8,5822E + 08
EIGENALS	6	123	(118)	77	(37.4%)	71	(42.3%)	119	211	3,8746E−05	1,1597E−03	1,5417E−03	2,4623E−03	9,9164E−03
BIGGS6	6	284	(275)	232	(18.3%)	245	(13.7%)	370	1364	1,1913E−01	1,7741E−02	4,0690E−03	7,7292E−05	1,7195E−01
HART6	6	64	(64)	31	(51.6%)	17	(73.4%)	64	119	−3,3142E + 04	−3,3184E + 04	−2,8911E + 04	−3,2605E + 04	−3,3229E + 04
CRAGGLVY	10	545	(540)	408	(25.1%)	339	(37.8%)	710	1026	1,8871E + 04	1,8865E + 04	1,8865E + 04	1,8865E + 04	1,8866E + 04
VARDIM	10	686	(446)	417	(39.2%)	374	(45.5%)	880	2061	8,7610E−09	3,2050E−08	1,9051E−07	1,1750E−07	2,6730E−03
MANCINO	10	184	(150)	79	(57.1%)	69	(62.5%)	143	276	3,7528E−05	9,7042E−05	3,4434E−04	6,1401E−04	1,5268E−03
POWER	10	550	(494)	294	(46.6%)	223	(59.4%)	587	206	9,5433E−03	3,9203E−03	4,7188E−03	2,0582E−03	2,6064E−02
MOREBV	10	110	(109)	52	(52.7%)	43	(60.9%)	113	476	1,0100E−03	8,0839E−04	9,8492E−04	1,6821E−01	6,0560E−03
BRYBND	10	505	(430)	298	(41.0%)	198	(60.8%)	418	528	4,4280E−04	3,0784E−04	1,7790E−04	1,2695E−01	9,9818E−04
BROWNAL	10	331	(243)	187	(43.5%)	132	(60.1%)	258	837	4,6269E−05	1,2322E−04	6,1906E−05	4,1225E−04	9,2867E−03
DQDRTIC	10	201	(79)	59	(70.6%)	43	(78.6%)	80	403	2,0929E−14	2,0728E−27	3,6499E−25	1,1197E−16	1,6263E−16
WATSON	12	667	(580)	339	(49.2%)	213	(68.1%)	590	1919	7,9451E−03	1,1484E−01	1,4885E + 00	2,1357E−01	4,3239E−01
DIXMAANK	15	964	(961)	414	(57.0%)	410	(57.5%)	1384	1118	1,0000E + 04	1,0000E + 04	1,0000E + 04	1,0000E + 04	1,0000E + 04
FMINSURF	16	695	(615)	455	(34.5%)	333	(52.1%)	713	1210	1,0000E + 04	1,0000E + 04	1,0000E + 04	1,0000E + 04	1,0000E + 04
Total number of function evaluations		7531	6612	4732		3947		8420	14676					

to 10^{-4}. The comparison between the algorithms is based on the number of function evaluations needed to reach the SAME precision. For the fairest comparison with DFO, the stopping criterion of CONDOR has been chosen so that CONDOR always stops with a little more precision in the result than DFO. In particular, in some cases, the CONDOR algorithm can find a better optimum after a few more evaluations (for a stricter stopping criterion). Table 1 shows all the numerical results. The number in parentheses in column 4 of Table 1 indicates the number of function evaluations 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, but loses 87 evaluations in this search. The total number of evaluations (reported in the first column) is thus 580 + 87 = 667. Table 1 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 CPUs). The CPU time is thus directly proportional to the numbers listed in columns 3, 5 and 7 of the table. The percentage given in columns 6 and 8 indicates the gain in term of computing time compared to 1 CPU. We can observe that:

1. CONDOR is a good choice compared to DFO when the dimension of the search space is above 4.
2. When the dimension of the search space is low, there is no need to make many samples of $f(x)$ to obtain a good approximation $Q_k(\delta)$. Using many CPUs is thus interesting only when the dimension of the search space is at least 4.

For numerical results demonstrating the robustness of CONDOR against noise, see [12].

4. Conclusions

Given a search space dimension between 2 and 35, and given some noise of small amplitude and high frequency on the objective function evaluations, among the best optimizers available are UOBYQA and its parallel, constrained extension CONDOR. When several CPUs 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).

The code for CONDOR is a complete C/C++ *stand-alone*, multi-platform (windows/UNIX) package, and is currently freely available at the homepage of the author (http://iridia.ulb.ac.be/~fvandenb/). Condor does not use any external, unavailable, copyrighted, expensive libraries. 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 for almost every platform. Different platforms/operating systems can be mixed together to deliver a huge computing power. The full description of the algorithm can be found in [12].

References

[1] Dennis JE Jr, Welaker HF. Inaccuracy in quasi-Newton methods: local improvement theorems. Math Prog Study 1984;22:70–85.

[2] Cosentino R, Alsalihi Z, Van Den Braembussche R. Expert system for radial impeller optimisation. In: Proc of 4th European Conference on Turbomachinery, ATI-CST-039/01, Florence, Italy, 2001.

[3] Pierret S, Van den Braembussche R. Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. J Turbomachinery, 1998, ASME 98-GT-4.

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

[5] Pazzi S, Martelli F, Michelassi V, Vanden Berghen, F, Bersini, H. Intelligent performance CFD optimisation of a centrifugal impeller. In: Proc of 5th European Conference on Turbomachinery, Prague, March, 2003.

[6] Dennis JE Jr, Torczon V. Direct search methods on parallel machines. SIAM J Optimization 1991;1(4):448–474.

[7] Gilmore P, Kelley CT. An implicit filtering algorithm for optimization of functions with many local minima. SIAM J Optimization 1995;5:269–285.

[8] Kelley CT. Iterative Methods for Optimization. Frontiers in Applied Mathematics. vol. 18. Philadelphia: SIAM, 1999.

[9] Bortz DM, Kelley CT. The simplex gradient and noisy optimization problems. North Carolina State University, Department of Mathematics, CRSC-TR97–27, Center for Research in Scientific Computation, Raleigh, NC, 1997.

[10] Powell MJD. UOBYQA: Unconstrained optimization by quadratic approximation. Math Prog 2002;B92:555–582.

[11] Fletcher R. Practical Methods of Optimization. Chichester: J Wiley & Sons, 1987.

[12] Vanden Berghen, F. Optimization algorithm for non-linear, constrained, derivative-free optimization of continuous, high-computing-load functions, IRIDIA. Université Libre de Bruxelles, Belgium, 2004. http://iridia.ulb.ac.be/~fvandenb/work/thesis/

[13] De Boor C, Ron A. On multivariate polynomial interpolation. Constr Approx 1990;6:287–302.

[14] Sauer T, Xu Y. On multivariate Lagrange interpolation. Math Comp 1995;64:1147–1170.

[15] Moré JJ, Sorensen DC. Computing a trust region step. SIAM J Sci Stat Comput 1983;4:553–572.

[16] Conn R, Scheinberg K, Toint PhL. Recent progress in unconstrained nonlinear optimization without derivatives. Math Prog, 1997;79:397–414.

[17] Conn AR, Gould NIM, Toint PhL. Trust-Region Methods. Englewood Cliffs, NJ: SIAM Society for Industrial & Applied Mathematics, 2000, pp. 307–323.

[18] Booker AJ, Dennis JE Jr, Frank PD, Serafini DB, Torczon V, Trosset MW. A rigorous framework for optimization of expensive functions by surrogates. Struct Optimization 1999;17(1):1–13.

[19] Pierret S, Ploumhans P, Gallez X, Caro S. Turbofan noise reduction using optimisation method coupled to aeroacoustic simulations. In: Proc of Design Optimization International Conference, 31 March–2 April, 2004, Athens, Greece.

[20] Golub, GH, Van Loan CF. Matrix Computations, 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

[21] Boggs PT, Tolle JW. Sequential quadratic programming. Acta Num 1996;1–000.

[22] Powell MJD. UOBYQA: unconstrained optimization by quadratic approximation. Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 2000, Report No. DAMTP2000/14.

[23] Conn AR, Gould NIM, Toint PhL. A derivative free optimization algorithm in practice. Department of Mathematics, University of Namur, Belgium, 1998, Report No. 98/11.

[24] Hock W, Schittkowski K. Test Examples for Nonlinear Programming Codes. Berlin: Springer, 1981.

[25] Gould NIM, Orban D, Toint Ph L. CUTEr (and SifDec), a Constrained and Unconstrained Testing Environment, revisited. Cerfacs, 2001, Report No. TR/PA/01/04.