A Filter Inexact-Restoration Method for Nonlinear Programming

Cândida Elisa P. Silva* M. Teresa T. Monteiro†

January 5, 2009

Abstract

A new iterative algorithm based on the inexact-restoration (IR) approach combined with the filter strategy to solve nonlinear constrained optimization problems is presented. The high level algorithm is suggested by Gonzaga et al. [7] but not yet implemented - the internal algorithms are not proposed. The filter, a new concept introduced by Fletcher and Leyffer [3], replaces the merit function avoiding the penalty parameter estimation and the difficulties related to the nondifferentiability. In the IR approach two independent phases are performed in each iteration - the feasibility and the optimality phases. The line search filter is combined with the first one phase to generate a "more feasible" point and then it is used in the optimality phase to reach an "optimal" point.

Numerical experiences with a collection of AMPL problems and a performance comparison with IPOPT are provided.

Keywords: Filter Method, Inexact-Restoration, Line Search.

1 Introduction

The inexact-restoration method introduced by Martínez [9] and also used in [8] treats feasibility and optimality as two independent phases. Each iteration of the method proceeds in two phases. In the first phase, feasibility of the current iterate is improved (a "more feasible" point is computed with respect

*Management and Industrial School, Polytechnic Institute of Porto, Portugal candidasilva@eseig.ipp.pt

†University of Minho, Portugal tm@dps.uminho.pt

to the current point). In the second phase the objective function value is reduced (an "optimal" point is calculated) in an approximate feasible set. The point that results from the second phase is compared with the current point using a merit function that combines feasibility and optimality.

Recently, Fletcher and Leyffer [3] have proposed a filter method as an alternative to the merit functions. These authors presented a SQP algorithm with trust-region technique to solve nonlinear constrained optimization problems. The motivation given by these authors for the development of the filter method is to avoid the necessity to determine a suitable value of the penalty parameter in the merit function. In constrained optimization, the two competing aims, minimization of the objective function and the satisfaction of the constraints, are combined into a single minimization problem, using a merit function. In the filter strategy two separated aims are considered instead of a combination of both.

Wächter and Biegler proposed a line search filter method for nonlinear programming introducing second order correction steps [11, 12]. They proved that the proposed method does not suffer from Maratos effect, so that fast local convergence to second order sufficient local solutions is achieved. Their work, a primal-dual interior-point algorithm with filter line search method [13], is implemented in the IPOPT code.

This work presents an algorithm to solve nonlinear constrained optimization problems, combining the IR approach with the filter method. A similar idea was suggested by Gonzaga et al. [7] where no methods are specified for the IR phases. In each phase of the IR method a filter scheme with line search technique is used instead of a merit function.

This paper is organized as follows. The next section defines the problem to solve. The inexact-restoration approach is presented in Section 3. Section 4 presents some concepts related to the filter method in the trust region and line search context. The new algorithm with its internal algorithms specifications is introduced in Section 5 and some convergence topics are mentioned. Finally in Section 6, numerical experiences, comparison with IPOPT solver and conclusions are reported.

2 Problem definition

The general nonlinear constrained optimization problem is defined by

$$\begin{cases} \min & f(x) \\ \text{s.t.} & lb_x \leq x \leq ub_x \\ & lb_c \leq c(x) \leq ub_c, \end{cases} \tag{NLP}$$

where $f(x) : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function, $x \in \mathbb{R}^n$ and lb_x, $ub_x \in \mathbb{R}^n$ are the lower and the upper bounds of the variable x, respectively, and $c(x) : \mathbb{R}^n \to \mathbb{R}^m$ is a set of m general constraints whose lower and upper bounds are lb_c and ub_c, respectively. The set of all constraints defines the feasible region. Note that equality constraints are included in the above formulation by setting the lower bound equal to the upper bound for the relevant constraint. Likewise it is possible to include one-sided constraints by setting the lower bound to $-\infty$ or the upper bound to ∞, depending on which bound is required.

3 Inexact-Restoration approach (IR)

The point of view of IR approach is that feasibility is an important feature of the problem that must be controlled independently of optimality. So, the methods based on IR consider feasibility and optimality at different phases of a single iteration. A well known drawback of feasible methods is their inability to follow very curved domains, which causes that very short steps might be computed far from the solution. IR methodology tries to avoid this inconvenient using procedures that automatically decrease the tolerance of infeasibility as the solution is approximated. In this way, large steps on an enlarged feasible region are computed at the beginning of the process. An essential feature of this methodology is that one is free to choose different algorithms both for the feasibility and for the optimality phase, so that problem characteristics can be exploited. It is shown [8] that the use of the Lagrangian function in the optimality phase favors practical fast convergence near the solution.

4 Filter Method

In the constrained optimization problem two conflicting criteria are always present, the minimization of the objective function, $f(x)$, and the satisfaction of the constraints, $c(x)$. All the constraints defined in (NLP) will be handled like $c(x) \leq 0$ and the optimization problem is formulated in its minimization form. A merit function combines these aims into a single minimization problem. These two competing aims can be written as two minimization problems

$$\min_{x \in I\!R^n} f(x) \quad \text{and} \quad \min_{x \in I\!R^n} h(x), \tag{1}$$

where

$$h(x) := \left\|c^+(x)\right\|_1 := \sum_{j=1}^{m} c_j^+(x),$$

with $c_j^+(x) = \max(0, c_j(x))$ is the sum of constraints violation.

A filter is defined as a set of pairs $(f(x^k), h(x^k))$ so that no pair dominates any other. The dominance concept comes from multi-objective optimization and establishes that $(f(x^k), h(x^k))$ dominates $(f(x^{k+1}), h(x^{k+1}))$ if and only if both $f(x^k) \leq f(x^{k+1})$ and $h(x^k) \leq h(x^{k+1})$ are verified. The filter, in Figure 1, can be represented in $\mathbb{R}^2$ as a set of pairs (f, h), defining the forbidden and allowed regions as well as the envelope that defines the sufficient reduction imposed in the filter acceptance. When a point x^k is accepted by the filter, which means that it belongs to the allowed region, the corresponding pair $(f(x^k), h(x^k))$ is introduced in the filter. All the pairs dominated by it will be removed from the filter.

Fletcher and Leyffer [3] introduce the concept of filter in an SQP algorithm with trust-region technique. The filter SQP algorithm starts with an initial point, the solution of the current QP subproblem, which produces a trial step, and with it the new trial point is determined. If it is accepted by the filter, this will be the new point, otherwise the step will be rejected, the trust-region radius is reduced and a new QP subproblem must be solved again. With this algorithm the usual descent criterion in the penalty function is replaced by the requirement that the new point is accepted by the filter. A more feasible and/or optimal point is achieved by the adjustment of the trust-region along the iterations of the algorithm. The idea is to use the filter as a criterion for accepting or rejecting a step.

Antunes and Monteiro [1] presented a SQP algorithm based on Fletcher and Leyffer idea where the trust-region is replaced by the line search technique. The main difference is that instead of using the trust-region to determine a trial point, uses line search method. The combination of line search technique with the filter method determines the step length acceptance. First the search direction is computed from a given initial point, solution of the current QP subproblem, and with it a trial point $x^{k+1} = x^k + \alpha d^k$ is computed. Then this point will be checked by the filter and if it is accepted, it is introduced. Otherwise, α is divided by two and the trial point x^{k+1} is updated and checked again by the filter, without solving another QP subproblem. This technique is represented in Figure 2.

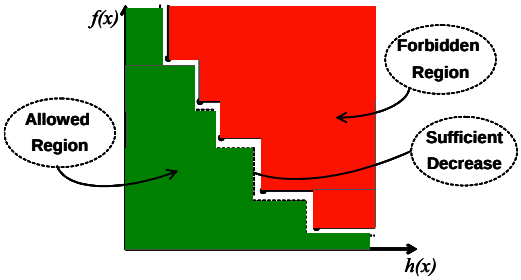

Figure 1: Filter with sufficient decrease

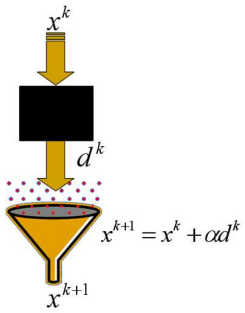

Figure 2: Line search scheme

5 Filter IR with Line Search

This section presents the developed algorithm, whose flowchart, without implementation details, is in Figure 4. It has two independent phases - the feasibility and the optimality phase. The goal in the first phase is, from a given point x^k, to reach a "more feasible" point z^k. In the optimality phase, an "optimal" point x^{k+1}, performed in z^k, is computed. Both phases have incorporated the filter method with line search technique. This procedure is represented in Figure 3, where $dfea$ and $dopt$ are the step in feasibility and optimality phase, respectively.

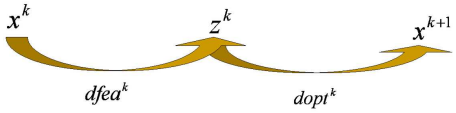

Figure 3: IR technique

5.1 Feasibility phase internal algorithm

The algorithm starts in the feasibility phase whose goal is to minimize the sum of all the constraints violation in the feasible set - if the point is feasible the algorithm will not perform this phase. From the initial trial point x^k the aim is to reach a more feasible point z^k. This point is obtained by solving the following LP subproblem:

$$\begin{aligned} \min_{d \in \mathbf{R}^n} \quad & \sum_{i \in J} A_i^k d \\ \text{s.t.} \quad & A_i^k d + c_i^k \leq 0, \qquad i \in J^{\perp} \end{aligned} \tag{LP}$$

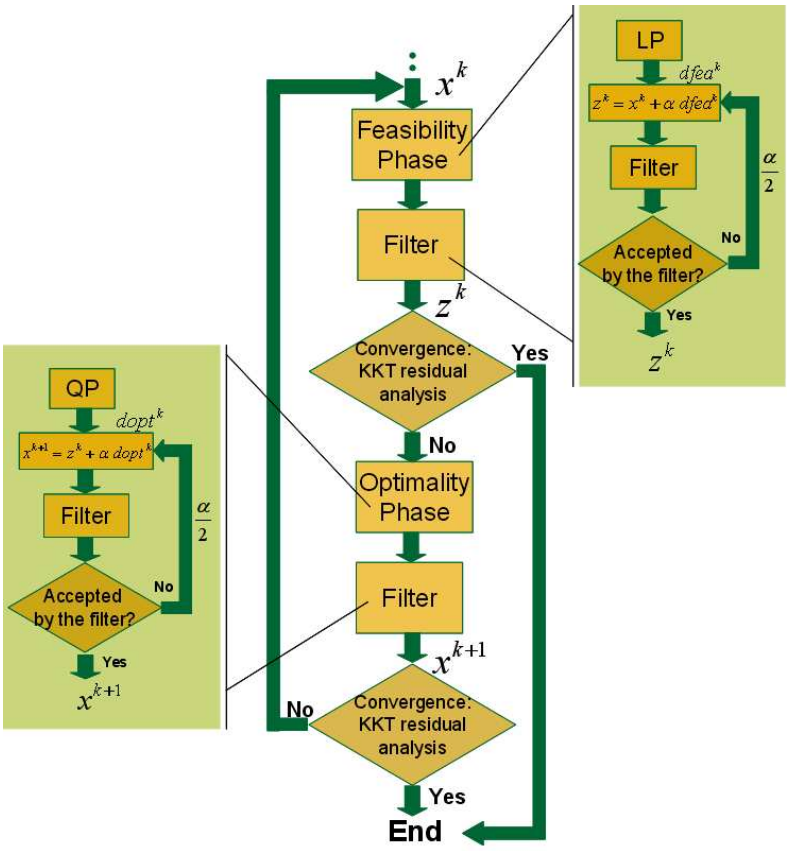

Figure 4: Filter IR flowchart

where $A^k = \nabla c(x^k)^T$ is the Jacobian matrix of the constraints and, J and $J^\perp$ are the sets of violated and satisfied constraints, respectively. The solution of this (LP) subproblem is the search direction $dfea^k$ (denoted by d in (LP)). The intermediate point $z^k = x^k + \alpha\ dfea^k$ is obtained by line search technique using the filter, where $\alpha \in \mathbb{R}$ is the step length.

The Filter IR management is quite different from the one used by Fletcher and Leyffer [3] and Antunes and Monteiro [1]. In the Filter IR a temporary pair $(f(x^k), h(x^k))$, corresponding to x^k, is used to evaluate the next trial point x^{k+1}.

The intermediate point z^k is accepted by the filter if the corresponding pair $(f(z^k), h(z^k))$ is not dominated by any other pair in the filter. The filter sufficient decrease acceptance condition is

$$h < \beta h(x^l) \quad \text{or} \quad f < f(x^l) - \gamma h \tag{2}$$

for all pairs $(f(x^l), h(x^l))$ in the filter, where β and γ are parameters such that $0 < \gamma < \beta < 1$, with β close to one and γ close to zero. Then it is also

verified if this pair causes a sufficient decrease in h when compared to the temporary pair $(f(x^k), h(x^k))$:

$$h(z^k) \leq (1-\gamma)h(x^k). \tag{3}$$

If this condition (3) is not satisfied then the intermediate point z^k is rejected and α is divided by two until the point is accepted or α is smaller than a tolerance. If the intermediate point z^k is a Kuhn-Tucker point and is feasible ($h(z^k) \leq \epsilon$, ϵ is a sufficiently small positive tolerance), then the algorithm terminates with success, otherwise the next phase will be performed.

5.2 Optimality phase internal algorithm

The goal of the optimality phase is to reduce the objective function from the z^k point. The corresponding QP subproblem is an approximation to the original problem (NLP):

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \quad & \frac{1}{2} d^T W^k d + d^T g^k \\ \text{s.t.} \quad & A^k d + c^k \leq 0, \end{aligned} \tag{QP}$$

where $g^k = \nabla f(x^k)$ is the gradient of the objective function, $A^k = \nabla c(x^k)^T$ is the Jacobian matrix of general constraints $c(x^k)$ and $W^k = \nabla^2 L(x^k, \lambda^k)$ is the Hessian matrix of the Lagrangian function. The solution of this QP subproblem is the search direction $dopt^k$ (denoted by d in (QP)). With this direction the new trial point $x^{k+1} = z^k + \alpha dopt^k$ is obtained. Then this point is tested by the filter, which is a similar procedure to the one described in previous phase and it is compared to the intermediate point z^k:

$$f(x^{k+1}) \leq (1-\gamma)f(z^k). \tag{4}$$

If this condition is verified x^{k+1} is the next point. Finally, if the following condition

$$f(x^{k+1}) \geq f(x^k) - min(h(x^k)^2, \omega) \tag{5}$$

is verified then the temporary pair $(f(x^k), h(x^k))$ is inserted into the filter and the dominated pairs are removed (ω in (5) is a small positive constant).

The algorithm terminates when a Kuhn-Tucker point is found and the feasibility is reached ($h(x^k) \leq \epsilon$). The corresponding algorithm can be written as follows:

ALGORITHM: ***Filter IR line search***

Given x^0 and λ^0, set $k = 0$; Let $(f(x^0), h(x^0))$ be a temporary pair;

REPEAT

Feasibility Phase:

Solve (LP) at x^k to obtain $dfea^k$;

Set $\alpha = 1$ and set $Accept = FALSE$;

REPEAT

Set $z^k = x^k + \alpha\ dfea^k$;

IF $(f(z^k), h(z^k))$ is accepted by the filter

AND $h(z^k) \leq (1-\gamma)h(x^k)$ **THEN**

Accept z^k; Set $Accept = TRUE$;

ELSE

Reject z^k;

Set $\alpha = \dfrac{\alpha}{2}$;

ENDIF;

UNTIL $Accept = TRUE$ **OR** $\alpha \leq TolAlpha$;

IF z^k is a KKT point and $h(z^k) \leq \epsilon$ **THEN** STOP with success.

ELSE

Optimality Phase:

Solve (QP) at z^k to obtain $dopt^k$;

Set $\alpha = 1$, $Accept = FALSE$;

REPEAT

Set $x^{k+1} = z^k + \alpha\ dopt^k$;

IF $(f(x^{k+1}), h(x^{k+1}))$ is accepted by the filter

AND $f(x^{k+1}) \leq (1-\gamma)f(z^k)$ **THEN**

Accept x^{k+1}; Set $Accept = TRUE$;

ELSE

Reject x^{k+1};

Set $\alpha = \dfrac{\alpha}{2}$;

ENDIF;

UNTIL $Accept = TRUE$ **OR** $\alpha \leq TolAlpha$;

ENDIF;

Filter update:

IF $(f(x^{k+1}), h(x^{k+1}))$ is accepted by the filter

AND $f(x^{k+1}) \geq f(x^k) - min(h(x^k)^2, \omega)$ **THEN**

Add temporary pair $(f(x^k), h(x^k))$ to the filter;

Remove points dominated by $(f(x^k), h(x^k))$ from the filter;

ENDIF;

Set $k = k + 1$, $x^k = x^{k+1}$;

UNTIL x^k is a KKT point and $h(x^k) \leq \epsilon$.

5.3 Some implementation details

In the implementation of the algorithm some interesting problems occur related to the Lagrange multipliers estimation and to the formulation of the (LP) subproblem. In order to solve the subproblems (LP) and (QP) of the feasibility and optimality phases, respectively, and also to calculate the stop criterium, it is necessary the Lagrangian multipliers estimation. The (LP) and (QP) subproblems are solved by LSSOL subroutine [5] from the NPSOL solver [6]. The Lagrange multiplier vector is estimated using the corresponding output of the LSSOL subroutine (resolution of the (QP)). For its initialization, several initial values were tested being the unitary vector the most suitable. Other decision is related to the formulation of the (LP) subproblem. The objective function of this subproblem considers all the violated constraints of the original problem. The set of the constraints is composed by all the satisfied constraints of the original problem. The LSSOL subroutine needs at least one constraint to be solved successfully. When all the constraints of the original problem are violated we decided to include the constraint that has less violation into the set of constraints. This constraint is not considered into the objective function. In this way, LSSOL subroutine is successfully solved.

5.4 Convergence Assumptions

The global convergence was studied by Gonzaga et al. [7], where it is proved that this method is independent of the internal algorithms used in each iteration, since these algorithms satisfy some requirements in their efficiency. It is shown that, under some assumptions, for a filter with a minimal size, the algorithm generates a stationary accumulation point and that for bigger filters, all the accumulation points are stationary.

The general hypotheses are:

H1: x^k and z^k remain in a convex compact domain $X \subset \mathbb{R}^n$.

H2: The objective and constraints functions ($f(x)$ and $c(x)$) are Lipschitz continuously differentiable in an open set containing X.

H3: All feasible accumulation points $\overline{x} \in X$ of x^k satisfy the Mangasarian-Fromovitz (M-F) constraints qualification, namely, the gradients of equality constraints are linearly independent, and there exists a direction $d \in \mathbb{R}^n$ such that $A_E(\overline{x})d = 0$ and $A_{\overline{I}}(\overline{x})d < 0$, where $\overline{I} = \{i \in I \,|\, c_i(\overline{x}) = 0\}$ ($A_E(.)$ and $A_{\overline{I}}(.)$ are the Jacobian matrix of equality constraints and active inequality constraints, respectively).

There are two more assumptions related to the internal algorithms, **H4** and **H5**, concerning to the feasibility phase and optimality phase algorithms, respectively.

H4: At all iterations $k \in \mathbb{N}$, the feasibility step must satisfy $h(z^k) < (1-\alpha)h(x^k)$ and z^k can not belong to the forbidden region.

H5: Given a feasible non-stationary point $\overline{x} \in X$, there exists a neighborhood V of $\overline{x}$ such that for any iterate $x^k \in V$, $f_0(x^{k+1}) \leq f_0(z^k)$ and x^{k+1} can not belong to the forbidden region.

6 Numerical experiences

This section presents the numerical experiences performed to test the algorithm. The computational experiences were made on a *centrino* with 504MB of RAM. The algorithm was implemented in C language for Windows operating system.

The 235 problems tested are in AMPL language [4], so the program was interfaced with this modelling language to read the problems automatically.

The first computational experience uses the IR algorithm to test all the problems in order to evaluate the algorithm robustness. These results are reported in the Tables 1, 2 and 3. The tables show the problem name, its dimension - n is the number of variables and m is the number of general constraints -, $\#it$ is the iteration counts, $\#f$ and $\#g$ are the function and gradient evaluation counts, respectively. The *note* is dedicated to remarks. The maximum iterations allowed was fixed in 1000. Two relevant remarks were found - some problems, denoted by a), suffer from Marato's effect, which means that the solution is obtained in few iterations but the algorithm does not converge, and in two problems the failure is related to the existence of a stationary point x with $h(x) > \epsilon$. The last remark had been already mentioned by Gonzaga et al. [7] in the convergence proof.

Table 1: General numerical results of the Filter IR algorithm

Problem	n	m	#it	#f	#g	feasibility	note
alsotame	2	1	3	31	6	Y	
biggsc4	4	7	2	4	3	Y	
bqp1var	1	0	1	1	1	N	
branin	2	0	5	26	7	Y	
camel6	2	0	8	9	9	Y	
cantilvr	5	1	1000	19962	1000	N	
cb2	3	3	1000	19981	1000	N	
cb3	3	3	1000	19981	1000	N	
chaconn1	3	3	1000	19962	1000	N	
chaconn2	3	3	1000	19962	1000	N	
chi	2	0	5	8	6	Y	
congigmz	3	5	16	243	16	N	
dembo7	16	20	1000	20013	1010	N	a)
dipigri	7	4	1000	19941	1000	N	a)
dixmaana	15	1	1	1	1	N	
dixmaanc	15	1	1	5	1	N	
dixmaane	15	1	1	5	1	N	
dixmaanf	15	1	1	5	1	N	
dixmaang	15	1	1	5	1	N	
dixmaanh	15	1	1	5	1	N	
dixmaani	15	1	1	5	1	N	
dixmaanj	15	1	1	6	1	N	
dixmaank	15	1	1	6	1	N	
dixmaanl	15	1	1	5	1	N	
engval1	2	1	15	15	15	N	
engval4	5	1	9	47	9	N	
explin	120	1	20	58	22	Y	
expquad	120	1	1	1	1	N	
fermat_socp_eps	5	3	1000	20000	1000	N	
fir_convex	11	243	1000	20000	1000	N	a)
fir_exp	12	244	1	2	2	Y	
fir_linear	11	243	1	2	2	Y	
fir_socp	12	244	4	26	6	Y	
fletcher	4	4	1000	39980	1999	N	
gigomez2	3	3	1000	19981	1000	N	a)
gigomez3	3	3	1000	19981	1000	N	a)
goffin	51	50	3	41	3	N	
harkerp2	100	1	18	18	18	N	
hatfldb	4	0	11	128	13	Y	
hatfldc	4	0	12	13	12	Y	
hatfldh	4	7	8	256	9	Y	
himmelp1	2	0	6	26	6	Y	
himmelp2	2	1	6	7	6	Y	
himmelp3	2	2	4	4	4	N	
himmelp4	2	3	3	22	3	Y	
himmelp5	2	3	3	3	3	Y	
himmelp6	2	4	4	5	4	Y	
hs001	2	0	24	33	25	N	
hs002	2	0	1000	19887	1001	N	a)
hs003	2	0	1	1	2	N	
hs004	2	0	2	1	3	N	
hs005	2	0	4	25	5	Y	
hs011	2	1	1000	16256	1000	N	
hs012	2	1	1000	17966	1000	N	
hs015	2	2	1	1	2	Y	
hs016	2	2	4	5	5	Y	
hs017	2	2	20	138	21	Y	
hs018	2	2	15	15	3	Y	
hs019	2	2	2	19	1	N	
hs020	2	3	5	6	6	Y	
hs021	2	1	1	2	2	Y	
hs022	2	2	9	126	10	N	
hs023	2	5	13	51	14	Y	
hs024	2	2	3	4	4	Y	
hs025	3	0	25	45	26	Y	
hs029	3	1	1000	1	1001	N	
hs030	3	1	2	12	3	Y	
hs031	3	1	7	8	8	Y	
hs033	3	2	7	8	8	Y	
hs034	3	2	1000	38742	1001	N	
hs035	3	1	1	1	2	N	
hs036	3	1	2	1	3	N	
hs037	3	1	1000	10919	1001	N	
hs038	4	0	23	27	24	Y	
hs043	4	3	1000	16953	1001	N	
hs044	4	6	6	8	7	N	
hs045	5	0	3	2	4	N	
hs059	2	3	7	8	8	Y	

Table 2: General numerical results of the Filter IR algorithm

Problem	n	m	#it	#f	#g	feasibility	note
hs064	3	1	1000	19810	1001	N	
hs066	3	2	1000	19796	1001	N	a)
hs072	4	3	1	1	2	N	
hs076	4	3	1	1	2	N	
hs083	5	3	4	11	5	N	
hs084	5	3	3	4	4	N	
hs086	5	6	3	4	4	Y	
hs089	3	1	1000	19890	1000	N	a)
hs095	6	4	2	3	2	Y	
hs096	6	4	4	45	6	Y	
hs097	6	3	9	27	9	Y	
hs098	6	3	3	5	3	Y	
hs100	7	4	1000	19941	1000	N	a)
hs100mod	7	4	1000	19951	1000	N	a)
hs102	7	6	25	203	49	N	
hs103	7	6	17	129	33	N	
hs104	8	6	1000	29253	1000	N	
hs106	8	6	630	11220	929	Y	
hs116	9	13	1000	38335	1965	N	
hs117	15	5	11	11	11	N	
hs118	15	17	1	2	2	Y	
hs15	2	2	3	21	4	N	
hs21mod	7	2	8	10	8	N	
hs23	2	5	13	51	13	Y	
hs268	5	5	1	1	1	N	
hs35	3	1	1	1	2	Y	
hs35mod	2	1	1	1	1	N	
hs3mod	2	0	1	1	1	N	
hs44	4	6	6	9	7	Y	
hs44new	4	5	3	4	3	Y	
hs5	2	0	3	23	4	Y	
hs64	3	1	1000	19819	1000	N	a)
hubfit	2	1	1	1	1	N	
humps	2	1	1	1	1	N	
kowalik	4	0	11	33	12	Y	
levy3	2	0	4	10	5	Y	
liarwhd	36	1	19	19	19	N	
liswet10	103	100	1	1	1	N	
liswet11	103	100	1	1	1	N	
liswet12	103	100	1	1	1	N	
liswet2	103	100	1	1	1	N	
liswet3	103	100	1	1	1	N	
liswet4	103	100	1	1	1	N	
liswet5	103	100	1	1	1	N	
liswet6	103	100	1	1	1	N	
liswet7	103	100	1	1	1	N	
liswet8	103	100	1	1	1	N	
liswet9	103	100	1	1	1	N	
logros	2	1	2	6	2	N	
lootsma	3	2	10	145	12	Y	
lsqfit	2	1	1	1	1	N	
madsen	3	6	1000	19924	1000	N	a)
makela1	3	2	1000	1	1000	N	a)
makela3	21	20	23	284	23	N	
makela4	21	40	2	21	2	N	
matrix2	6	2	14	38	14	N	
mifflin1	3	2	1000	17966	1000	N	
mifflin2	3	2	3	22	3	N	
minmaxrb	3	4	2	21	2	N	
nondquad	100	1	18	18	19	N	
osborne1	5	0	17	78	18	Y	
oslbqp	8	0	1	21	2	Y	
pfit1	3	1	1	1	1	N	
pfit1ls	3	1	1	1	1	N	
pfit2	3	1	1	1	1	N	
pfit2ls	3	1	1	1	1	N	
pfit3	3	1	1	1	1	N	
pfit3ls	3	1	1	1	1	N	
pfit4	3	1	1	1	1	N	
pfit4ls	3	1	1	1	1	N	
polak1	3	2	10	104	10	N	
polak3	12	10	1000	19905	1000	N	a)
powell	4	0	16	19	17	Y	
powell20	10	10	1	2	2	Y	
power	10	1	15	15	15	N	
price	2	0	6	13	7	Y	
pspdoc	4	0	5	27	5	N	
qrtquad	12	1	15	41	17	Y	
qudlin	12	1	42	3	2	N	

Table 3: General numerical results of the Filter IR algorithm

Problem	n	m	#it	#f	#g	feasibility	note
rosenmmx	5	4	1000	17974	1000	N	
s215	2	1	3	21	4	N	
s218	2	1	14	14	14	N	
s221	2	1	19	19	20	N	
s222	2	1	61	90	82	Y	
s223	2	2	6	10	6	Y	
s224	2	2	2	79	4	Y	
s225	2	5	8	26	9	N	
s226	2	2	2	3	3	Y	
s227	2	2	7	101	7	N	
s229	2	0	19	28	20	Y	
s230	2	2	1000	19963	1001	N	
s231	2	2	21	25	22	N	
s232	2	2	2	2	3	N	
s233	2	1	7	7	8	N	
s234	2	1	13	17	14	Y	
s236	2	2	2	2	3	Y	
s237	2	3	2	21	3	Y	
s238	2	3	6	13	7	Y	
s239	2	1	2	22	3	Y	
s242	3	0	2	2	3	N	
s244	3	0	9	31	10	Y	
s249	3	0	8	7	8	N	
s250	3	1	2	1	3	N	
s251	3	1	1000	10919	1001	N	
s253	3	1	3	3	4	N	
s257	4	0	10	11	10	N	
s259	4	0	9	16	9	N	
s268	5	5	1	1	2	N	
s270	5	1	12	14	13	N	
s277	4	4	1000	20000	1000	N	
s278	6	6	1000	20000	1000	N	
s279	8	8	1000	20000	1000	N	
s280	10	10	1000	20000	1000	N	
s307	2	0	10	48	11	N	
s315	2	3	1000	17985	1000	N	
s324	2	2	1000	19962	1000	N	a)
s326	2	2	412	539	413	Y	
s328	2	0	5	26	6	Y	
s329	2	3	120	3452	239	N	
s331	2	1	24	44	25	Y	
s332	2	1	1000	6643	1525	N	
s337	3	1	6	45	7	N	
s341	3	1	2	4	3	Y	
s342	3	1	3	6	4	Y	
s343	3	2	1	2	2	Y	
s346	3	2	1	2	2	Y	
s354	4	1	1000	19848	1000	N	
s357	4	35	6	8	7	Y	
s358	5	0	781	9281	782	Y	
s359	5	14	1	1	2 0	N	
s360	5	2	1000	2000	1000	N	
s361	5	6	2	2	3 0	N	
s365mod	7	4	1000	61	1000	N	
s366	7	14	1000	19875	1010	N	
s368	8	0	2	22	3	Y	
s372	9	12	1000	19981	1000	N	a)
schwefel	5	0	9	10	10	Y	
shekel	4	0	13	57	14	Y	
simbqp	2	0	1	1	2	N	
simpllpa	2	2	2	20	2	N	
simpllpb	2	3	2	20	2	N	
sineval	2	1	13	25	13	N	
sisser	2	1	1	7	2	N	
snake	2	2	1	1	1	N	
stancim	3	3	1000	19981	1000	N	a)
steiner_nonconvex	33	17	1000	19728	1000	N	$h \neq 0$
steiner_socp_eps	33	17	1000	20000	1000	N	a)
steiner_socp_vareps	33	17	1000	11111	1000	N	a)
tf12	3	100	1000	20000	1000	N	a)
tre	2	0	4	5	5	N	
vardim	10	1	25	25	26	N	
weapon	100	12	11	12	12	Y	
womflet	3	3	1	1	2	N	
zecevic2	2	2	1	1	2	Y	
zecevic3	2	2	1000	2500	1000	Y	$h \neq 0$
zecevic4	2	2	3	4	4	Y	
zy2	3	1	5	8	6	Y	

To evaluate the relevance of the feasibility phase, two versions of the algorithm are tested - the first one, with feasibility and optimality phases and the second with optimality phase only. The results are in Tables 4 and 5. These tables present two different columns - f, the objective function value at the solution found and SS to indicate if the same solution for both versions is reached (Y - yes and N - no).

Table 4: Comparative analysis - with and without feasibility phase

	With feasibility phase	Without feasibility phase			
Problem	f	#it	#f	f	SS
alsotame	0.082085	1000	19962	9.807.872	N
biggsc4	-24.375	3	3	-24.375	Y
branin	0.397887	5	26	0.397888	Y
camel6	-0.215464	7	26	210.425	N
chi	-19.248.788	5	43	279.324.672	N
explin	2.762.383	16	26	3.929.659	N
fir_exp	1.374.301.869	1000	19981	1.046.488	N
fir_linear	0.045342	1000	20000	0.045342	N
fir_socp	10.375	1000	20000	1.046.488	N
hatfldb	0.005573	22	59	0.005573	Y
hatfldc	0	1000	0	14.106.20	N
hatfldh	-24.375	1000	4493	-24.5	N
himmelp1	-62.053.869	8	85	81.177.341	N
himmelp2	-62.053.869	8	46	-8.198.032	N
himmelp4	-59.013.124	3	2	-58.013.124	Y
himmelp5	-59.013.124	8	7	-58.013.124	Y
himmelp6	-59.013.124	38	206	-8.198.032	N
hs005	122.837	3	22	222.837	Y
hs015	306.5	4	41	306.5	Y
hs016	23.144.661	5	24	23.144.661	Y
hs017	1.000.002	12	69	1	Y
hs018	5	1000	19962	5	Y
hs020	4.019.873	7	50	4.019.873	Y
hs021	-99.96	1	1	-99.96	Y
hs023	2	1000	19846	9.472.136	N
hs024	-1	2	21	0	N
hs025	0	25	44	0	Y
hs030	1	2	2	1	Y
hs031	6.000.001	6	45	6	Y
hs033	-4.585.786	1000	19772	2	N
hs038	0	19	19	0	Y
hs059	-6.749.505	1000	12862	-7.840.675	N
hs086	-32.348.679	3	3	-32.348.679	N
hs095	974.478	1000	20000	0.015620	N
hs096	0.461579	1000	19926	0.015620	N
hs097	461.629	1000	19981	3.135.809	N
hs098	908.118	1000	19981	3.135.809	N
hs106	2.058.916.567	1000	19887	6.745.848.611	N
hs118	66.482.045	1	1	66.482.045	N
hs23	2	1000	19910	9.472.136	N
hs35	0.111111	1	1	0.111112	Y
hs44	-15	7	12	-14	Y
hs44new	-13	6	8	-15	N
hs5	122.837	4	44	306.5	N
kowalik	0.000307	11	13	0.000308	Y
levy3	-23.243.044	2	1	-11.178.666	N
lootsma	1.414.214	1000	19753	8	N
osborne1	0.024518	3	11	1.106.036	N
oslbqp	6.25	1	1	7.25	Y
powell	0	19	19	0	N
powell20	578.125	1000	20000	588.125	Y
price	0	50	73	0	N
qrtquad	0.000071	1000	19830	0	N

Table 5: Comparative analysis - with and without feasibility phase

	With feasibility phase	Without feasibility phase			
Problem	f	#it	#f	f	SS
s222	-1.5	1000	17983	-1.662.003	N
s223	0	1000	17983	-1.662.003	N
s224	-304	2	39	-304	N
s226	0	1000	17925	-3.808.642	N
s229	0	21	28	0	Y
s234	-0.8	23	30	0	N
s236	-58.903.436	88	531	-8.197.517	Y
s237	-58.903.436	9	8	-58.903.436	N
s238	-58.903.436	1000	12936	-8.221.491	N
s239	-58.903.436	88	531	-8.197.517	N
s244	0	10	14	0	Y
s270	-1	9	9	0	Y
s326	-79.807.823	1000	13962	-8.112.289	N
s328	1.744.152	11	49	1.744.152	Y
s331	4.258.385	11	49	1.744.152	N
s341	0	1000	14972	-2.296.481	N
s342	0	1000	140	0	N
s343	-0.000000	5	5	-5.684.783	N
s346	-0.000000	5	5	-5.684.783	N
s357	0.358457	6	7	0.358457	Y
s358	0.000055	21	79	0.000055	Y
s368	0	2	21	0	Y
schwefel	0	9	9	0	Y
shekel	-2.630.472	8	19	-268.286	N
tre	0	7	8	0	N
weapon	-173.556.958	9	9	-173.556.958	Y
zecevic2	-4.125	1	1	-3.125	Y
zecevic3	97.306.951	1000	19887	9.730.945	N
zecevic4	7.557.508	4	4	8.557.508	Y
zy2	2	7	25	3	Y

Without further examination the results of this code seem very encouraging, but measuring and comparing software is a very difficult task. Dolan and Moré [2] present a tool to analyse the relative performance of the optimization codes with respect to a specific metric (number of iterations, CPU time).

For instances, if the metric is the number of iterations, for every solver s and every problem p, the ratio of the number of iterations of solver s on problem p over the fastest solve for problem p is computed and the base 2 logarithm is taken,

$$log_2 \left(\frac{\#iter(s,p)}{best\_iter(p)} \right). \tag{6}$$

By sorting these ratios in ascending order for every solver, the resulting plots can be interpreted as the probability that a given solver solves a problem within a certain multiple of the fastest solver. An easy interpretation of this graphic is that for any given metric a solver is the best when its graphic is tending faster to 1. This comparison is very interesting when the test set has a large number of problems.

This tool is used to compare the role of the feasibility phase, that is, Filter IR with and without feasibility phase. The performance metric considered is the number of iterations. The graphic of performance profiles are presented in Figure 5 and a *log* scale is used.

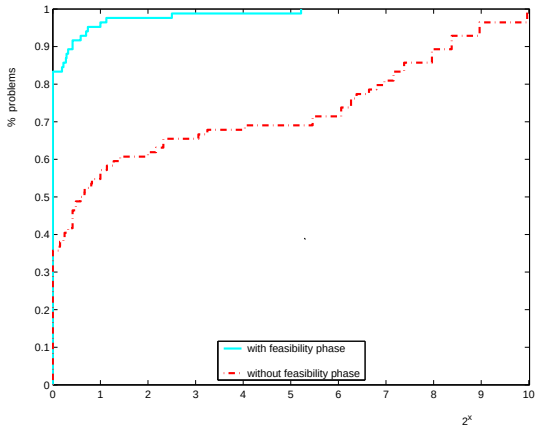

Figure 5: Iterations performance profiles

From this figure it is clear that Filter IR with feasibility phase has the highest probability of being the optimal solver for more than 80% of the problems. A detailed numerical comparison of Filter IR to other NLP solvers, namely NPSOL and LOQO, is presented in [10].

6.1 Comparison with IPOPT

Wächter and Biegler [13] present the IPOPT code, an implementation of a primal-dual interior-point algorithm with a filter line search method for nonlinear programming. In this work a comparison with other interior-point codes is presented. A detailed numerical study based on 954 problems from the CUTEr test set is reported. The results can be downloaded from Wächter home page (`http://www.research.ibm.com/people/a/andreasw`). Based on these results we decided to compare the Filter IR code with IPOPT. Since the 235 test problems are from different databases we have considered 122 problems from CUTEr tested by both codes. The main purpose of the comparison is to give an idea of the relative performance of the Filter IR.

Our metrics are the number of iterations (Figure 6) and the number of function evaluations (Figure 7). The iterations performance profile analysis shows that for about 70% of the problems the Filter IR has the highest probability of being the optimal solver. With respect to the function evaluations the Figure 7 shows that in 70% of the problems the performance is similar but the IPOPT has better performance for the rest of the problems. The authors [13] say that performing the regular backtracking line search procedure can be unsuccessful due to rounding errors, and can result in an unnecessary switch to the restoration phase. In order to prevent this, they

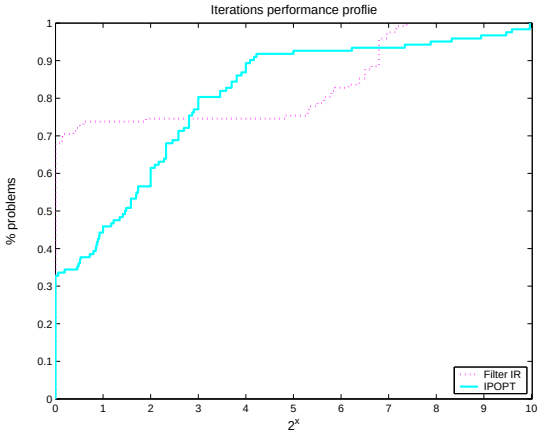

Figure 6: Iterations performance profiles

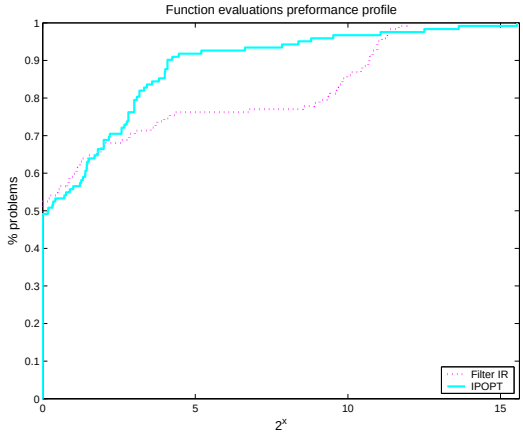

Figure 7: Function evaluations performance profiles

allow the algorithm to take the full step - their algorithm saves some function evaluations with this option.

7 Conclusions and future work

The numerical results are already very promising and encourage the algorithm development. As future work some important tasks will be performed - to prepare the algorithm for equality constraints problems, that must have a special treatment, and to use test problems with larger dimension. The filter management analysis, like the relation between its size at the end of the iterative process and the number of insertions, will be object of study. The

C code must be optimized and then it is also expected the CPU time performance analysis. The comparison with IPOPT shows that some test problems penalize the Filter IR performance - a careful analysis must be performed to identify the reasons of this behaviour. To introduce second order corrections avoiding the Marato's effect and to implement the filter restoration phase are also ideas to consider.

References

[1] A. S. Antunes and M. T. T. Monteiro. A filter algorithm and other nlp solvers: Performance comparative analysis. *Recent Advances in Optimization. A. Seeger (Ed.). Lectures Notes in Economics and Mathematical Systems*, 563:425–434, 2006.

[2] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, Ser. A 91:201–213, 2002.

[3] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, A 91:239–269, 2002.

[4] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming.* Boyd & Fraser publishing comapny, Massachusetts, 1993.

[5] P. E. Gill, S. J. Hammarling, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for LSSOL: A fortran package for constrained linear least-squares and convex quadratic programming. Technical Report 86-1, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1986.

[6] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for NPSOL 5.0: A fortran package for nonlinear programming. Technical Report CA.94 305, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1998.

[7] C. C. Gonzaga, E. Karas, and M. Vanti. A globally convergent filter method for nonlinear programming. *SIAM Journal on Optimization*, 14:646–669, 2003.

[8] J. M. Martínez. Inexact-restoration method with Lagrangian tangent decrease and new merit function for nonlinear programming. *Journal of Optimization Theory and Applications*, 111:39–58, 2001.

[9] J. M. Martínez and E. A. Pilotta. Inexact-restoration algorithm for constrained optimization. *Journal of Optimization Theory and Applications*, 104:135–163, 2000.

[10] C. E. P. Silva and M. T. T. Monteiro. A filter algorithm - comparison with NLP solvers. *International Journal of Computer Mathematics*, 2007. to appear.

[11] A. Wächter and L. T. Biegler. Line search filter methods for nonlinear programming: Local convergence. *SIAM Journal Optimization*, 16(1):32–48, 2005.

[12] A. Wächter and L. T. Biegler. Line search methods for nonlinear programming: Motivation and global convergence. *SIAM Journal of Optimization*, 16(1):1–31, 2005.

[13] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.