Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach

A.R. Conn[1], Nick Gould[2] and Ph.L. Toint[3]

Report 93/15 June 16, 1993

Abstract. This paper examines the question of modifying the decomposition of a partially separable function in order to improve computational efficiency of large-scale minimization algorithms using a conjugate-gradient inner iteration. The context and motivation are given and the application of a simple strategy discussed on examples extracted from the CUTE test problem collection.

[1] **IBM T.J. Watson Research Centre**
Yorktown Heights, USA

[2] **CERFACS**
Toulouse, France, EC

[3] **Department of Mathematics,**
Facultés Universitaires ND de la Paix, Namur, Belgium, EC

Keywords : exploitation of structure, algorithmic efficiency, partially separable functions.

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No F49620-91-C-0079. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

1 Introduction

Large-scale numerical optimization, like many other fields involving large problems, heavily relies on two fundamental but distinct endeavours: the use of structure and the search for maximum algorithmic efficiency. Indeed, many of the methods proposed in this area require that the user specifies the problem's structure in some prescribed way, and are designed to exploit this given structure to the largest extent possible. However, it is frequently assumed that the problem's structure is *given*, and that a good algorithm has to exploit it. In this paper, we consider the complementary point of view: we examine the question of *modifying* the problem's structure, in the hope that this modified structure can lead to improved algorithmic performance. Although not new in other areas of computational mathematics (see, for instance the work by Chan and McCormick [3] on how to make sparse matrices sparser), this idea does not seem to have been much studied in the context of large-scale optimization.

It is the purpose of this paper to consider this question in the context of partially separable functions. Introduced by Griewank and Toint [8], this particular structure and its generalization to group partial separability have shown to be very useful in the design of algorithms for large-scale optimization problems, both constrained and unconstrained. For instance, the LANCELOT package (see Conn *et al.* [7]) is based on this structural concept. In this context, we will consider that the partially separable structure of a function is given, and will then try to improve it with a very specific goal in mind: we aim at reducing the amount of computational time spent in the calculation of a step of a truncated-Newton algorithm using the conjugate gradient technique. This particular choice is motivated by the frequent use of this technique in large-scale optimization methods, and, more precisely, by the potential benefits that could be achieved within the LANCELOT package itself.

The paper is organized as follows. Section 2 formally introduces two related problems in modifying a partially separable structure: element merging and expansion. Section 3 describes a simple algorithmic approach to partially separable structure improvement, while Section 4 presents some results obtained by applying the algorithm of Section 3 to test examples extracted from the CUTE test problem collection of Bongartz *et al.* [2]. A more general discussion of the subject is presented in Section 5.

2 The merging and expansion problems in partially separable structures

In order to motivate our approach in a simple framework, we consider the unconstrained optimization problem of minimizing

$$f(x) = \sum_{i=1}^{m} f_i(x), \tag{2.1}$$

a partially separable function of the n-dimensional real vector x. (We refer the reader to Conn *et al.* [4] or Chapter 2 of [7] for a detailed introduction of partial separability and group partial separability.) Assume furthermore, for the moment, that $f(x)$ is convex and twice continuously differentiable. Suppose finally that the problem is to be solved on a sequential computer by applying Newton's method, and that the linear conjugate gradients algorithm is selected for calculating the Newton's step at a given iterate x. The problem is then to solve (possibly approximately) the linear system

$$H(x)s = -g(x) \tag{2.2}$$

where $H(x)$ denotes the (positive definite) Hessian matrix of f at x, $g(x)$ its gradient at the same point and where s is the desired step. Since the main computational cost within the conjugate gradient algorithm is the multiplication of the involved matrix with a vector, we see that the cost of solving (2.2), even approximately, is likely to be dominated by that of computing products of the form $H(x)v$ for given x and v. Since x is fixed for a given Newton iteration, we will omit it from now on.

Now observe that, because of (2.1), one has that

$$H = \sum_{i=1}^{m} H_i, \tag{2.3}$$

where H_i is the Hessian of the i-th *element function* f_i (at x). Hence one can compute the desired product in at least two different ways, as

$$Hv = \left(\sum_{i=1}^{m} H_i\right) v \quad \text{or} \quad Hv = \sum_{i=1}^{m}(H_i v). \tag{2.4}$$

We refer to the first possibility as the *fully assembled form* and to the second as the *fully disaggregate form.* But these are only the two extreme situations: if m is large (which is the case of interest) they are many possible ways to write the vector Hv, depending on which elements are assembled before the product of the partially assembled matrix with v is finally computed.

An additional degree of freedom may be present in a partially separable structure. It is indeed often the case that a distinction can be made between the *elemental* variables associated with the i-th element and the *internal* variables associated with this element. The vector of elemental variables for the i-th element, which we denote x_i^E is a subvector of dimension n_i, say, of the vector x, containing only the components of x that explicitly appear in the formulation of the i-th element function. A vector of internal variables is then defined in the case where the i-th element function can be written, for all x, as

$$f_i(x_i^E) = f_i(W_i x_i^I) \tag{2.5}$$

for some $p_i \times n_i$ full rank matrix W_i, with $p_i \leq n_i$, and for some vector of internal variables x_i^I. (Again, see Conn *et al.* [7] for a detailed exposition.) We say that n_i is the *elemental dimension* of the i-th element, while p_i is its *internal* dimension and W_i is the associated *range transformation.* We furthermore denote by H_i^E the $n_i \times n_i$ matrix restriction of H_i to the subspace of the elemental variables of the i-th element. When $p_i < n_i$, the *elemental Hessian* H_i^E is written and usually stored as a product of two range transformations and an *internal Hessian* H_i^I, that is

$$H_i^E = W_i^T H_i^I W_i. \tag{2.6}$$

Thus even the nonzero part of a simple product $H_i v$ can be computed as

$$H_i^E v = \left(W_i^T H_i^I W_i\right) v \;\text{ or }\; H_i^E v = W_i^T (H_i^I (W_i v)) \tag{2.7}$$

Two questions then arise if the overall product Hv is to be computed efficiently.

- How far should one go into the partial assembly of the matrices H_i (or, equivalently, of the H_i^E) between the two extremes of (2.4)? This can be viewed as deciding whether to "merge" elements in the partially separable structure of f.
- Should one "expand" H_i^I by explicitly computing the result of (2.6) before computing the partial product $H_i v$?

Efficient answers to these questions will naturally involve some trade-off between computing speed and storage requirements. Note that we assume here that the vector v is dense, as is usually the case in a conjugate-gradient technique. Products of Hessian with sparse vectors are also of interest, for instance in the context of a Generalized Cauchy Point calculation, but are typically performed in a specialized and cheaper fashion (see Section 3.3.5 of Conn *et al.* [7], for instance).

We note that we only consider modifying the partially separable structure of f by aggregating some of its components. Indeed, if (2.1) is given by the user, there is no automatic way to disaggregate the problem further, as this typically requires refinements in the user's model. This might of course be desirable, but is beyond the scope of this paper.

We next observe that the questions raised above are also valid if the problem is not convex. Indeed, conjugate gradients are still often used in this more general case: either the Hessian is suitably modified to make it positive definite, or directions of negative curvature detected within the conjugate gradient iterations are exploited in a trust region framework. But the efficiency of the matrix-vector products remains crucial.

We also note that merging elements and/or expanding elemental Hessians is not always computationally advantageous: it strongly depends on the initial decomposition (2.1). The procedure described below should therefore have little or no effect if this initial decomposition was determined with the preoccupation of making matrix vector products reasonably efficient. We only aim here at improving possibly unfortunate choices of (2.1).

We finally claim that our definition of computing efficiency (solely based on the number of floating point operations), although somewhat restrictive, is an acceptable *a priori* way to assess the potential of element merging before attempting to rewrite a specialized large-scale optimization code that can exploit this potential to its real extent.

3 A first algorithmic approach to merging and expansion

We now consider the merging and expansion question in more detail. Consider first the possible expansion of element i from internal to elemental representation. The situation is pictured in Figure 1.

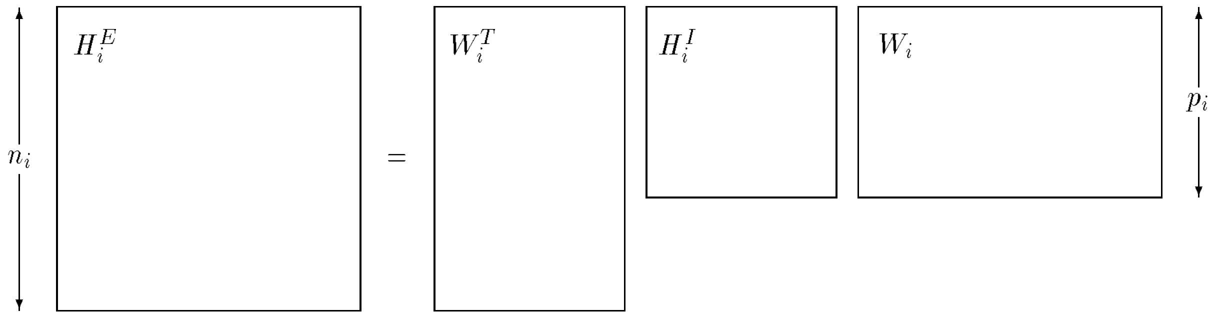

Figure 1: Elemental vs internal Hessian representation

Note that not all matrices in this picture are dense. In fact, W_i (and W_i^T) often contain a significant proportion of zeros. Let d_i be the number of nonzeros in W_i. Given this number, we may then decide to expand the i-th element if the product of the matrix on the left (of Figure 1) with v is less costly that the products on the right, that is when

$$n_i^2 \leq 2d_i + p_i^2. \tag{3.1}$$

Consider now element merging, and assume, for simplicity, that we examine the possibility of merging two elements, elements i and k, say, both expressed in terms of their elemental variables (either originally stored in terms of these, or expanded in a first step). Figure 2 shows the structure of the Hessian matrix H restricted to those elements. In this Figure, n_i and n_k stand for the elemental dimensions of elements i and k respectively. We denote by n_{ik} the number of elemental variables that are common to both elements. Notice that H_i and H_k are assumed to be stored separately as dense matrices.

The number of floating point operations[1], or flops, required to compute the product H_iv+H_kv is then $n_i^2 + n_k^2$. If, on the other hand, we decide to merge elements i and k, the Hessian matrix of that merged element is represented in Figure 2 by the large square of dimension $n_i + n_k - n_{ik}$. Because the zeros in the off-diagonal blocks are stored in the merged Hessian just as other non-zero values, computing the product $(H_i + H_k)v$ now costs $(n_i + n_k - n_{ik})^2$ flops. It is thus

[1] A floating point operation is defined here as an add-multiply pair.

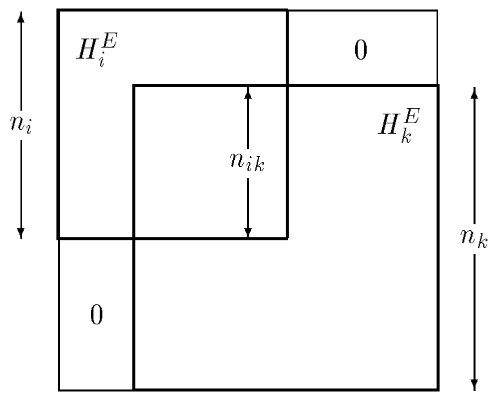

Figure 2: Two elements in the Hessian matrix

advantageous to merge elements i and k if

$$n_{ik}^2 \geq 2(n_k - n_{ik})(n_i - n_{ik}), \tag{3.2}$$

which is to say that the area of the overlap between the two elemental Hessians is at least that of the two off-diagonal blocks. Given n_i, n_k and n_{ik}, (3.2) thus provides a simple rule for merging two elements in elemental representation.

If one or both elements i and k are stored in terms of their internal variables, the situation is slightly more complex. If we define

$$c_i = \begin{cases} n_i^2 & \text{if element } i \text{ is stored in elemental representation,} \\ 2d_i + p_i^2 & \text{if element } i \text{ is stored in internal representation,} \end{cases} \tag{3.3}$$

then a suitable test is obviously given by the condition

$$(n_i + n_k - n_{ik})^2 \leq c_i + c_k. \tag{3.4}$$

If this condition holds, elements i and k are first expanded and their expanded representations are then merged.

Gathering conditions (3.4) and (3.1), we may set up a simple algorithm, whose idea is

1. to first examine all elements and decide, on the basis of (3.1), if they should be expanded,

2. to then consider pairs of elements and decide, on the basis of (3.4), if they should be merged.

We note that only elements for which elemental and internal dimension differ need being considered in the first step. It is also necessary to compute the density d_i for all such elements. This is quite acceptable from a computational point of view, as it only involves work of the order of the number of elements m (assuming n_i is small compared with m). The situation is even more favourable in the frequent case where all elements involve only very few *element types*, which determine their internal or elemental representation.

If we now turn to the second step, we see that only pairs of elements having common elemental variables need being examined for possible merging. Furthermore, the number of such common variables must be known for all these pairs. A naive implementation of this scheme would thus require of the order of m^2 operations, which is excessive when m is large (often much larger that n). Moreover, it is still possible to merge two elements which result themselves from previous merging operations. This makes deciding on the best merging sequence (given our computational efficiency criterion) a truly combinatorial problem. More precisely, it can be viewed as a large-scale set covering problem (see Nemhauser and Wolsey [10], for instance) where one wishes to cover the set $\mathcal{E}$ of all elements with merged elements (subsets of $\mathcal{E}$) with minimal computational cost. We do not intend, in this paper, to explore in depth the specialized algorithms for set covering, but we will rather design a relatively simple computational procedure for our element merging problem. This procedure can be described as follows.

Element merging procedure

Step 1: Compute the lists of all elements involving a given variable

This can be achieved in a single loop on the elements. Let e_{ij} be the j-th element involving the i-th variable.

Step 2: Perform a merging pass

For each variable i in turn, and for each $j > 1$, examine if element $k = e_{i1}$ should be merged with element $s = e_{ij}$:

1. compute n_{ks}, the number of variables common to elements k and s,
2. merge elements k and s if (3.4) holds,
3. update the element/variable lists if merging occurred.

Step 3: Stopping test

If any merging occured in the execution of Step 2, re-execute Step 2. Otherwise stop.

End of procedure

There is no doubt that the procedure can be improved. For instance, one might wish to avoid recomputing n_{ks} but instead update it when two elements are merged. But, despite its simplistic nature, this scheme will allow us to illustrate the potential benefits of element merging.

We close this section by mentioning a structural improvement which can sometimes be achieved if the objective function f is *group partially separable*, that is if it can be expressed as

$$f(x) = \sum_{j=1}^{q} g_j \left(a_j^T x - b_j + \sum_{i \in E_j} w_i f_i(x) \right), \tag{3.5}$$

where, for each $j = 1, \ldots, q$, g_j is a continuous real function of one real variable, a_j is a given vector of $\Re^n$, b_j a given scalar, E_j the set of indices of the element functions f_i appearing in the j-th *group*, and where the scalars w_i are known as *weights*. We say that the j-th group is *trivial* if

$g_j(\alpha) = \alpha$. If we now assume that there is more than one trivial group, it is immediately obvious that all trivial groups could be merged into a single one with

$$a = \sum_{j \in \mathcal{T}} a_j, \quad b = \sum_{j \in \mathcal{T}} b_j \text{ and } E = \bigcup_{j \in \mathcal{T}} E_j, \tag{3.6}$$

where $\mathcal{T}$ is the index set of all trivial groups. This structural modification does not affect the Hessian times vectors products, and is therefore not immediately relevant for the objective pursued in this paper. It is however of some practical value, as it simplifies the data structure associated with the problem description. We thus include it within the expansion stage of our simple algorithm for structure improvement.

4 Preliminary applications

The element and group merging and expansion techniques described in the previous section were implemented in Fortran and applied to a number of examples from the CUTE test problem collection (see Bongartz *et al.* [2]). On a large number of these examples, the algorithm produced no modification of the structure. This merely shows that the initial partially separable structure of many of the CUTE examples is relatively satisfactory from the point of view adopted here. However, several test problems were not left unmodified: we report below on the changes produced by our simple technique on some of them.

We first report the effect of the merging of all trivial groups into a single one, as discussed at the end of Section 3. Our results are shown in Table 1, where both the number of groups (under the heading "Groups") and the total number of nonzero linear coefficients a_j (under the heading "Linear coeff.") are reported. Each of these quantities are detailed before (in the "initial" columns, corrresponding to the problem's structure as given in the CUTE collection) and after (in the "tr .merged" columns) merging of all trivial groups.

The effect of merging trivial groups is clearly apparent in several examples: they typically only contain trivial groups, but sometimes many of them, in the initial decomposition. An exception is the HAGER3 example, where this initial merging only reduces the number of groups by one third. We also see that the total storage requirement for storing the linear coefficient may substantially decrease with trivial group merging, as happens for the NCB20B example.

After this preliminary merging is performed, we now turn to the effect of structural modifications impacting the amount of arithmetic in matrix-vector products. Our results are summarized in Table 2. In this table, we indicate the effect of the algorithm on the structure itself, reflected by the number of elements (in the "Elements" columns). We also present the effect of the structural modifications on our main criterion, the number of floating point operations in a matrix-vector product Hv (in the "Flops" columns), as well as on the amount of storage requested for the complete Hessian matrix H, taking symmetry into account (in the "Storage" columns). The sub-headings "initial", "expanded" and "merged" respectively correspond to the initial structure (as given within CUTE), to the structure after element expansion and after element merging.

We note the following points.

Problem name	Groups		Linear coeff.	
	initial	tr.merged	initial	tr.merged
NCB20B	1000	1	19620	1000
TORSION1	5184	1	5184	5184
JNLBRNGA	5329	1	5329	5329
LMINSURF	5476	5476	0	0
OBSTCLBU	5329	1	5329	5329
HAGER3	15000	10001	20000	20000
HILBERTB	1275	1	0	0
SINQUAD	5000	5000	1	1
SCHMVETT	4998	1	0	0
CRAGGLVY	12495	12495	17493	17493

Table 1: The effect of merging trivial groups

- We first notice the effect of element merging on the number of elements itself. We see a sometimes significant decrease in the number of elements, in particular for problem NCB20B. The major reduction on this example is explained by the fact that this problem has a band structured Hessian of semi-bandwidth 20, which is originally described as the superposition of 981 principal submatrices of dimension 20, overlapping each other in 19 variables, to which are added 1000 one dimensional diagonal elements. Merging these elements is thus clearly advantageous.

- As expected, the overall number of floating point operations needed to compute Hv is steadily decreasing for all examples. Although not very striking on problem HILBERTB, the gain exceeds 10% on all other cases. Furthermore, it can be extremely important, for instance for problems NCB20B and HAGER3. This good performance on NCB20B is again explained by the significant amount of overlap between elements in the initial decomposition.

  Other cases of interest are TORSION1, JNLBRNGA and OBSTCLBU. These problem are quadratic with a structure arising from the discretization of a two dimensional variational problem. In the original description, the diagonal terms of the Hessian were separated from the off-diagonal terms. They are included in principal submatrices of dimension larger than one in the modified structure.

  It is important to emphasize here that the reduction obtained is very worthwhile, even if it is modest. Indeed, matrix-vector products of the form Hv occur at *every* conjugate gradient iteration in the (approximate) solution of (2.2), and this latter system needs to be solved at every iteration of a truncated Newton's method. For instance, a total of 3819 conjugate gradient iterations are required by LANCELOT (with default settings) to reduce the norm of the objective function's gradient below 10^{-7} for NCB20B , a moderately dif-

Problem name	Elements		Flops			Storage		
	initial	merged	initial	expanded	merged	initial	expanded	merged
NCB20B	1981	6	393400	393400	209025	207010	207010	105060
TORSION1	20736	18144	103680	82944	72576	20736	62208	57024
JNLBRNGA	21316	18651	106580	85264	74604	21316	63948	58618
LMINSURF	10952	10952	54760	43808	43808	10952	32856	32856
OBSTCLBU	21316	18651	106580	85264	74604	21316	63948	58618
HAGER3	10000	5000	70000	65000	45000	30000	45000	35000
HILBERTB	1275	1275	4950	4950	4900	3725	3725	3724
SINQUAD	14996	9998	34988	29990	24992	14996	24486	22486
SCHMVETT	14994	4998	89964	79968	79968	24990	44982	44982
CRAGGLVY	4998	4998	14994	12495	12495	4998	9996	9996

Table 2: Effect of structure modification on operation count and storage

ficult unconstrained problem in 1000 variables. Moreover, if the considered problem has constraints and an augmented Lagrangian (see Powell [11], Bertsekas [1] or Conn *et al.* [5]) or a Lagrangian barrier (see Conn *et al.* [6]) approach is used, there may even be several sequences of Newton's iterations [in other words, we solve several (bound-constrained) optimization problems], which typically results in a relatively large number of conjugate gradient iterations to solve the problem.

- If we now consider the memory requirements, we see that expansion, whenever it occurs, clearly increases the necessary storage by a factor which, in our examples, could be as much as three. We may also conclude from these results that the internal representation is often very efficient (in terms of space) compared to elemental representation. This may be an important observation when storage becomes an issue, either because of the mere size of the problem, or because of restrictions due to a particular computer platform.
- The computational gain obtained by element expansion is often comparable to that obtained by element merging. But the previous remark indicates that the benefit from element expansion is obtained at some storage cost. In contrast, element merging typically reduces both storage and arithmetic.

5 Discussion

We have introduced the concepts of element merging and expansion within a given partially separable structure, and have proposed a simple use of these concepts to improve the structure from a very specific point of view: that of optimizing the amount of arithmetic in matrix-vector products of the type arising in conjugate gradients. We have also shown that our proposal can have a clearly beneficial effect on problems whose initial decomposition may be natural, but suboptimal from the point of view of optimizing the amount of arithmetic in matrix-vector products.

It is very clear that merging and expansion may have other kinds of impact on optimization algorithms. An interesting other instance is when partitioned quasi-Newton approximation schemes (see Griewank and Toint [9]) are used to determine the Hessian matrix H, as is a possibility in LANCELOT. In this case the definition of an element is crucial because a different secant equation is enforced *per element*. The decomposition then induces the structure of the Hessian matrix, and the off-diagonal zero blocks of Figure 2 do not appear when two elements are merged. Hence one typically obtains different approximations for two elemental Hessians, depending on whether they are merged or not. As the quality of the Hessian's approximation is crucial for the overall behaviour of the minimization algorithm (in terms of iterations, for example), we may observe a direct effect of element merging at this very aggregate level. We refer the interested reader to Toint [12] for an analysis of this effect in the context of large-scale nonlinear least-squares calculations.

We also note here that computer architecture may play an important role in the decisions considered here. Indeed element merging is directly related to the granularity of the matrix-vector product calculations. If parallel processors are available, the optimum level of granularity may vary depending on what particular machine is considered, and good merging schemes should therefore vary accordingly.

Finally, we note that the techniques considered above are largely independent of the ordering of the variables and/or elements, inasmuch as they are based on lists of elements involving given variables. However, some dependency on the ordering is still present because we attempt to merge the *first* and subsequent elements in each such list. It is difficult to say from our preliminary experience in what measure this dependence plays a significant role in the final results.

Despite its direct practical interest, we of course realize that the research described in this paper is very limited in scope. We hope that it will encourage further research into better ways to improve the structural description of large optimization problems.

6 Acknowledgements

The authors wish to acknowledge additional funding provided by a NATO travel grant.

References

[1] Bertsekas, D.P. (1982), *Constrained Optimization and Lagrange Multipliers Methods.* Academic Press, London.

[2] Bongartz, I., Conn, A.R., Gould, N. and Toint, Ph.L. (1993), CUTE: Constrained and Unconstrained Testing Environment. Technical Report 93/10, Department of Mathematics, FUNDP, Namur, Belgium.

[3] Chang, S.F. and McCormick, S.T. (1992), A hierarchical algorithm for making sparse matrices sparser. *Mathematical Programming*, 56(1):1–31.

[4] Conn, A.R., Gould, N. and Toint, Ph.L. (1990), An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. In R. Glowinski and A. Lichnewsky, editors, *Computing Methods in Applied Sciences and Engineering*, pages 42–51, Philadelphia, USA. SIAM.

[5] Conn, A.R., Gould, N. and Toint, Ph.L. (1991), A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572.

[6] Conn, A.R., Gould, N. and Toint, Ph.L. (1992), A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. Technical Report 92/07, Department of Mathematics, FUNDP, Namur, Belgium.

[7] Conn, A.R., Gould, N. and Toint, Ph.L. (1992), LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A).* Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York.

[8] Griewank, A. and Toint, Ph.L. (1982), On the unconstained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312, London and New York. Academic Press.

[9] Griewank, A. and Toint, Ph.L. (1982), Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:119–137.

[10] Nemhauser, G.L. and Wolsey, L.A. (1988), *Integer and Combinatorial Optimization.* J. Wiley and Sons, New York.

[11] Powell, M.J.D. (1969), A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, London and New York. Academic Press.

[12] Toint, Ph.L. (1987), On large scale nonlinear least squares calculations. *SIAM Journal on Scientific and Statistical Computing*, 8(3):416–435.