Large-Scale Numerical Optimization:
Introduction and Overview*

Thomas F. Coleman**

TR 91-1236
September 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*To appear in the Encyclopedia of Computer Science and Technology, published by Marcel Dekker, Inc.

**Computer Science Department, Advanced Computing Research Institute and Center of Applied Mathematics, Cornell University, Ithaca, NY 14853. Research partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000 and by the Computational Mathematics Program of the National Science Foundation under grant DMS-8706133.

LARGE-SCALE NUMERICAL OPTIMIZATION: INTRODUCTION AND OVERVIEW*

THOMAS F. COLEMAN†

Abstract. We give an introductory overview of the field of large-scale numerical optimization; some of the basic research issues and recent developments are described. Our emphasis is on methods, techniques, and practical concerns. We hope this article will be of interest to both users and students of numerical optimization.

* To appear in the Encylopedia of Computer Science and Technology, published by Marcel Dekker, Inc.

† Computer Science Department, Advanced Computing Research Institute, and Center for Applied Mathematics, Cornell University, Ithaca, NY 14853. Research partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000 and by the Computational Mathematics Program of the National Science Foundation under grant DMS-8706133.

I. Introduction. Large-scale optimization is concerned with the minimization (or maximization) of functions of large numbers of variables. Additional algebraic constraints may be imposed on some of the variables, or combinations of variables, in which case we are dealing with constrained optimization. It is difficult to give a precise definition of "large-scale": however, a pragmatic view is that a problem should be viewed as "large-scale", in a given computing environment, if it is economical to exploit structure. Structure can refer to sparsity (i.e., presence of many zeros in the various matrices), inherent parallelism, separability characteristics, and so on.

The purpose of this article is to give a personal introductory overview of the field, emphasizing methods, techniques, and practical concerns. We give theoretical issues a very brief treatment in this article. This is not meant to imply that theoretical issues are not important; indeed, every successful optimization method is supported, in part, by strong theoretical underpinnings. However in this article, in the interest of brevity and breadth, we place theory in the background and instead highlight numerical and practical issues related to efficient computer implementation.

Research activity in the field is high; we hope this article will introduce the reader to some of the issues and trends. More details can be found in the proceedings of a recent workshop on this area [1]. Some of the ideas are discussed at greater length in [2]; background in optimization can be found in a number of optimization texts including [3,4,5,6,7]. Recent research developments are discussed in Fletcher [8] as well as in the handbook edited by Nemhauser, Rinnooy Kan, and Todd [9]. Some parallel computing issues in optimization are discussed in [10,11].

The unconstrained minimization problem is usually stated as $\min_x f(x)$, where f is a real-valued function of n real variables $x = (x_1, x_2, ..., x_n)^T$. In this article we are primarily concerned with the case where f is smooth and differentiable[1] (usually twice continuously differentiable); however, many important large-scale optimization problems arise which do not satisfy these assumptions. For example, in some cases some of the variables (perhaps all) must attain integral values. Typically this makes the problem more difficult to solve and usually very costly. Wu [13] provides an example of such a problem and a parallel method of solution. Even without integrality constraints the general problem, minimize $f(x)$, can be very difficult to solve, especially if the global minimizer is required (i.e., a point x_* satisfying $f(x_*) \leq f(x)$, over all x). The distance conformation problem, in Example 1, is an example of such a problem.

There are several things to note about this problem. First, a point x is a global solution if and only if $f(x) = 0$; therefore, verifying[2] if a candidate solution is in fact a global solution is easy – this is not always the case in global optimization. Second, it is easy to see that the Hessian matrix, $\nabla^2 f(x)$, will be a sparse matrix if $|S| << \frac{n(n-1)}{2}$: variable x_i is related to variable x_j if and only if $(i,j) \in S$. The problem as stated actually yields a singular Hessian matrix, at any point – consider, for example, simple translations or rotations. Singular matrices can lead to numerical difficulties; however, in this case it is possible to overcome this difficulty by introducing a few simple constraints (to orient the molecule, arbitrarily, in space).

Notice that to satisfy the known information a global minimizer of (1) is required.

[1] We assume that the gradient of f, ∇f, can be computed. There are methods available that do not require the gradient, e.g., [12], but we do not discuss such methods here.

[2] However, in practice the "known" distances may be contaminated with error in which case it may not be possible to realize $f(x) = 0$, for any x. This gives more credence to the optimization formulation (1).

Example 1 [14,15].

This problem arises in several application areas including molecular chemistry, surveying, and satellite ranging. We will describe the problem in the parlance of molecular chemistry. It is required to locate, in 3-space, the positions of the atoms of a molecule given (incomplete) pairwise distance data. That is, let S be the set of index pairs corresponding to known distances, i.e., $(i,j) \in S$ if the distance between atom i and atom j, d_{ij}, is known. The cardinality of the set S, $|S|$, is usually much smaller than the number of all possible pairs, i.e., $|S| << \frac{n(n-1)}{2}$. The problem is to locate all atoms x_i, $i = 1,2,...,n$, in 3-space such that $\|x_i - x_j\| = d_{ij}$ for all $(i,j) \in S$. The dimension of the problem, n, is typically between several hundred to several thousand. The usual optimization formulation is

$$\text{(1)} \qquad \min \ f(x) = \sum_{(i,j)\in S} (\|x_i - x_j\|_2^2 - d_{ij}^2)^2.$$

FIG. 1. *The distance conformation problem*

However, locating a global minimizer of (1) is, in general, almost intractable. Indeed, even if it is known that the molecule is a 1-dimensional structure (i.e., all the atoms form a single straight line), a very unlikely and simplified situation, the global minimization of (1) is still essentially intractable, in general. (Hendrickson [15] shows that this 1-dimensional problem is at least as hard as the "partition problem" which is known to be NP-complete – see Garey and Johnson [16] for an introduction to the theory of NP-completeness.)

We are not concerned with global optimization techniques in this article. Currently, there is a lot of research activity in this area including development of parallel computational techniques, e.g., [17,18,19,20]. It is a very important area. However, due to the extreme difficulty of this problem, it appears that domain-specific heuristics will usually play an important role in the design of efficient computational techniques for large-scale global optimization problems.

In this article we are primarily concerned with local minimization. In the unconstrained case this means finding a point x_* with the property that there is a neighborhood containing x_* such that f attains its lowest value, in this neighborhood, at x_*. More formally we say that x_* is a local minimizer if there exists a scalar $\delta > 0$ such that $f(x_*) \leq f(x)$ for all x satisfying $\|x - x_*\| < \delta$. Most optimization problems in science and engineering are adequately solved by local minimization. Moreover, global optimization problems are usually solved using a local minimization procedure as a subroutine, e.g., [18,19,21].

The application areas where optimization is used are too numerous to enumerate; indeed, it is difficult to think of a significant branch of computational science where optimization is absent. Moreover, realistic models often lead to large-scale problems. To illustrate further, we consider two more examples, both from the bio-medical research area. Both examples are merely sketched here to illustrate the nature of the optimization problem and its relationship to the application – to fully understand the derivation of the optimization problem the reader must refer to the sources cited.

Important things to note about Example 2 are: the objective function is nonlinear, there are lower bounds on the variables, the matrix S is large, very sparse, and with some block-structure (sparsity and block structure are both consequences of the grid design and "nearest-neighbor relationships").

Example 2 [22,23,24,25]. In this example one is interested in determining the distribution of radioactivity across a body cross-section. If a grid is placed across the body cross-section, dividing the cross-section into a number of squares, the quantity to be estimated is the level of radioactivity in each square, say x_i in square (or pixel) i. Assume the grid divides the cross-section into n pixels. To yield useful results, n must be very large – typically in the range $[10,000 \;-\; 100,000]$. So $x = (x_1, x_2, ..., x_n)^T$ is the vector to be determined. However, the quantity that can be measured experimentally is the level of activity along a number of straight lines that cut across the grid. Let there be m lines l_i each with measured radioactivity level y_i. Partition $l_i = (l_{i1}, l_{i2}, ..., l_{in})^T$ where l_{ij} denotes the length of intersection of the i-th line with the j-th pixel. It is proposed [25] to solve the following optimization problem to determine the vector x:

$$\min_x \{f(x) = x^T S x \; + \; \sum_{i=1}^{m} (l_i^T x - y_i \cdot ln(l_i^T x)) : \quad x \geq 0\} \tag{2}$$

where S is a sparse symmetric positive semi-definite matrix (defined in [25]).

FIG. 2. *Image Reconstruction*

A few things to note about Example 3. First, an evaluation of the objective function is very expensive – requiring a finite element analysis, i.e., the determination of the matrix K and the solution of (4). The calculation of the gradient (and even the Hessian matrix, if required) is possible and is not inordinately expensive provided it is done, with care, along with the finite element analysis. If insufficient care is taken with the manner in which the function is evaluated the entire process will be unmanageably expensive. Sparse matrix technology will be required for the the solution of y; unfortunately, it will not be required for computations involving the Hessian matrix – the Hessian matrix is dense. The reason for this is that each component of y typical depends on all the variables x, through equation (4).

This third example differs from the previous two in that sparsity and structure do not show up in the Hessian matrix, which is dense. Instead, sparsity and structure play an important role in the evaluation of the function (and derivatives) and cannot be ignored here. Still, even with full attention paid to efficient function evaluations, the time spent to evaluate f (and derivatives) at a given point dominates all else. This fact – the extreme expense of evaluating the function – should play a determining role in the choice of optimization algorithm.

Many efficient and reliable algorithms have been developed for small-scale and medium-scale optimization: why are they not applicable to problems with many variables? The short answer is that they ignore structure usually present in large-scale problems; therefore, they are not as efficient as they need to be to solve such problems in reasonable time. For example, methods that by their nature cannot easily exploit sparsity cannot be applied to large-scale and sparse problems – computing time would be unmanageable. More subtly, perhaps, methods that consist of a large number of inexpensive but sequential or serial "major" iterations may be inappropriate for a large-scale problem in a parallel computing environment. (The major iterations cannot be overlapped; there is not enough work within each iteration to "crank up" a parallel supercomputer.) In this case algorithms with fewer but more expensive major iterations may be much more suitable provided the computations

Example 3 [26,27]. Broadly speaking this example falls into the subclass of optimization known as structural optimization. This type of problem arises when modelling structures of various kinds and is characterized by an objective function – the function to be minimized – of both dependent and independent variables. The dependent variables are themselves functions of the independent variables and can be determined only through the numerical solution to a system of partial differential equations. In this particular example of a structural optimization problem, the optimization process predicts how bone will "reconstruct" itself from a given starting position. The independent variables typically are bone density, bone thickness, and shape parameters. The objective function f is usually a combination of mass and strain energy. Before discretization the optimization problem is:

$$\min_{x} \{f(y(x), x) : \ l \le x \le u\}, \tag{3}$$

where f is a function combining mass and strain energy, and l, u are given vectors of lower and upper bounds respectively. The vector x represents the independent variables. The dependent variable y depends on x through a system of partial differential equations. Typically this problem is solved by first discretizing, dividing the bone to be modelled into a large number of "elements", and choosing finite vectors y and x. So, $y = y(x)$, and y is usually determined numerically, given x, by using finite element analysis. Amongst other things this involves forming and solving a very large sparse and positive-definite linear system of equations:

$$K(x)y = F(x) \tag{4}$$

where $K(x)$, the "global stiffness" matrix, is positive definite for all x, and F is a vector-valued function.

FIG. 3. *The modelling of bone reconstruction*

within these major iterations can be done efficiently in parallel. As a final example, large-scale problems often consist of a sum of functions of simple form – second derivatives can often be obtained at relatively little extra expense beyond evaluation of the function and the gradient (and parallelism can be used). Therefore, an efficient approach to a large-scale problem may involve the calculation and use of second derivatives whereas this is usually not the case in the small-scale setting.

However, despite this apparent need for different approaches when optimization problems become large, it is important to realize that there is a common backbone to methods for large-scale and small-scale optimization. This is the Newton iteration. Practically all of optimization consists of variations on the Newton process: globalization strategies, approximation techniques, and efficient robust implementations.

II. Unconstrained Minimization. The problem is: minimize $f(x)$, where f is a twice continuously differentiable real-valued function of n real variables $x = (x_1, x_2, ..., x_n)^T$. We are interested in locating a local minimizer: i.e., find a point x_* such that there exists a scalar $\delta > 0$ with the property that $f(x_*) \le f(x)$ for all x satisfying $\|x - x_*\| < \delta$.

There are two basic globalization strategies in unconstrained minimization: line search methods and trust-region methods. (By globalization strategy we refer to a method to ensure convergence to a local minimizer from a distant point. We do not refer to a method to find the global minimizer.)

In a line search method a direction of descent is determined, i.e., a direction s is determined satisfying $\nabla f(x_c)^T s < 0$, where x_c is the current point (i.e., the current approximation to the solution). Then, a one-dimensional approximate minimization is performed on the function $f(x_c + \alpha s)$, for $\alpha > 0$, to determine an acceptable α. Dennis and Schnabel [3] discuss this issue at fair length – in general the current consensus is to terminate the line search process under fairly lenient conditions, and in each iteration to initially try $\alpha = 1$.

An alternative to a line search algorithm is a trust-region procedure. In this case a trial step s_* is determined by minimizing the local quadratic model of f, at x_c, subject to a restriction on the size of the solution, and possibly subject to s being in some specified subspace S: quadratic algorithm is being used).

$$\min_{s \in S} \{\nabla f(x_c)^T s + \frac{1}{2} s^T H_c s : \|s\| \leq \Delta_c\}. \tag{5}$$

The matrix H_c is the current Hessian matrix $\nabla^2 f(x_c)$, or an approximation to it. In the full-blown trust-region problem S is the entire space R^n. However, in some recent and very promising work (mentioned in greater detail below) S is a 2-dimensional subspace. The norm used in the constraint is usually the 2-norm; however, both the 1-norm and the ∞-norm have also been advocated. The solution to (5), s_*, is accepted and x is updated, $x = x_c + s_*$, if $f(x_c + s_*) < f(x_c)$. The trust region bound Δ_c is adjusted for the next iteration, according to various simple rules, e.g., [5,28,29], and the process iterates.

Which approach should one use, line search or trust-region? Which is better for large-scale optimization?

There is a place for both approaches in large-scale optimization. The line search strategy is clearly appropriate if a positive definite approximation to the true Hessian is being used, e.g., a quasi-Newton or secant method. A problem with special structure may also be appropriate for a line search method since a specialized line search procedure to exploit this structure may be possible – this is much harder to do in a trust-region approach.

A trust-region method is better suited to handle indefiniteness[3] and is most reasonable when the true Hessian (or an accurate approximation) is being used. Convergence properties are stronger for trust-region methods and they tend to outperform line search methods on difficult problems (e.g., problems that are very nonlinear with a lot of negative curvature). The mark against trust-region methods has been their linear algebra cost (and complexity) within each major iteration – experience suggests almost two Cholesky factorizations, on average, per iteration. However, Byrd and Schnabel [30] propose choosing S to be a 2-dimensional subspace in which case the cost of the trust-region problem, once S is formed, is negligible (see also [31]). The trick is how to choose S. In [30] it is suggeted to let S be the space spanned by the gradient and the Newton direction, if the Hessian is positive definite, and the space spanned by the gradient and a direction of negative curvature[4] otherwise. A negative curvature direction can be obtained using an "incomplete" Cholesky factorization and perhaps some further low-order computations. This approach is still in its' early stages but seems most promising. Indeed, a combination of this reduced trust-region approach followed by a line search may capture the best of both worlds – this is a research question currently under investigation by the author.

[3] Note that H_c need not be positive-definite in (5).

[4] The vector s is a direction of negative curvature, with respect to a Hessian matrix H, if $s^T H s < 0$.

In any event, trust-region or line search, it is important to obtain or approximate second-order information (i.e., the Hessian matrix) to get acceptable convergence properties. As we illustrate below, it is often reasonable in the large-scale setting to obtain analytic expressions for the second derivatives. (See also the section on automatic differentiation, Section 4.) The extra computational cost beyond the cost of evaluating the function and the gradient is often acceptable – and the decrease in the number of iterations (over an approach that approximates the second derivatives) may be significant. However, an efficient Hessian computation can usually be computed only if intermediate quantities used in the evaluation of the function and the gradient are "re-used". Therefore, the common practice of writing separate subroutines to evaluate the function, gradient, and the Hessian is not a good one in this context (Figure 4 illustrates the use of a single subroutine to evaluate $f(x), \nabla f(x)$, and $\nabla^2 f(x)$ for the distance conformation problem.)

Of course it is not always possible or desirable to compute second-derivatives. It may not be convenient. For example, the user may have had a function and gradient routine handed down (from generation to generation) and that's where he or she wants to begin, not back at first principles. It may be the case that the problem is so large that even with sparsity it is not feasible to form and store the Hessian matrix – in this case, as we discuss briefly below, "product-form" approximations may be useful. In summary, there is a need for Hessian approximations and that's what we discuss next. Initially we focus on techniques that require storage of a sparse matrix; we follow this with a short discussion on possible alternatives suitable for very large problems.

A. Sparse Secant Updates. One of the biggest success stories in optimization has been the development of quasi-Newton (or secant) methods for unconstrained minimization. Dennis and Schnabel [3] provide an excellent introduction to this area; the classic theoretical reference is [32]. Unfortunately, the adaptation of this approach to the large-scale case, in which sparsity is a factor, has been less successful. (A possible exception is the use of secant updates in the context of a function f that is conveniently expressed in a partial separable manner – see below for more details.)

A secant method for multi-dimensional minimization requires that the user simply supply subroutines to evaluate f and the gradient of f , ∇f, at any given point x. An approximation $H(x)$ to the symmetric Hessian matrix, $\nabla^2 f(x)$, is maintained and updated at each iteration. The update requires the two pairs (x_c, x_+), $(\nabla f(x_c), \nabla f(x_+))$, and the current symmetric positive definite approximation H_c. The more successful updates require that the new approximation H_+ satisfy what is known as the secant (or quasi-Newton) equation:

$$H_+(x_+ - x_c) = \nabla f(x_+) - \nabla f(x_c). \tag{6}$$

The best updates also require that the approximation H_+ be positive definite (and symmetric) and this can be ensured provided H_c is positive definite and

$$s^T(\nabla f(x_+) - \nabla f(x_c)) > 0, \tag{7}$$

where $x_+ - x_c = \alpha s$ for some positive scalar α. A key point is this: condition (7) is consistent with approximate minimization of the function f along the line s starting at point x_c. This is crucial because it means that with an appropriate one-dimensional line search it is possible to obtain good decrease in the objective function f as well as ensure that a Hessian update (preserving the quasi-Newton equation (6), positive-definiteness, and symmetry) can be performed.

Many updating formulae have been suggested; however, the most successful to date (empirically) has been the BFGS (Broyden-Fletcher-Goldfarb-Shanno) update – see, for example, Chapter 9 in [3]. The updated matrix H_+ is obtained from the current approximation, H_c, with a rank-2 update. (Alternatively, an update to a matrix approximating the inverse of the Hessian can be performed.) A computational attraction of this positive definite secant update is the low linear algebra cost to perform the update: the work is $O(n^2)$, which includes the time to compute the next search direction. This is obviously true if the inverse approximation is updated; achieving this bound with a numerically stable update to the Cholesky factors of the Hessian approximation is illustrated, for example, in [3,7].

Unfortunately this success story does not carry over to the sparse setting. In the large-scale situation it is often the case that the Hessian matrix is sparse: i.e., most of the entries are zero for all values of x (Examples 1 and 2 illustrate this). Typically, the sparsity pattern is either known or can be determined before the optimization process begins. Therefore, a worthwhile goal is to maintain an approximating matrix with the same sparsity pattern – allowing zeros to turn into nonzeros can dramatically increase computational costs to the extent that the overall expense of a single iteration is prohibitive. Unfortunately, the quest for a sparse, symmetric, positive-definite secant method has produced a grab-bag of heuristic techniques and strategies, e.g., [33,34,35,36,37,38,39], none of which possess the satisfying theoretical and computational underpinnings of the dense positive definite secant updates suitable for small-scale problems. Coleman [2] summarizes this effort.

B. Sparse Finite -Differences. Frustrations with the development of a successful sparse secant update has lead to a search for alternatives and the generation of rather successful sparsity exploiting finite-differencing schemes.

To introduce this subject it is easiest to begin with the sparse finite-difference estimation of an unsymmetric Jacobian matrix. This arises in the common situation when trying to minimize a nonlinear least-squares function,

$$(8) \qquad \min_x f(x) \; = \; \frac{1}{2} F(x)^T F(x)$$

where $F(x)$ is a vector-valued function: $F(x) = (f_1(x), f_2(x), ..., f_m(x))^T$ and each component function $f_i(x)$ is a real-valued function of the vector $x \in R^n$. Typically $m > n$. Hence, the function $F(x)$ maps $R^n \to R^m$ and has an m-by-n Jacobian matrix $J(x) = F'(x)$. Our objective is to estimate $J(x)$ using a subroutine to evaluate $F(x)$. By Taylor's Theorem,

$$(9) \qquad J(x)d = \frac{F(x+\gamma d) - F(x))}{\gamma} + O(\gamma),$$

where γ, the step size, is a positive quantity and d is an arbitrary vector of unit length. Therefore an approximation to the i^{th} column of the Jacobian matrix can be obtained with a subroutine to evaluate F, by choosing $d = e^i$ where e^i is the i^{th} column of the identity matrix. Clearly it is possible to obtain an approximation to the Jacobian matrix, at the current point x_c, using $n + 1$ function evaluations by estimating each column of J in turn.

Curtis, Powell, and Reid [40] were the first to point out that many fewer function evaluations may be enough to estimate a sparse matrix. For example, a tridiagonal matrix requires only three finite-differences. The general idea is to partition the columns of the Jacobian matrix J into groups of "structurally independent" columns. Two columns are structurally independent if their row indices corresponding to nonzeros are disjoint. Coleman and Moré [41,42] proposed graph-coloring heuristics to determine partitions with few groups – each group corresponding to a function evaluation and a finite difference. The practical performance of this approach seems quite satisfactory requiring very few finite-differences (relative to n) on practical problems [41] . Nevertheless, there is room for possible improvement in the situation when there are a few dense rows: the number of required finite-differences is bounded below by the maximum number of nonzeros in any row. In such a case what may be required is independent approximation of the dense rows (perhaps a secant approximation).

In the more general minimization context, $\min_x f(x)$, we are required to estimate the sparse matrix of second derivatives (the Hessian matrix) given a subroutine to evaluate $\nabla f(x)$. The unsymmetric approaches, described above, can be used here; however, it is usually better to use a method that also exploits the symmetry of the Hessian matrix. Appropriate column partitionings, allowing for the use of symmetry, are provided by various graph colorings [43,44,45] and once again the practical performance of this approach seems to be quite good, requiring few finite-differences relative to n, and taking near optimal advantage of symmetry.

In summary, the sparse finite-difference approach, given a subroutine to evaluate $\nabla f(x)$ in the general minimization case, and given a subroutine to evaluate $F(x)$ in the nonlinear least-squares case, is often a viable approach for large-scale sparse problems. The number of evaluations is usually very small compared to n, the problem dimension. It is also true that the required finite-difference evaluations are totally independent tasks and so parallel computation of this aspect is easy. However, the efficient utilization of many processors, i.e., more than the number of required finite-differences, would require a parallel evaluation of $\nabla f(x)$. This situation is onsidered in [46,47,48].

We do have three concerns. First, there is accuracy. While the accuracy in the Hessian approximation is much better than a quasi-Newton (or secant) method, choosing the appropriate step size, especially in the presence of noise, can be delicate. In this case automatic differentiation may (eventually) provide a more accurate alternative[5](we discuss this in Section 4). Second, while it is practically very useful to have matrix estimation methods that require the user to supply only a gradient subroutine, in some cases it may be more efficient to access the function and the gradient in a more fine-grained manner – e.g., component-wise, or perhaps as a sum of "small" functions (see the section on partial separability below). Finally, the estimated Hessian matrices will be symmetric but not, in general, positive definite. Therefore solving for the Newton direction does not necessarily lead to a descent direction and so a negative-curvature method (discussed above) is appropriate.

C. Partial Separability. Many large-scale optimization problems are presented in a partially separable form:

$$f(x) = \sum_{i=1}^{n_e} f_i(x), \tag{10}$$

where each of the "element" functions f_i has a Hessian matrix of low rank relative to n. Considerable work has been done on the development of large-scale optimization methods especially designed to exploit this form: Philippe Toint, of the University of Namur, Belgium, has been the primary advocate of this approach, e.g., [50,51,52,53]. Certainly this

[5] It is interesting to note that the graph-coloring analogies used in the sparse finite-difference work are also applicable in the automatic differentiation setting [49].

form leads to convenient design of subroutines to evaluate the gradient (and perhaps the Hessian). For example, in Figure 4, we consider the evaluation of the function, gradient, and Hessian in the distance conformation problem. To simplify we consider the special case when the molecule is restricted to lie in a straight line, i.e., the 1-dimensional distance conformation problem.

{Determine function value}
For $k = 1 : n_e$ do {n_e is the number of edges (elements)}
 Let $e_k = (i, j),\ i > j$
 $t^1_k = x_i - x_j$
enddo
$t^2 = t^1 .* t^1$ {pointwise multiplication}
$t^3 = t^2 - d .* d$ {pointwise multiplication, subtraction}
$f = (t^3)^T(t^3)$

{Determine gradient. Initially, $g = 0$.}
For $k = 1 : n_e$ do
 Let $e_k = (i, j), i > j$
 $g_i = g_i + 4t^3(k) \cdot t^1(k)$
 $g_j = g_j - 4t^3(k) \cdot t^1(k)$
enddo

{Determine Hessian. Initially, $H = 0$.}
For $k = 1 : n_e$ do
 Let $e_k = (i, j), i > j$
 $H(i, j) = H(i, j) - 8t^2(k) - 4t^3(k)$
 $H(j, i) = H(i, j)$
 $H(i, i) = H(i, i) + 8t^2(k) + 4t^3(k)$
 $H(j, j) = H(j, j) + 8t^2(k) + 4t^3(k)$
enddo

FIG. 4. *Evaluation of 1-dimensional molecule function, gradient, and Hessian*

Note that to evaluate the function alone requires $5 \cdot n_e$ operations (adds or multiplies), to evaluate the function and the gradient requires $9 \cdot n_e$ operations, and to evaluate the function, gradient, and Hessian requires $14 \cdot n_e$ operations. Therefore, in this case not only does partial separability lead to a convenient form for evaluating f and its first and second derivatives, it leads to an efficient method as well. Note that a single subroutine to evaluate the triple $(f, \nabla f, \nabla^2 f)$, or a leading subset of this triple, enhances efficiency: intermediate quantities can be re-used.

If the element functions all depend on only a few variables – the usual case – then the full Hessian will be sparse and, if desired, can be formed to use in the calculation of a search direction. However, the definition of partial separability implies that each element function has a symmetric low-rank Hessian matrix which does not necessarily imply dependence on only a few variables. Still, if there are only a few such global element functions, i.e., element functions that depend on many of the variables, then it may still be possible to proceed with a direct factorization method using sparsity. To illustrate, consider the following.

Let $H = H_0 + H_1$ where H_0 is sparse and H_1 is dense but is computed as $H_1 = UU^T$, where U is a t-by-n matrix with $t << n$, and $\text{rank}(U) = t$. A system of the form $Hs = r$ can be replaced with:

$$(11) \qquad \begin{bmatrix} H_0 & U \\ U^T & -I \end{bmatrix} \begin{bmatrix} s \\ v \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}.$$

Note that system (11) can be solved using a symmetric sparse factorization, e.g., [54]. Of course if the true Hessian H is not positive-definite then we may wish to use a factorization of the matrix in (11) capable of revealing negative curvature with respect to the Hessian matrix H.

Instead of evaluating the Hessian matrix, it may be advantageous to estimate the elemental Hessian matrices, separately, using dense secant approximations. Each element function is of low rank and so a small dense approximation to each element Hessian is reasonable. (The elemental Hessian approximations can then be added together to form the full Hessian approximation as described above; alternatively, the elemental Hessian matrices can be used in an iterative way to solve for the next search direction, in which case they need not be collected together. A third possibility, in cases where sparsity prevails, is to integrate their summation with a sparse multi-frontal linear solver [54].) Unfortunately the elemental matrix approximations cannot be guaranteed to be positive definite and so the positive definite secant updates cannot be used. This has lead to a serious investigation of alternative updates with the symmetric rank-1 update (SR1) being particularly popular, e.g., [55,56,57].

Exploiting the partial separable structure of many large-scale optimization problems is often an economic approach. However, this approach does not serve every need. We consider three limitations.

First, there are large-scale optimization problems that are not in a partially separable form but which can be solved using sparsity. Consider, for example, the following. Let f_1, f_2, and f_3 each be functions of many variables, each with sparse Hessian matrices. Assume that the sum of the Hessian matrices, $\sum_{i=1}^{3} \nabla^2 f_i$, is sparse. Now consider the problem: $\min_x f(x) = f_1(x) \cdot f_2(x) \cdot f_3(x)$. Clearly f is not in partially separable form. However, it turns out that the Hessian matrix of f can be written:

$$(12) \qquad \nabla^2 f = \nabla^2 f_1 \cdot f_2 f_3 + \nabla^2 f_2 \cdot f_1 f_2 + \nabla^2 f_3 \cdot f_1 f_2 + UTU^T,$$

where $U = (\nabla f_1, \nabla f_2, \nabla f_3)$, and T is a symmetric 3-by-3 matrix,

$$(13) \qquad T = \begin{bmatrix} 0 & f_3 & f_2 \\ f_3 & 0 & f_1 \\ f_2 & f_1 & 0 \end{bmatrix}.$$

Therefore, the Hessian is the sum of a sparse matrix and a symmetric low-rank matrix. The Newton step (for example) can be computed by using a technique similar to (11). Therefore, restriction to the partial separable form is not useful in this case.

Second, the optimization problem may be easily expressed in partially separable form, in principle, but perhaps it is not convenient for the user to do so. For example, the user may wish to begin with a given subroutine to evaluate the function and gradient.

Third, there is the case of global variables. A variable is global, with respect to a given partial separable formulation, if it appears in most of the element functions. If there are several such variables, and they are involved in significant common evaluations in most of the elements, then separate evaluation of element functions may be an inefficient way to evaluate the overall function or gradient.

In summary, the partial separable form arises frequently in large-scale optimization and it is often worthwhile exploiting this. However, it may not always be convenient to express a function in this form and so there will remain a need for optimization approaches that make fewer assumptions on the user-supplied subroutines.

D. Conjugate Gradient and Limited-Memory Secant Methods. For very large problems it may not even be feasible to store a sparse matrix approximation to the true Hessian matrix. In such cases conjugate gradient methods and limited-memory secant methods provide alternatives.

A number of nonlinear conjugate gradient methods have been proposed, e.g., [58,59,60, 61,62,63,64,65], and Fortran codes are publicly available. Their great attraction, beyond simplicity, is their very small memory requirement – in their pure forms several dense vectors are required but no matrices. For example, consider the Polak-Ribière [62] algorithm in Figure 5.

Repeat

$$\beta_k = \frac{(y_{k-1})^T \nabla f(x_k)}{\|\nabla f(x_{k-1})\|^2} \quad \{y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})\}$$
$$d_k = -\nabla f(x_k) + \beta_k d_{k-1}$$
$$x_{k+1} = x_k + \alpha_k d_k$$

FIG. 5. *The Polak-Ribière conjugate-gradient algorithm*

A line search algorithm will determine α_k, the step length. Typical of conjugate gradient methods, this method is easy to program (given a line search routine) and requires little memory – just a handful of n-vectors.

Acceleration of convergence of these procedures can usually occur through the use of clever "restart" procedures and preconditioning stratagies. Unfortunately, the skillful use of these techniques is very much an art usually requiring some experience with the problem at hand. Nocedal [66] has done some numerical experimentation and comparisons amongst several nonlinear conjugate gradient codes: he reports his results and gives tentative conclusions and advice.

An alternative to a nonlinear conjugate gradient method is a limited-memory secant method, e.g., [67,68,69]. A limited-memory secant approximation does not update an explicit matrix approximation; instead, an initial sparse approximation to the inverse Hessian is saved (usually a diagonal matrix) along with a small collection of vectors representing the recent updates. This set is typically updated at each iteration by deleting vectors corresponding to old information and adding new. Note that since the density of the approximating matrix is not an issue – it is not formed explicitly – the very successful dense positive-definite secant update formulae can be used. Of course this is off-set to some extent by the fact that only a small set of updates can be used. Nash and Nocedal [70] report on numerical experiments, comparing limited-memory quasi-Newton codes to (truncated) Newton methods.

E. Concluding Remarks on Unconstrained Problems. We conclude this section with a few remarks on the state-of-affairs with respect to the large-scale unconstrained optimization problem.

First, it is clear that there has been much progress in this area over the past ten years. Work is continuing – and further research is definitely needed – but several alternatives are now available. Indeed, software packages especially tailored to the large-scale user are available, e.g., [50,71]. Of course with a variety of methods comes the question: which to choose? Expect no clear definitive answer to this question for a long time, if ever. What we can offer are some (personal) guidelines.

The methods that are most robust and can achieve the greatest accuracy are the methods that compute second derivatives (or use finite-difference Hessian approximations) and that use sparse matrix factorizations with the ability to detect and use negative curvature. Computing second derivatives is feasible for many large-scale optimization problems. Such methods are particularly good for ill-conditioned problems and problems with much negative curvature. On the down side, the memory requirements may still be too much despite using sparsity, and some problems can be solved just as well with less effort. Methods that compute second derivatives (or use finite-differences) and use an iterative technique to solve the linearized problem, e.g., conjugate gradients (CG) , and that have the ability to deal reasonably with negative curvature, provide an alternative if the direct factorization is too expensive in space or time, e.g., [72,73]. Of course the effectiveness of the CG-iteration is problem dependent and may need preconditioning – a heuristic situation. In terms of overall reliability and robustness, the elemental secant methods follow the second-derivative methods (if the problem is expressed in the partially separable format) followed by the limited-memory secant and nonlinear congugate gradient routines. (Of course this list can be read in reverse order: starting from the most inexpensive methods at the bottom (if they work!) to the most expensive at the top.)

The sparse secant updates, i.e., those that do not exploit the partially-separable structure, are too inconsistent in behaviour to include in our list; however, they still provide a tool that can be useful on occasion.

A final remark: this list emphasizes the need for a computing environment that allows for experimentation within the large-scale optimization context. It is clear that the job of selecting a suitable approach to a large-scale optimization problem is not an easy one and is certainly not easily done without a trial-and-error process. This is very difficult to do today, without a major time investment, due to the lack of a flexible computing environment for large-scale optimization.

III. Constrained Optimization. The general constrained optimization problem we consider is

$$\min_{x} \{f(x) : \; c(x) = 0, \; l \le x \le u\}, \tag{14}$$

where f is a twice continuously-differentiable mapping from R^n to R^1, c is a twice continuously-differentiable mapping from R^n to R^m, with $m < n$, and l, u are vectors of lower and upper bounds respectively, $l \le u$. (For any i, if $l_i = -\infty$, then x_i is unbounded below; if $u_i = \infty$ then x_i is unbounded above.) The region defined by the constraints, $\{c(x) = 0, \; l \le x \le u\}$, is coined the feasible region; we denote this region $\mathcal{F}$. A problem may have no bounds on the variables, i.e., $u = \infty$, $l = -\infty$, in which case the optimization problem reduces to an equality-constrained problem: $\min_x\{f(x) : \; c(x) = 0\}$.

There are other formulations of the general problem. For example, a problem may include constraints of the form $c(x) \ge 0$; however, simple transformations – such as the introduction of slack variables – can yield the form given in (14) and so, for simplicity, we assume this formulation.

Again we restrict ourselves to methods for finding a local minimizer. In the constrained case this means we look to find x_* such that there exists a scalar $\delta > 0$, where δ, x_* satisfy:

$$f(x_*) \leq f(x), \quad \forall x \in \{x : \|x - x_*\| < \delta, \; x \in \mathcal{F}\}. \tag{15}$$

Most of the issues and concerns in unconstrained minimization carry over to the constrained case. For example, the two basic globalization strategies for unconstrained problems, line search and trust-region, have counterparts for constrained problems (though each has more complications in the presence of constraints).

In a certain sense constraints make a problem easier by restricting the domain of interest. However, constraints do complicate things in several ways.

First, handling constraints and exploiting sparsity are two objectives that can be difficult to achieve simultaneously. For example, a common approach to a problem with linear (or linearized) constraints is to determine a basis for the null space of the matrix corresponding to the linear (linearized) constraints and then work in the null space, e.g., [7]. To illustrate, consider the problem, $\min_x \{f(x) : Ax = 0\}$. If the columns of the matrix Z form a basis for the null space of A, then a "reduced Newton system" is given by

$$(Z^T \nabla^2 f Z)\bar{s} = -Z^T \nabla f, \tag{16}$$

and the reduced Newton step is $s = Z\bar{s}$. This is a powerful methodology in constrained optimization. However, it is difficult to adapt to a large-scale problem with the additional hope of maintaining sparse matrices. There has been some successful work on understanding the problem of determining a sparse basis for the null space of a sparse matrix [74,75, 76], i.e., finding a sparse matrix Z in (16); however, this is only part of the problem. A direct factorization method requires a representation of the reduced Hessian matrix, matrix $Z^T \nabla^2 f Z$ in (16), and if sparsity is also desired this is a difficult request to satisfy. Of course an iterative null space approach can be followed without explicitly forming $Z^T \nabla^2 f Z$ – i.e., in (16) we need not form the matrix $Z^T \nabla^2 f Z$ but instead save the sparse matrices, $Z, \nabla^2 f$, and apply them, in sequence, to perform multiplication within an iterative solver. Nevertheless, the point is our options have been limited.

A second difficulty caused by the presence of constraints occurs when the constraints are nonlinear. The difficulty stems from the fact that most computational mathematics is linear; hence, a nonlinear manifold is usually difficult to follow. Algorithms that try to maintain feasibility, in the general nonlinearly constrained problem, can be very slow – spending a lot of time trying to follow nonlinear boundaries. To see the usefulness of linearity, consider the nonlinear equality-constrained problem:

$$\min_z \{f(z) : \; c(z) = 0\}. \tag{17}$$

In general it is difficult to restrict the iterates $\{z_k\}$ to be feasible, i.e., $c(z_k) = 0, \; k = 1, 2, \ldots$. However, if there is sufficient linearity then it is possible. For example, suppose the variables z can be partitioned $z = (y, x)$ such that $c(z) = 0$ can be equivalently written:

$$T(x) \cdot y + \bar{c}(x) \;=\; 0, \tag{18}$$

where $T(x)$ is a nonsingular matrix for all x. Then, y appears in a linear role and can be "eliminated" via the equation:

$$y \;=\; -T(x)^{-1}\bar{c}(x). \tag{19}$$

Therefore, problem (17) can be reduced to the following unconstrained problem:

$$\min_{x} f(y(x), x) \tag{20}$$

where x is the "independent" vector and y depends on x through (19). An algorithm for unconstrained minimization can be used to solve (20), and the nonlinear manifold, $c(z) = 0$, will be satisfied (implicitly) at every iteration. The key to the reduction of (17) to (20) is the linear role[6] played by the vector y.

An alternative to an algorithm that maintains feasibilit is one that ignores it (except in the limit). This leads to two concerns. First, there is the question of how to force convergence from a distant point and what constitutes an improvement – hence the need for "merit" functions (and associated difficulties). Penalty and barrier functions are typical merit functions (see, for example, [4,5,7,6]). Second, in practical problems functions sometimes are not defined outside the feasible region – penalty or exterior methods clearly have difficulties in such cases.

A third difficulty due to the presence of constraints is specifically related to inequalities (in our formulation (14), this refers to finite bounds on the variables). Inequalities give a combinatorial flavor to the problem which can cause some algorithms to require many iterations when the dimension is high. In addition, inequality constraints can increase the complexity of dealing with sparsity and can make the design of efficient parallel algorithms difficult.

There are two basic categories of algorithms for problems with inequalities. These are "active-set" algorithms and "passive-set" algorithms. (The first term is in common usage; the second is less common – we use it to refer to algorithms that are not "active-set"!) In each iteration of an active-set algorithm the inequality constraints are divided into two sets. The set of constraints currently satisfied exactly defines the active set $\mathcal{A}$: $\mathcal{A}(x) = \{i : x_i = l_i, \ \ or \ \ x_i = u_i\}$. The complementary set, the set of constaints corresponding to variables not at their bounds, is the set of inactivities. In active-set algorithms, the determination of the "next step" depends on these two sets.

In contrast, a "passive-set" algorithm does not permit variables to be at their bounds except in an asymptotic sense, i.e., in the limit as $k \to \infty$, where k is the iteration counter.

There are many examples of successful algorithms in both camps, active-set and passive-set. The simplex method for linear programming, e.g., [77], is the most prominent example in the active-set category; smooth penalty and barrier methods illustrate the passive-set approach. Note that an algorithm that is an active-set algorithm need not be restricted to a feasible-point algorithm, e.g., [78].

Passive-set algorithms appear well suited to large-scale problems: they typically require fewer iterations than many of their active-set counterparts and can be implemented using fixed data structures – very convenient for exploiting sparsity and parallelism. Some active-set algorithms are poorly suited to the large-scale setting – typically, those that restrict or inhibit change in the activity set from one iteration to the next do not fare well on large problems.[7] However, recently there has been considerable investigation into active-set methods that allow for significant change in the active set from one iteration to the next, and these methods do appear much more promising for large-scale problems.

[6] Example 3 in the Introduction illustrates this type of "elimination".

[7] A glaring counter-example to this statement is the simplex method which restricts change in the activity set to a single variable each iteration. This strategy does not work well for more general (nonlinear) functions. Neither does it work well for some linear programming problems with special structure such as staircase problems.

In summary, the presence of constraints, despite decreasing the volume of the search space, tends to create more difficult problems (especially in the large-scale case when maintaining sparsity is important). Nevertheless, there has been significant progress lately – especially with respect to simple bound constraints and linear/quadratic programming – and our purpose here is to summarize and highlight some of the progress as well as attempt to give pointers to the future.

A. Minimization Subject to Bounds. Many optimization problems are constrained in a simple way :

$$\min_x \{f(x) : \ l \leq x \leq u\}, \tag{21}$$

where $l \leq u$. This is sometimes referred to as a "box-constrained" problem.

Recently there has been considerable research effort on box-constrained problems, aimed at developing new algorithms, and refining old ones, especially with an eye toward large-scale sparse problems. Some of this work has been targeted specifically at the case where the objective function is quadratic. We will begin there.

Let $f(x) = g^T x + \frac{1}{2} x^T H x$, where H is a symmetric matrix; hence, our problem is:

$$\min_x \{g^T x + \frac{1}{2} x^T H x : \ l \leq x \leq u\}. \tag{22}$$

Clearly if the set of variables bound at the solution is known, i.e., if $\mathcal{A}(x) = \mathcal{A}(x_*)$, and the tight variables are at their appropriate bounds, then a reduced Newton system will find the solution in a single step:

$$solve\ H_R s_R = -g_R, \quad update : \ x_R^+ = x_R + s_R, \tag{23}$$

where the subscript R denotes the "free" variables (i.e., those variables not at bounds). The matrix H_R consists of those rows and columns of H identified by the free variables. The idea behind the Moré and Toraldo algorithm [79], building on previous work by Dembo [80], and Bertsekas [81], is to mix steps that attempt to "guess" the active set at the solution with reduced Newton steps. The reduced Newton steps (23) are not solved for exactly: Moré and Toraldo propose a conjugate gradient "inner-iteration" for the inexact determination of this reduced Newton step. If a boundary is hit in the process then an efficient piecewise linear minimization is followed.

Therefore, the Moré-Toraldo algorithm is an active-set method with the ability to completely change the activity set in a single iteration – this is important for a large-scale problem. Coleman and Hulbert [82] suggest an alternative active-set algorithm in which they show how to compute and use direct sparse factorizations using the space required by the sparse Cholesky factor of a positive-definite matrix with the structure of the original matrix H.

Alternative strategies in the passive-set vein, also suitable for the large sparse case but currently restricted to the situation where H is positive definite, have been suggested by Han, Pardalos, and Ye [83] and Coleman and Hulbert [84]. These two approaches are entirely different – the Han-Pardalos-Ye method is an interior-point algorithm following the work in [85]; the Coleman-Hulbert algorithm is an exterior-point algorithm based on a Newton approach to the optimality conditions. However, the two methods are similar from the linear algebra point of view. Both methods require the repeated solution to a (short) sequence of positive-definite systems, all with the same structure as the original matrix H.

Work on extending these approaches to problems with indefinite matrices, and to problems with linear constraints, is now continuing.

Adaptation of the methods mentioned above to problems with general nonlinear objective functions is under investigation. Other related methods, again especially targeted toward the large-scale case, have also been proposed recently, e.g., [55,56,86,87,88].

What makes these newly proposed methods attractive for large-scale problems? First, they all try to avoid excessive number of iterations due to the combinatorial aspect of the problem (caused by the inequalities). Second, these methods pay some respect to the problem size by either considering iterative linear solvers or sparse direct factorizations. Finally, in several of the cases there has been some attention paid to the use of parallelism.

B. Linear Constraints (and Bounds). Here we consider problems of the form:

$$\min_x \{f(x): \; Ax = b, \; l \leq x \leq u\}. \tag{24}$$

There has been recent research activity on this problem, especially in the case where f is a quadratic function. Again the basic approaches generally fall into two categories: active-set or passive-set. Many of the basic approaches used in the box-constrained case, discussed above, can be adapted to this more general situation. The algorithms for (24) are more complex, of course, due to the need to project onto the linear manifold $Ax = b$.

Three linear algebra situations play a prominent role in most of the proposed large-scale algorithms for problem (24). We will center our discussion around this aspect.

First, there is the weighted least-squares problem. This problem is central to virtually all of the new passive-set algorithms for linear programming[8] including interior-point, barrier, and exterior-point algorithms. Some examples of work in this area: [89,90,91,92, 93,94,95,96,97,98,99,100]. Several important related problems can also be efficiently solved using a sequence of weighted least-squares problems: linear l_1 estimation[101,102], ; linear l_∞ estimation[103]; the linear p^{th}-norm problem [104].

These various passive-set methods for linear programming have strikingly different computational and theoretical properties; however, from a linear algebra point of view they are quite similar. In particular, a sequence of least-squares problems of the form

$$D_k^{-1} A^T s_k \overset{\text{l.s.}}{=} -D_k g \tag{25}$$

must be solved; a primary aspect in which the algorithms differ is in the choice of diagonal weighting matrices D_k. These new passive-set algorithms have, in turn, increased interest in methods for solving a sequence of sparse least-squares problems including direct (sparse) factorization methods, parallel solvers, and iterative least-squares solvers, e.g., [47,105,106, 107,48].

A second linear algebra problem arising in some of the new algorithms for large-scale problems with linear constraints, is the solution of reduced Newton systems of the form (16). In some cases this involves determining a basis Z for the null space of a matrix A such that the reduced Hessian matrix, $Z^T \nabla^2 f Z$, is sparse and can enjoy a sparse factorization. This works when the structure of $\nabla^2 f$ is very sparse and simple, e.g., [108]. However, this does not appear to be a promising avenue for general problems: construction of a sparse (or compactly represented) matrix Z and the design of an efficient iterative solver for (16) has more potential for general problems. Research issues include the development

[8] Linear programming is a special case of (24) in which the objective function is linear: $f(x) = g^T x$.

of efficient preconditioning strategies and how to deal with bounds on the variables (and perhaps changing activity sets).

The third linear algebra problem is the solution of sparse symmetric indefinite systems of the form:

$$\begin{bmatrix} H & A^T \\ A & -D \end{bmatrix} \begin{bmatrix} s \\ r \end{bmatrix} = \begin{bmatrix} -g \\ 0 \end{bmatrix}, \tag{26}$$

where D is a positive diagonal matrix (in some cases D is the identity, or a scaled identity matrix; in some cases D is the zero matrix). Note that the least-squares problem (25) can be solved using a system of the form (26) – the reader can work out the details. Clearly any sparsity that was present in the original matrices H and A shows up in this augmented form (26), and that is the attraction of this approach. The dimension of the system is large, $m + n$, but this is often more than compensated for by the sparseness of the matrices. Unfortunately, the matrix in (26) is not positive definite; hence, the arsenal of techniques developed for sparse symmetric positive-definite systems is not applicable. Therefore, renewed interest is being shown in new and old techniques, direct and iterative, for solving sparse symmetric indefinite systems in general, and systems with the block structure evident in (26) in particular,e.g., [109,54,110]. Beyond solving systems of the form (26), optimization has other concerns: determination (and use) of directions of negative curvature, guaranteeing descent properties, trust-region implementations, and the affect of the bound constraints (in active-set methods consideration must be given to the modification of a sparse factorization when the activity set changes, e.g., [111]).

The ease with which a large-scale optimization algorithm can be integrated with an appropriate parallel sparse linear solver, to produce an efficient over-all procedure, will play an important role in determining the ultimate utility of proposed large-scale optimization algorithms in parallel computing environments. Certainly many of the new passive-set methods appear to be leading contenders since their interface with parallel linear solvers is exceedingly simple – just plug it in. In contrast, some other large-scale methods, such as the simplex method for linear programming, appear difficult to adapt to the parallel world. Certainly a simple interface with a "black-box" sparse linear solver would not produce an efficient overall procedure in this case.

C. Nonlinearly Constrained Problems. Effective procedures for problems with nonlinear constraints, even for the dense case, are still being heavily researched. Certainly the importance of sequential quadratic programming (SQP) – in which a sequence of quadratic approximations is solved (perhaps approximately) – is well-established, e.g., [112,113,114,115,116,117]. Therefore the work on large-scale quadratic programming,

$$\min_s \{\nabla f(x_c)^T s + \frac{1}{2} s^T H_c s : \quad As = 0, \quad l_c \leq s \leq u_c\} \tag{27}$$

is directly relevant, e.g., [118,111]. However, globalization stratagies are still quite unsettled, with centers of gravity continually shifting from l_2-penalty functions, to l_1-penalty functions, to barrier functions, to augmented Lagrangian approaches, and around again. Despite this apparent thrashing, there is progress: for example, it is now clear that the l_2-penalty function can be used much more effectively than was thought possible just a few years ago. Some of the numerical difficulties caused by ill-conditioning can be overcome to a large degree, e.g., [119,120].

Still, despite this uncertainty as to the the best designs of overall methods, it is clear that the main linear algebra concerns are the same as for the linearly-constrained problems discussed above (due to linearization), with increased emphasis on approximate solutions. So the progress on solving large linear systems of the form (26) is directly applicable.

IV. Concluding Remarks. In this article we have given the reader a quick personal tour of large-scale optimization. This field is enjoying a surge of research activity with several new trends; we have tried to capture some of them and explain some of the central concerns of large-scale optimization.

We conclude with a discussion of two developments, each with so much potential they may change the face of large-scale optimization altogether: they are automatic differentiation and parallelism.

A. Automatic Differentiation. Evaluation of the function, gradients, and possibly second-derivatives, often represents a significant portion of the overall computational cost of an optimization procedure. Moreover, hand-coding of derivative routines is an error-prone and tedious task; numerical approximations carry with them numerical errors. Automatic differentiation potentially offers an error-free, automatic, and efficient alternative. There still are some problems – summarized in our concluding paragraph in this section – but the recent flurry of research activity and related results, e.g., [121,122,49,123,124,125,126,127] suggests great promise. The potential impact on large-scale optimization work is significant.

Automatic differentiation (AD) works like this. The user presents a description of the function to be minimized (and constraints) to the AD-compiler. This is usually in the form of a Fortran or C program. The automatic compiler then produces a new program (usually in C or Fortran) that will evaluate the function and the gradient (and the Hessian if requested) at any point x in a single efficient routine. The code produced by the compiler is efficient: efficiency is achieved by calculating intermediate quantities in a good order and by saving and re-using intermediate quantities in the function, gradient, and Hessian calculations.

Automatic differentiation is distinct from symbolic differentiation. A symbolic differentiator will accept the same input as the automatic differentiator, but will produce code that represents the gradient (and Hessian if requested). The function evaluation routine is unchanged – it remains the user-written code. No account is made of efficient computation of the pair $\{f(x), \nabla f(x)\}$, or the triplet $\{f(x), \nabla f(x), \nabla^2 f(x)\}$. Intermediate quantities are not re-used; typically, there is no bound on the accuracy of the resulting computations when the gradient (or Hessian) routines are used (in contrast to automatic differentiation).

The automatic differentiator can also reveal parallelism that can be used in the function, gradient, and Hessian evaluations. This is because the computational graph that is used in the AD-compiler to determine the order of computation naturally exhibits the available parallelism as well. Researchers in automatic differentiation are already investigating this aspect.

The basic idea behind automatic differentiation is repeated application of the chain rule and saving intermediate quantities along the way. Indeed it has been argued [122] that instead of actually forming the Newton system it may be more efficient to solve a larger but sparser adjoint system that involves the intermediate quantities treated as temporary variables. This is similar to the techniques we used in (11) and (12)-(13).

There are still problems to be worked out. The first is a subtle one (and may not admit a solution). The code produced by the automatic differentiator is dependent on the form of the user-supplied function routine. Everything follows from this. However, different user-supplied encodings of the same function will yield different ouputs from the AD-compiler.

It appears these outputs may differ widely and it is certainly not clear how to best encode the function originally.

Other problems: the space required by the automatic differentiation process (saving all those intermediate quantities!) may be quite substantial; the AD-compiler itself is a large and complex code begging portability and ease-of-use issues; finally, can sparsity be fully and cleanly exploited? In our view these issues are extremely important for ultimate utility in the large-scale optimization context. Research activity is ongoing on these fronts and we can expect good progress in the next few years.

B. Parallel Computation. Parallel computation will soon play a major role in much of large-scale computational mathematics. Is optimization poised to effectively utilize this new and powerful technology? What are the research issues specifically relating parallel computation and large-scale optimization?

In order to see what role parallelism can play in large-scale optimization it is useful to both broaden and narrow our focus. On the broader scale it should be remembered that local minimization occurs in a context – this broader context often involves the solution of many independent minimizations that can be done in parallel – this is easy coarse-grained parallelism and yields very high parallel efficiency. Examples in this domain: global optimization typically involves the solution of many local disjoint minimization problems, e.g., [18,21]; optimization problems with integrality constraints can be solved using various branch-and-bound techniques involving local nonlinear minimizations, e.g., [13]. In both these situations some communication and orchestration is required; however, the computational work in the local minimizations easily dominates communication/global orchestration costs by several orders of magnitude. These are important problem classes that typically yield many disjoint local minimization problems; however, the most common is probably the following. In many cases the objective function to be minimized is not really known precisely – often the form is known (or accepted) but there are several unknown parameters. These parameters are usually determined only after a lengthy trial-and-error period involving many independent minimizations – another perfect candidate for coarse-grained parallelism.

If we now sharpen the focus and look within an optimization procedure, we see the possibility of using parallelism[9] in both linear algebra computations and in the evaluation of function, gradients, and Hessians, e.g., [131,132,46,47,133,48]. There are some subtleties here that have optimization implications and so we must examine this possibility more carefully.

First, it is clear that if the function and derivative evaluations dominate the computational costs then parallelism must begin there. A good example of this is the bone modelling problem we discussed in Example 3. In this example, on realistic problems, the finite-element analysis (i.e., the function evaluation) can take in excess of 90% of the serial computational time. Of course the difficulty here is that this parallelization effort – organize the function evaluation to compute the function efficiently in parallel – falls into the hands of the user in general. General guidelines can be given but beyond that it will usually be up to the user. (It may be possible to design efficient parallel strategies for some very common function forms, e.g., partially separable functions. It may also be that automatic differentiation will save the day.)

[9] The literature on the parallel solution of problems with special structure is extensive, e.g.,[128,129,130], but falls outside the domain of this article.

Another issue is brought up by this bone reconstruction example. Despite the dominance of the overall computational cost by the function evaluation, it is incorrect to infer that the selection of the optimization routine is relatively unimportant. It is true that the computational cost of the linear algebra directly involved in the determination of a search direction may be (relatively) insignificant in this case; however, the choice of algorithm can dramatically influence the number of function evaluations and therefore a judicious choice can be most significant. For example, for a sufficiently large problem with inequalities some active-set methods will be a poor choice. Due to the combinatorial sensitivity of some active-set methods there may be a significant growth in the number of iterations – and therefore function evaluations – as the dimension grows. Passive-set methods, on the other hand, are more likely to require fewer (more expensive) iterations, and therefore fewer function evaluations. Moreover, in a parallel computing environment, it is difficult to effectively orchestrate the changing activity sets and corresponding matrix dimensions of an active-set method. Passive-set methods typically deal with a fixed dimension and fixed matrix structures and are therefore more likely to yield efficient parallelism. Finally, an algorithm that uses exact second-derivatives may also significantly decrease the number of function evaluations. If the second-derivatives can be computed efficiently, using parallelism, then this extra cost per iteration (over a method that uses only first derivatives) may be cost-effective.

To emphasize: the availability of a parallel computer should influence the choice of serial optimization algorithm to be used in a parallel computing environment. A serial algorithm that yields more work that can be effectively computed in parallel, perhaps at some increased serial cost per iteration, is to be preferred if the number of serial outer-iterations decreases significantly.

Linear algebra considerations may also affect the choice of serial optimization algorithm used in a parallel computing environment. For example, in the serial world the unconstrained minimization of nonlinear functions is often performed by the positive definite secant update (BFGS). The modern choice is to update the Cholesky factor of the current approximation in a stable and efficient manner requiring $O(n^2)$ work. Therefore, the linear algebra involved per-iteration is quite acceptable : $O(n^2)$ work for the Cholesky update, and $O(n^2)$ for the triangular solve. Unfortunately, this may not work well on a parallel machine. The difficulty is that known stable algorithms for these two steps exhibit limited parallelism. Couple this with the low-order of work required, within each iteration, and it is difficult to achieve good parallel efficiency, e.g., [132,133]. It may be better to reconsider methods for updating the the inverse approximation in this case , e.g., [131] – the work is still $O(n^2)$ but the algorithms required exhibit much more parallelism. (Of course this option is not viable for sparse problems since the inverse of a sparse matrix is almost always dense, e.g., [134].)

Optimization problems with large dense matrices exist, e.g., Example 3; for these we can expect to do well in the linear algebra arena, given the emphasis the linear algebra community has placed on developing high-performance dense linear algebra routines. However, most large-scale optimization algorithms do not exhibit large dense matrices. Much more common are large sparse matrices and large structured systems. Large-scale optimization needs effective high-performance algorithms for these classes of matrices.

Certainly there is considerable work on special structures as well as general sparse systems. However, these are difficult problems and progress is slow – more research work is needed.

Nevertheless, it is expected that reasonable success will be obtained in our ability to obtain high-performance parallel algorithms for standard factorizations of large sparse matrices and large matrices with a variety of standard structures. Therefore, serial optimization algorithms that plug into efficient parallel versions of standard linear algebra tasks – for example the sparse QR-factorization – will yield better overall performance than algorithms that use specialized low-order sparse matrix updating. We expect passive-set algorithms to ultimately yield better performance on parallel computers than active-set methods, for example.

In summary, numerical optimization and parallelism appear to intersect on two levels. At the high end, there is coarse-grained parallelism for problems that yield numerous disjoint local minimizations. This is typically easy parallelism to get – though research on effective orchestration, communication, load balancing is ongoing and still needed, and is very important. Obtaining parallelism for optimization at the low end – linear algebra (especially sparse and/or structured systems) and function and derivative calculations – is also ongoing and yields both fine-grained and medium-grained parallelism. The (local) optimization challenge consists of designing serial optimization algorithms that can effectively use the parallelized linear algebra and function evaluation subroutines.

It is interesting to note that we have said nothing about the design of general purpose parallel local minimization algorithms per se. The parallelism we have discussed encircles the local minimization procedure or is internal to it, but the optimization algorithm itself remains serial – typically some version of Newton's method (an inherently serial process). The general purpose parallel optimization research to date has largely been concerned with the design and modification of serial algorithms that can effectively use parallelism at either of these two levels. Barring a breakthrough – say a truly parallel Newton process – this will continue to be the case.

Acknowledgements. I would like to thank a number of my Cornell colleagues for suggesting improvements to an earlier version of this paper. Specifically, I thank Moshe Braner, Shirish Chinchalkar, Yuying Li, Jianguo Liu, Michael Todd, Zhijun Wu, and Wei Yuan.

REFERENCES

[1] Thomas F. Coleman and Yuying Li. *Large-scale numerical optimization.* SIAM, 1990.

[2] Thomas F. Coleman. *Large Sparse Numerical Optimization.* Lecture Notes in Computer Science, Volume 165, Springer-Verlag, 1984.

[3] John E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization.* Prentice-Hall, 1983.

[4] A. V. Fiacco and Garth P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley and Sons, 1968.

[5] R. Fletcher. *Practical Methods of Optimization, Second Edition.* John Wiley and Sons, 1987.

[6] Garth P. McCormick. *Nonlinear Programming. Theory, Algorithms, and Applications.* John Wiley and Sons, 1983.

[7] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

[8] R. Fletcher. Recent developments in methods for nonlinear programming. Technical Report NA/123, University of Dundee, 1990.

[9] G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd. *Optimization, Volume 1 of Handbooks in Operations Research and Management Science.* North–Holland, 1989.

[10] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation.* Prentice-Hall, 1989.

[11] S.A. Zenios. Parallel optimization: current status and an annotated bibliography. *ORSA J. Computing*, 1:20–43, 1989.

[12] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[13] Zhijun Wu. A subgradient algorithm for nonlinear integer programming and its parallel implementation. Technical Report TR91-09, Dept. Math. Sci., Rice University, 1990.

[14] Bruce A. Hendrickson. Conditions for unique graph embeddings. Technical Report CS-88-950, Cornell University, 1988.

[15] Bruce A. Hendrickson. *The molecule problem: Determining conformation from pairwise distances.* Cornell University Ph.D thesis, Computer Science, 1991.

[16] Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H. Freeman, San Fransisco, 1979.

[17] Richard H. Byrd, C.L. Dert, A.H.G. Rinnooy Kan, and Robert B. Schnabel. Concurrent stochastic methods for global optimization. *Mathematical Programming*, 46(1):1–29, 1990.

[18] E. Eskow and R. B. Schnabel. Mathematical modeling of a parallel global optimization algorithm. Technical Report CU-CS395-88, University of Colorado, Boulder, 1988.

[19] A.H.G. Rinnooy and G.T. Timmer. A stochastic approach to global optimization. In P. Boggs, R. Byrd, and R.B. Schnabel, editors, *Numerical Optimization*, pages 245–262. SIAM, 1984.

[20] A. Torn and A. Zilinskas. Global optimization. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science.* Springer-Verlag, 1989.

[21] Sharon L. Smith, Elizabeth Eskow, and Robert B. Schnabel. Adaptive, asynchronous stochastic global optimization algorithms for sequential and parallel computation. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 207–227. SIAM, 1990.

[22] Y. Censor and G.T. Herman. On some optimization techniques in image reconstruction from projections. *Appl. Num. Math.*, 3:365–391, 1987.

[23] G.T. Herman. *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography.* Academic Press, 1980.

[24] G. T. Herman. *Image Reconstruction from Projections: Implementation and Applications.* Springer, Berlin, 1979.

[25] G. T. Herman, D. Odhner, K. D. Toennies, and S. A. Zenios. A parallelized algorithm for image reconstruction from noisy projections. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 3–21. SIAM, 1990.

[26] G. Subbarayan and D. L. Bartel. A variational model for bone construction/reconstruction and its applications. Technical report, Sibley School of Mechanical and Aerospace Engineering, 1989.

[27] G. Subbarayan and D. L. Bartel. VSAFE: a program for variational sensitivity analysis using the finite element discretization. Technical report, Sibley School of Mechanical and Aerospace Engineering, Cornell University, 1989.

[28] D. M. Gay. Computing optimal locally constrained steps. *SIAM Journal on Scientific and Statistical Computing*, 2:186–197, 1981.

[29] Jorge J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4:553–572, 1983.

[30] Richard H. Byrd and Robert B. Schnabel. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Mathematical Programming*, 40:247–263, 1988.

[31] Gerald A. Schultz, R. B. Schnabel, and Richard H. Byrd. A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties. *SIAM Journal on Numerical Analysis*, 22(1):47–67, 1985.

[32] John E. Dennis and Jorge J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19:46–89, 1977.

[33] M.J.D. Powell and Ph. L. Toint. A note on quasi-Newton formulae for sparse second derivative matrices. *Mathematical Programming*, 20:144–151, 1981.

[34] M.J.D. Powell and Ph. L. Toint. The Shanno-Toint procedure for updating sparse symmetric matrices. *IMA Journal of Numerical Analysis*, 1:403–413, 1981.

[35] D. F. Shanno. On the variable metric methods for sparse Hessians. *Math. Comp.*, 34:499–514, 1980.

[36] D. C. Sorensen. An example concerning quasi-newton estimation of a sparse Hessian. *SIGNUM Newsletter*, 16:8–10, 1981.

[37] Ph. L. Toint. On sparse and symmetric updating subject to a linear equation. *Math. Comp*, 32:839–851, 1977.

[38] Ph. L. Toint. A note on sparsity exploiting quasi-Newton methods. *Mathematical Programming*, 21:172–181, 1981.

[39] Ph. L. Toint. A sparse quasi-Newton update derived variationally with a non-diagonally weighted Frobenius norm. *Math. Comp*, 37a:425–434, 1981.

[40] A. Curtis, M.J.D. Powell, and J. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.

[41] Thomas F. Coleman and Jorge J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20:187–209, 1983.

[42] Thomas F. Coleman and Jorge J. Moré. Software for estimating sparse Jacobian matrices. *ACM Transactions on Mathematical Software*, 10:329–345, 1984.

[43] Thomas F. Coleman and Jin-Yi Cai. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM Journal on Applied Mathematics*, 7:221–235, 1986.

[44] Thomas F. Coleman and Jorge J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Mathematical Programming*, 28:243–270, 1984.

[45] M.J.D. Powell and Ph. L. Toint. On the estimation of sparse Hessian matrices. *SIAM Journal on Numerical Analysis*, 16:1060–1074, 1979.

[46] Thomas F. Coleman and Paul E. Plassmann. Solution of nonlinear least-squares problems on a multiprocessor. In G.A. van Zee and J.G.G. van de Vorst, editors, *Parallel Computing 1988, Shell Conference Proceedings*. Springer-Verlag, Lecture Notes in Computer Science, 384, 1989.

[47] Thomas F. Coleman and Paul E. Plassmann. A parallel nonlinear least-squares solver: Theoretical analysis and numerical results. *SIAM Journal on Scientific and Statistical Computing*, to appear.

[48] P. E. Plassmann. Sparse Jacobian estimation and factorization on a multiprocessor. In T. F. Coleman and Y. Li, editors, *Large-Scale Numerical Optimization*, pages 152–179. SIAM, 1990.

[49] Andreas Griewank. The chain rule revisited in scientific computing. Technical Report MCS-P227-0491, Argonne National Laboratory, 1991.

[50] A. R. Conn, N.I.M. Gould, and Ph.L. Toint. An introduction to the structure of large scale nonlinear optimization problems and the Lancelot project. In *Computing Methods in Applied Sciences and Engineering*, pages 42–54. SIAM, 1990.

[51] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312. Academic Press, 1982.

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

[53] A. Griewank and Ph. L. Toint. On the existence of convex decompositions of partially separable functions. *Mathematical Programming*, 28(1):25–50, 1984.

[54] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[55] Andrew R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25:433–460, 1988.

[56] Andrew R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430,

1988.

[57] H. Khalfan, R.H. Byrd, and R.B. Schnabel. A theoretical and experimental study of the symmetric rank one update. Technical Report CU-CS-489-90, Dept. of Computer Science, University of Colorado, 1990.

[58] A. Buckley and A. LeNir. QN-like variable storage conjugate gradients. *Mathematical Programming*, 27:155–175, 1983.

[59] A. Buckley and A. LeNir. BBVSCG - A variable storage algorithm for function minimization. *ACM Transactions on Mathematical Software*, 11/2:103–119, 1985.

[60] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 6:163–168, 1964.

[61] P. E. Gill and W. Murray. Conjugate-gradient methods for large-scale nonlinear optimization. Technical Report SOL79-15, Dept. Operations Research, Stanford University, 1979.

[62] E. Polak and G. Ribière. Note sur la convergence de methodes de directions conjugées. *Rev. Francaise Informat. Recherche Operationelle*, 16:35–43, 1960.

[63] M. J. D. Powell. Nonconvex minimization calculations and the conjugate gradient method. In D.F. Griffiths, editor, *Numerical Analysis Proceedings.* Springer Verlag, 1984.

[64] D. F. Shanno. Conjugate-gradient methods with inexact searches. *Math. of Oper. Res.*, 3:244–256, 1978.

[65] D. F. Shanno and K.H. Phua. Remark on algorithm 500: minimization of unconstrained multivariate functions. *ACM Trans. on Mathematical Software*, 6:618–622, 1980.

[66] Jorge Nocedal. The performance of several algorithms for large scale unconstrained optimization. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 138–151. SIAM, 1990.

[67] R. Fletcher. Low storage methods for uncosntrained optimization. Technical Report NA/117, University of Dundee, 1990.

[68] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. Technical report, Dept. Electrical Engineering and Computer Science, Northwestern University, 1988.

[69] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[70] S. G. Nash and J. Nocedal. A numerical study of the limited memory BFGS method and the truncated Newton method for large scale optimization. *SIAM Journal on Optimization*, 1:358–372, 1991.

[71] B. A. Murtagh and M. A. Saunders. MINOS/augmented user's guide. Technical Report SOL-80-14, Stanford University, 1987.

[72] S. G. Nash. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6:599–616, 1985.

[73] T. Steihaug. The conjugate gradient methods and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20:626–637, 1983.

[74] Thomas F. Coleman and Alex Pothen. The null space problem I: Complexity. *SIAM Journal on Algebraic and Discrete Methods*, 7:527–537, 1987.

[75] Thomas F. Coleman and Alex Pothen. The null space problem II: Algorithms. *SIAM Journal on Algebraic and Discrete Methods*, 8:544–563, 1987.

[76] J. R. Gilbert and M.T. Heath. Computing a sparse basis for the nullspace. *SIAM Journal on Algebraic and Discrete Methods*, 8:446–459, 1987.

[77] Vasek Chvátal. *Linear Programming.* W.H. Freeman and Company, 1980.

[78] Andrew R. Conn. Linear programming via a non-differentiable penalty function. *SIAM Journal on Numerical Analysis*, 13:224–241, 1988.

[79] Jorge J. Moré and Gerardo Toraldo. Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55:377–400, 1989.

[80] Ron S. Dembo and Ulrich Tulowitzki. On the minimization of quadratic functions subject to box constraints. Technical Report B 71, Yale University, 1983.

[81] Dimitri P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2):221–246, 1982.

[82] Thomas F. Coleman and Laurie Hulbert. A direct active set algorithm for large sparse quadratic programs with simple bounds. *Mathematical Programming*, 45:373–406, 1989.

[83] Chi-Geun Han, Panos M. Pardalos, and Yinyu Ye. Computational aspects of an interior point algorithm for quadratic programming problems with box constraints. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 92–112. SIAM, 1990.

[84] Thomas F. Coleman and Laurie Hulbert. A globally and superlinearly convergent algorithm for convex quadratic programs with simple bounds. Technical Report TR 90-1092, Computer Science Department, Cornell University, February, 1990 (to appear in SIAM Journal on Optimization).

[85] Yinyu Ye. On the interior algorithms for nonconvex quadratic programming. Technical report, Integrated Systems Inc., 1988.

[86] J. J. Júdice and F. M. Pires. Direct methods for convex quadratic programs subject to box constraints. Departamento de matemática, Universidade de Coimbra, 3000 Coimbra, Portugal, 1989.

[87] P. Lotstedt. Solving the minimal least squares problem subject to bounds on the variables. *BIT*, 24:206–224, 1984.

[88] E. K. Yang and J. W. Tolle. A class of methods for solving large convex quadratic programs subject to box constraints. Technical report, Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina, 1988.

[89] I. Adler, M. G. C. Resende, G. Veiga, and N. Karmarkar. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44, 1989.

[90] E.R. Barnes. A variation on Karmarkar's algorithm for solving linear programming problems. *Mathematical Programming*, 36:174–182, 1986.

[91] Thomas F. Coleman and Yuying Li. A quadratically-convergent algorithm for the linear programming problem with lower and upper bounds. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 49–57. SIAM, 1990. Proceedings of the Mathematical Sciences Institute workshop, October 1989, Cornell University.

[92] D. M. Gay. A variant of Karmarkar's linear programming algorithm for problems in standard form. *Mathematical Programming*, 37:81–90, 1987.

[93] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209, 1986.

[94] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[95] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. The primal-dual interior point method on the cray supercomputer. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 70–80. SIAM, 1990.

[96] K. A. McShane, C.L. Monma, and D.F. Shanno. An implementation of a primal-dual interior point method for linear programming. *ORSA Journal on Computing*, 1:70–83, 1989.

[97] M. J. Todd. Recent developments and new directions in linear programming. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 109–157. Kluwer Academic Publishers, 1989.

[98] M. J. Todd. Exploiting special structure in Karmarkar's linear programming algorithm. *Mathematical Programming*, 41:97–113, 1988.

[99] Robert J. Vanderbei, Marc S. Meketon, and Barry A. Freedman. A modification of Karmarkar's linear programming algorithm. *Algorithmica*, 1:395–407, 1986.

[100] Y. Ye. An $O(n^3 L)$ potential reduction algorithm for linear programming. *Mathematical Programming*, 50:239–258, 1991.

[101] Thomas F. Coleman and Yuying Li. A global and quadratic affine scaling method for (augmented) linear l_1 problems. In *Proceedings of the 1989 Dundee Conference on Numerical Analysis*, 1989.

[102] Thomas F. Coleman and Yuying Li. A global and quadratic affine scaling method for linear l_1 problems. Technical Report 89-1026, Computer Science Department, Cornell University, 1989 (to appear in Mathematical Programming.

[103] Thomas F. Coleman and Yuying Li. A global and quadratically-convergent method for linear l_∞ problems. Technical Report 90-1121, Computer Science Department, Cornell University, April, 1990 (to appear in SIAM Journal on Optimization).

[104] Yuying Li. A globally convergent method for l_p problems. Technical Report 91-1212, Computer Science Dept., Cornell University, 1991.

[105] S. G. Kratzer. Massively parallel sparse-matrix computations. Technical report, SRC-TR-90-008, Supercomputing Research Center, 1990.

[106] U. Öreborn. *A direct method for sparse nonnegative least squares problems*. PhD thesis, Department of Mathematics, Linköping University, Linköping, Sweden, 1986.

[107] Paul E. Plassmann. *The parallel solution of nonlinear least-squares problems*. Center for Applied Mathematics Ph.D thesis, Cornell University, 1990.

[108] Soren S. Nielsen and Stavros A. Zenios. A massively parallel algorithm for nonlinear stochastic network problems. Technical Report 90-09-08, Dept. of Wharton School, University of Pennsylvania,

1990.

[109] I.S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. The factorization of sparse symmetric indefinite matrices. Technical Report RAL-90-066, Science and Engineering Research Council, Rutherford Appleton Laboratory, 1990.

[110] P.E. Gill, W. Murray, M.A. Saunders, and M.H Wright. A schur-complement method for sparse quadratic programming. In M.G. Cox and S.J. Hammarling, editors, *Reliable Numerical Computing*. Oxford University Press, to appear.

[111] N.I.M. Gould. An algorithm for large scale quadratic programming. Technical Report 89-036, Computer Science and Systems Division, Harwell Laboratory, 1989.

[112] Thomas F. Coleman. On characterizations of superlinear convergence for constrained optimization. In Eugene L. Allgower and Kurt George, editors, *Computational Solution of Nonlinear Systems of Equations, Lectures in Applied Mathematics, Volume 26*, pages 113–134. American Mathematical Society, 1990.

[113] J. Stoer and R. A. Tapia. On the characterization of Q-superlinear convergence of quasi-Newton methods for constrained optimization. Technical Report 84-2, Dept. of Mathematical Sciences, Rice University, July, 1984 (Revised October 1986).

[114] Rodrigo Fontecilla, Trond Steihaug, and Richard A. Tapia. A convergence theory for a class of quasi-Newton methods for constrained optimization. *SIAM Journal on Numerical Analysis*, 24:1133–1151, 1987.

[115] J. Goodman. Newton's method for constrained optimization. *Mathematical Programming*, 33:162–171, 1985.

[116] J. Nocedal and M. Overton. Projected Hessian updating algorithms for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 22:821–850, 1985.

[117] R. A. Tapia. A stable approach to Newton's method for optimization problems with equality constraints. *Journal of Optimization Theory and Applications*, 14:453–476, 1974.

[118] P. E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. A Schur complement method for sparse quadratic programming. In *Reliable Numerical Computation*, pages 113–138. Clarendon Press, Oxford UK, 1990.

[119] Thomas F. Coleman and Christian Hempel. Computing a trust region step for a penalty function. *SIAM Journal on Scientific and Statistical Computing*, 11:180–201, 1990.

[120] N.I.M. Gould. On the accurate determination of search directions for simple differentiable penalty functions. *I.M.A. J. Numer. Anal.*, 6:357–372, 1986.

[121] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic Publishers, 1989.

[122] Andreas Griewank. Direct calculation of newton steps without accumulating Jacobians. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 115–137. SIAM, 1990.

[123] A. Griewank, D. Juedes, and J. Srinivasan. ADOL-C, A package for the automatic differentiation of of algorithms written in C/C++. Technical Report MCSA-180-1190, Argonne National Laboratory, 1990.

[124] M. Iri and K. Kubota. Methods of fast automatic differentiation and applications. Technical report, Mathematical Engineering and instrumentation Physics, University of Tokyo, 1988.

[125] M. Iri, T. Tsuchiya, and M. Hoshi. Automatic computation of partial derivatives and rounding error estimates with applications to large-scale systems of nonlinear equations. *Journal of Computational and Applied Mathematics*, 24:365–392, 1988.

[126] K. V. Kim, J. Nesterov, V.A. Skokov, and B.V. Cherkasskii. An efficient algorithm for computing derivatives and extremal problems. *MATEKON*, 21:49–67, 1985.

[127] G. L. Miller, V. Ramachandran, and E. Kaltofen. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM Journal on Computing*, 17:687–695, 1988.

[128] E. D. Chajakis and S.A. Zenios. Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization. Technical Report 89-10-07, Decision Sciences Dept., University of Pennsylvania, 1990.

[129] S. A. Zenios and Y. Censor. Massively parallel row-action algorithms for some nonlinear transportation problems. Technical Report 89-09-10, Decision Sciences Dept., University of Pennsylvania, 1989.

[130] Stavros A. Zenios, R. Qi, and E. D. Chajakis. A comparative study of parallel dual coordinate ascent implementations for nonlinear network optimization. In Thomas F. Coleman and Yuying Li, editors, *Large-Scale Numerical Optimization*, pages 238–255. SIAM, 1990.

[131] R. Byrd, R.B. Schnabel, and G. Shultz. Parallel quasi-Newton methods for unconstrained optimization. Technical report, Department of Computer Science, University of Colorado, 1988.

[132] Thomas F. Coleman and Guangye Li. Solving systems of nonlinear equations on a message-passing multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 11:1116–1135, 1990.

[133] Guangye Li and Thomas F. Coleman. A new method for solving triangular systems on a distributed memory message-passing multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 10:382–396, 1989.

[134] John R. Gilbert. Predicting structure in sparse matrix computations. Technical Report CS-86-750, Cornell University, 1986.