INFORMS Journal on Computing
Vol. 21, No. 2, Spring 2009, pp. 209–223
ISSN 1091-9856 | EISSN 1526-5528 | 09 | 2102 | 0209

DOI 10.1287/ijoc.1080.0286

Efficient Computation of Sparse Hessians Using Coloring and Automatic Differentiation

Assefaw H. Gebremedhin, Arijit Tarafdar
Department of Computer Science and Center for Computational Sciences, Old Dominion University, Norfolk, Virginia 23529 {agebreme@purdue.edu, tarafdar@cs.odu.edu}

Alex Pothen
Department of Computer Science and Computing Research Institute, Purdue University, West Lafayette, Indiana 47907, apothen@purdue.edu

Andrea Walther
Institute of Scientific Computing, Technical University of Dresden, D-01062 Dresden, Germany, andrea.walther@tu-dresden.de

The computation of a sparse Hessian matrix H using automatic differentiation (AD) can be made efficient using the following four-step procedure: (1) Determine the *sparsity structure* of H, (2) obtain a *seed* matrix S that defines a column partition of H using a specialized *coloring* on the adjacency graph of H, (3) compute the *compressed* Hessian matrix $B \equiv HS$, and (4) *recover* the numerical values of the entries of H from B.

The coloring variant used in the second step depends on whether the recovery in the fourth step is *direct* or *indirect*: a direct method uses *star coloring* and an indirect method uses *acyclic coloring*. In an earlier work, we had designed and implemented effective heuristic algorithms for these two NP-hard coloring problems. Recently, we integrated part of the developed software with the AD tool ADOL-C, which has recently acquired a sparsity detection capability. In this paper, we provide a detailed description and analysis of the recovery algorithms and experimentally demonstrate the efficacy of the coloring techniques in the overall process of computing the Hessian of a given function using ADOL-C as an example of an AD tool. We also present new analytical results on star and acyclic coloring of chordal graphs. The experimental results show that sparsity exploitation via coloring yields enormous savings in runtime and makes the computation of Hessians of very large size feasible. The results also show that evaluating a Hessian via an indirect method is often faster than a direct evaluation. This speedup is achieved without compromising numerical accuracy.

Key words: sparse Hessian computation; acyclic coloring; star coloring; automatic differentiation; combinatorial scientific computing

History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received October 2006; revised August 2007, April 2008; accepted April 2008. Published online in *Articles in Advance* September 15, 2008.

1. Introduction

1.1. Background

Solvers for nonlinear optimization problems such as IPOPT (Wächter and Biegler 2006) and LOQO (Vanderbei and Shanno 1999) require second-order derivatives of the Lagrangian function. Furthermore, exact Hessians are needed in parametric sensitivity analysis, such as in the control of dynamical systems in real time (Büskens and Maurer 2001). A Hessian matrix or Hessian-vector products can be computed accurately using *automatic differentiation* (AD), a generic name for a set of chain–rule-based techniques for evaluating derivatives of a function given as a computer program (Griewank 2000).

A Hessian matrix that arises in a large-scale application is typically *sparse*. Sparsity, along with symmetry, can be exploited to reduce the runtime and the storage required to compute the Hessian using AD (or estimate it using *finite differences*). Matrix *compression* has been found to be an effective technique in this context: given an $n \times n$ Hessian matrix H of known sparsity structure, an $n \times p$ *seed* matrix S, with p as small as possible, is determined; then the numerical values in the $n \times p$ compressed matrix $B \equiv HS$ are obtained using AD; and finally, the nonzero entries of the original Hessian H are recovered from the compressed representation B.

One way a matrix H is compressed to form a matrix B is to *partition* the columns of H into p structurally disjoint groups and let each column of B be the sum of the columns of H that belong to the same group. Mathematically, S is an $n \times p$ matrix whose (j, k) entry is one if column h_j of the matrix H belongs to group k, and zero otherwise. Because the matrix S defines a partitioning of the columns of H, each of its

entries is either zero or one, and in every row r of S there exists exactly one column c where the entry s_{rc} is equal to one.

The criteria used to find a suitable seed matrix S where the parameter p is minimized—the specific partitioning problem—depend on whether the numerical values of the entries of the Hessian H are to be recovered from the compressed matrix B *directly* or *indirectly* (via substitution). In a direct recovery, no further arithmetic is required, whereas in a recovery via substitution, a set of simple triangular systems of equations needs to be solved implicitly. The partitioning requirements for a seed matrix suitable for a substitution method are less strict than the requirements for a seed matrix appropriate for a direct method. Hence, the former usually results in a smaller p compared with the latter.

The computation of a seed matrix requires only structural information. Such matrix problems can be formulated conveniently and solved as graph problems. Coleman and Moré (1984) showed that a *star coloring* of the adjacency graph of a Hessian models the partitioning problem that occurs in the computation of the Hessian via a direct method. Coleman and Cai (1986) showed that the corresponding model in a substitution-based computation is *acyclic coloring*. These specialized graph-coloring problems are described in §2.

In a previous work, we had developed effective heuristic algorithms for the NP-hard star- and acyclic-coloring problems, and showed that the algorithms are superior to previous approaches (Gebremedhin et al. 2007). In an even earlier work (Gebremedhin et al. 2005), we had provided a comprehensive review of the role of graph coloring in the efficient computation of Jacobians and Hessians. Recently, we have developed efficient algorithms for recovering the numerical values of a Hessian from a compressed representation obtained using a star or an acyclic coloring. These algorithms take advantage of *two-colored structures* that the associated coloring algorithms maintain. Our implementations of the coloring as well as the recovery algorithms, which form part of the software package ColPack (Gebremedhin et al. 2009), have been integrated with ADOL-C, an operator-overloading based AD tool for the differentiation of functions specified in C/C++ (Griewank et al. 1996). Walther (2008) recently developed a sparsity detection technique for Hessians and added the functionality to ADOL-C.

1.2. Contributions

Our contributions in this paper are analytical as well as experimental. In terms of theory, in §3 we provide a detailed description and analysis of the Hessian recovery algorithms mentioned earlier. Our analysis includes time complexity as well as numerical stability: we show that our recovery algorithms are linear in the size of the problem and that indirect recovery using two-colored trees is nearly as stable as direct recovery. We also present, in §4, new results on star and acyclic coloring of chordal graphs, a class to which adjacency graphs of banded matrices belong.

Experimentally, we demonstrate in §5 the advantage that coloring techniques offer in sparse Hessian computation using AD. We use two test cases: a real-world power transmission problem and a synthetic unconstrained quadratic optimization problem. In both cases, we compute Hessians of various dimensions and sparsity structures, including banded and random structures, using ADOL-C. The results obtained show that sparsity exploitation via star and acyclic coloring enables one to affordably compute Hessians of dimensions that could not have been computed otherwise. For sizes where dense Hessian computation is at least possible, the saving in runtime obtained by exploiting sparsity via coloring is dramatic. Furthermore, an indirect method that uses acyclic coloring quite often is found to be faster than a direct method that relies on star coloring, considering the overall process. This speedup is achieved without compromising numerical accuracy. The experimental results also show that both the star- and the acyclic-coloring heuristic algorithm we used find *optimal* solutions for banded matrices.

1.3. Related Work

In sparse Hessian computation, an approach that can be used either orthogonal to or in combination with compression is the use of *elimination* techniques on the computational graph of the Hessian. This approach was first considered by Dixon (1991) and is the subject of current research in the AD community. Partial separability is another related approach used in sparse Hessian computation. Gay (1996) has considered such an approach and its implementation using AMPL.

2. Graph Coloring

In the current work, we have used the star- and acyclic-coloring algorithms we had developed earlier (Gebremedhin et al. 2007). After reviewing a few preliminary concepts, we briefly discuss these algorithms in this section.

2.1. Preliminaries

Two distinct vertices in a graph are *distance-k* neighbors if a shortest path connecting them consists of at most k edges. The *degree-k* of a vertex v, denoted by $d_k(v)$, is the number of distinct paths of length at most k edges that start at the vertex v; the average degree-k is denoted by $\bar{d}_k(v)$. A *distance-k coloring* of a graph $G = (V, E)$ is a mapping $\phi\colon V \to \{1, 2, \ldots, p\}$

such that whenever vertices v and w are distance-k neighbors, $\phi(v) \neq \phi(w)$. The kth power of a graph $G = (V, E)$ is the graph $G^k = (V, F)$, where $(v, w) \in F$ whenever vertices v and w are distance-k neighbors in G. A mapping ϕ is a distance-k coloring of a graph G if and only if ϕ is a distance-1 coloring of the graph G^k.

A *star coloring* of a graph is a distance-1 coloring where, in addition, every path in the graph on four vertices (P_4) is required to use at least three colors. An *acyclic coloring* of a graph is a distance-1 coloring with the further restriction that every cycle in the graph uses at least three colors. The names star and acyclic coloring are due to the structure of two-colored induced subgraphs. In a star-colored graph, a subgraph induced by any two color classes—sets of vertices having the same color—is a collection of stars. A *star* is a complete bipartite graph in which one of the vertex sets consists of a single vertex, called the *hub*. The other vertices of the star are *spokes*. An edge is a degenerate case of a star, in which one vertex is the hub and the other, the spoke, assigned arbitrarily. Similarly, in an acyclically colored graph, a subgraph induced by any two color classes is a collection of trees, and thus acyclic.

Each one of the distance-k, star-, and acyclic-coloring problems, whose objective is to appropriately color a graph using the fewest possible colors, is known to be NP-hard (Coleman and Moré 1984, Coleman and Cai 1986, Lin and Skiena 1995). The minimum number in each case is referred to as a specialized *chromatic* number of the graph. The distance-k chromatic number $\chi_k(G)$, the star chromatic number $\chi_s(G)$, and the acyclic chromatic number $\chi_a(G)$ of a general graph G are all known to be hard to approximate (Gebremedhin et al. 2007). The following observation on the relationship among these chromatic numbers follows directly from the definitions of the various colorings and the fact that a distance-1 coloring of a graph G is equivalent to a distance-2 coloring of the square graph G^2.

Observation 2.1. For every graph G, $\chi_1(G) \leq \chi_a(G) \leq \chi_s(G) \leq \chi_2(G) = \chi_1(G^2)$.

2.2. Star- and Acyclic-Coloring Algorithms

The star- and acyclic-coloring heuristic algorithms we developed in Gebremedhin et al. (2007) are founded on one common key idea: maintain and efficiently use the structure of two-colored induced subgraphs. As stated earlier, the respective subgraphs here are a collection of stars or trees. Both algorithms are *greedy* in the sense that a partial coloring is progressively extended by processing one vertex at a time. In each step, a vertex v of an input graph $G = (V, E)$ is assigned the *smallest* allowable color with respect to the current partial coloring. This is achieved by first determining the set of forbidden colors to the vertex v. Here, identifying and forbidding colors used by the vertices adjacent to the vertex v is straightforward. Maintaining the collection of two-colored structures turns out to be crucial for identifying and forbidding colors that could lead to two-colored paths on four vertices in the case of star coloring, and to two-colored cycles in the case of acyclic coloring.

For star coloring, there are exactly two cases that need to be considered to avoid a two-colored P_4 involving the vertex v.

1. If the vertex v has two adjacent vertices w and x of the *same* color, then the color of every vertex adjacent to w and the color of every vertex adjacent to x should be marked as a forbidden color to the vertex v.
2. If the vertex v has exactly one adjacent vertex w of color c, and w belongs to a star S consisting of at least two edges but is not the hub of S, then the color used by the hub of S should be marked as a forbidden color to the vertex v.

Likewise, for acyclic coloring, there is one case that needs to be addressed to avoid a two-colored cycle involving the vertex v:

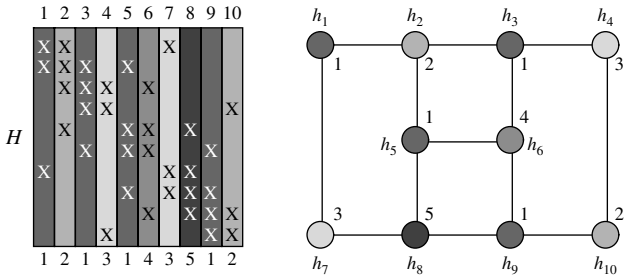
1. If the vertex v is adjacent to *two* vertices in a two-colored tree T, and these vertices have the *same* color, then the other color used in the tree T should be marked as a forbidden color to the vertex v.

Once the color for the vertex v is determined, every edge incident on v that leads to an already-colored vertex is placed in the appropriate two-colored structure. Note that because a star (respectively, an acyclic) coloring is also a distance-1 coloring, there is a unique two-colored star (respectively, two-colored tree) to which such an edge belongs. In other words, a star (respectively, an acyclic) coloring *partitions* the edge set of a graph into two-colored stars (respectively, two-colored trees). In the acyclic-coloring algorithm being described, the placement of an edge in a two-colored tree may result in the merging of two previously disconnected trees in a two-colored forest. Such a situation does not arise in the star-coloring case.

In Gebremedhin et al. (2007) it was shown that the acyclic-coloring algorithm sketched above can be implemented efficiently using the *disjoint-set* data structure to maintain the collection of two-colored trees. A simpler data structure was shown to be sufficient for a similar purpose in the star-coloring algorithm. The complexity of the star- and the acyclic-coloring algorithm was shown to be $O(|V|\bar{d}_2^2)$ and $O(|V|\bar{d}_2 \cdot \alpha)$, respectively, where α is the inverse of Ackermann's function, a function that grows very slowly with $|V|$.

3. Sparse Hessian Computation

We now relate the graph-coloring variants discussed in the previous section with partitioning strategies for Hessian compression for direct and indirect recovery.



$$B_1 = HS_1 = \begin{pmatrix} h_{11} & h_{12} & h_{17} & 0 & 0 \\ h_{21}+h_{23}+h_{25} & h_{22} & 0 & 0 & 0 \\ h_{33} & h_{32} & h_{34} & h_{36} & 0 \\ h_{43} & h_{4,10} & h_{44} & 0 & 0 \\ h_{55} & h_{52} & 0 & h_{56} & h_{58} \\ h_{63}+h_{65}+h_{69} & 0 & 0 & h_{66} & 0 \\ h_{71} & 0 & h_{77} & 0 & h_{78} \\ h_{85}+h_{89} & 0 & h_{87} & 0 & h_{88} \\ h_{99} & h_{9,10} & 0 & h_{96} & h_{98} \\ h_{10,9} & h_{10,10} & h_{10,4} & 0 & 0 \end{pmatrix}$$

Figure 1 **Top—A Symmetrically Orthogonal Partition of the Columns of a Hessian and Its Representation as a Star Coloring of the Adjacency Graph; Bottom—The Compressed Matrix Obtained by Adding Together Columns that Belong to the Same Group in the Partitioning**

Note. The illustration uses five colors—red, blue, yellow, green, and navy blue—that are also represented by the integers 1, 2, 3, 4, and 5, respectively, as shown at the bottom edge of the matrix as well as adjacent to the vertices of the graph.

3.1. Direct Method

3.1.1. Matrix Partitioning. For a direct recovery, the columns of a Hessian matrix H need to be partitioned in such a way that for every nonzero entry h_{ij}, either h_{ij} or its symmetric counterpart h_{ji} appears as a sole entry in the compressed matrix. A symmetrically orthogonal partition of H precisely captures this requirement. A partition of the columns of a Hessian H is *symmetrically orthogonal* if for every nonzero element, h_{ij}, either (1) the group containing column h_j has no other column with a nonzero in row i, or (2) the group containing column h_i has no other column with a nonzero in row j.

The left part of the upper row of Figure 1 illustrates a symmetrically orthogonal partition of a Hessian matrix H with a specific sparsity pattern. A nonzero entry of H is denoted by the symbol "X" and a zero entry is left blank. The 10 columns of H are partitioned into five groups, and columns that belong to the same group are painted with the same color. The five groups in the partition—represented by the colors red, blue, yellow, green, and navy blue—are also identified by the integers 1 through 5, respectively, as shown at the bottom edge of the matrix. (See the Online Supplement to this article, available at http://joc.pubs.informs.org/ecompanion.html, for a full-color version of this illustration, as well as the one for Figure 4.) The partition in the illustration in Figure 1 defines a 10×5 seed matrix, which we denote by S_1. For example, column 1 of S_1 has 1s in rows 1, 3, 5, and 9, corresponding to the columns of H that belong to group 1 (color red), and 0s in all other rows. The lower row of Figure 1 shows the resultant compressed matrix $B_1 = HS_1$. The reader can easily verify that every nonzero entry of the matrix H (or its symmetric counterpart) can be read off directly from some entry of the matrix B_1.

3.1.2. Coloring Model. Let H be a Hessian, each of whose diagonal elements is nonzero. The *adjacency graph* $G(H)$ of the matrix H is an undirected graph whose vertex set consists of the columns of H and whose edge set consists of pairs (h_i, h_j) whenever the matrix entry h_{ij}, $i \neq j$, is nonzero. In such a graph, entries h_{ij} and h_{ji} are represented by the single edge (h_i, h_j), and there are no explicit edges representing the diagonal entries of H. Coleman and Moré (1984) established that the problem of finding a symmetrically orthogonal partition of a Hessian having the fewest groups is equivalent to the star-coloring problem on its adjacency graph. The right part of the upper row of Figure 1 illustrates this equivalence.

3.1.3. Recovery Routines. Figure 2 outlines a simple routine, called DIRECTRECOVER1, for recovering the numerical values of the nonzero entries of a Hessian H from its compressed representation B obtained via a star coloring of $G(H)$. The routine achieves this by considering the structure of H one row at a time. The nonzero entries in a specific row are in turn considered one element at a time. For each nonzero entry h_{ij} in row i, the if-test in the inner for-loop checks whether there exists a column index j' where entries h_{ij} and $h_{ij'}$ "collide"; i.e., columns h_j and $h_{j'}$ belong to the same group. Depending on the outcome of the test, either the entry h_{ij} or the entry h_{ji}

Input: The adjacency graph $G(H)$ of a Hessian H of order n; a vertex-indexed integer array color specifying a star coloring of $G(H)$; a compressed matrix B representing H.
Output: Numerical values in H.

```
for i ← 1, 2, ..., n
  for each j where h_ij ≠ 0
    if ∃j' ≠ j where h_ij' ≠ 0 and color[h_j] = color[h_j']
      H[j, i] ← B[j, color[h_i]];
    else
      H[i, j] ← B[i, color[h_j]];
    end-if
  end-for
end-for
```

Figure 2 **DIRECTRECOVER1—A Routine for Recovering a Hessian from a Star-Coloring-Based Compressed Representation**

Input: The adjacency graph $G(H)$ of a Hessian H of order n; a vertex-indexed integer array color specifying a star coloring of $G(H)$; a set $\mathcal{S}$ of two-colored stars; a compressed matrix B representing H.
Output: Numerical values in H.
for $i \leftarrow 1, 2, \ldots, n$
  $H[i, i] \leftarrow B[i, \text{color}[h_i]]$;
end-for
for each two-colored star $S \in \mathcal{S}$
  Let h_j be the *hub* vertex in S;
  for each *spoke* vertex $h_i \in S$
    $H[i, j] \leftarrow B[i, \text{color}[h_j]]$;
  end-for
end-for

Figure 3 **DIRECTRECOVER2—A Routine Using Two-Colored Stars for Recovering a Hessian from a Star-Coloring-Based Compressed Representation**

is read from an appropriate location in the matrix B. Clearly, the if-test can be performed within $O(d_1(h_i))$ time, where h_i is the vertex being considered in the current iteration of the outer for-loop, and $d_1(h_i)$ is its degree-1 in the adjacency graph $G(H)$. Hence, the complexity of DIRECTRECOVER1 is $O(|E| \cdot \bar{d}_1)$, where $\bar{d}_1$ is the average vertex degree in $G(H)$.

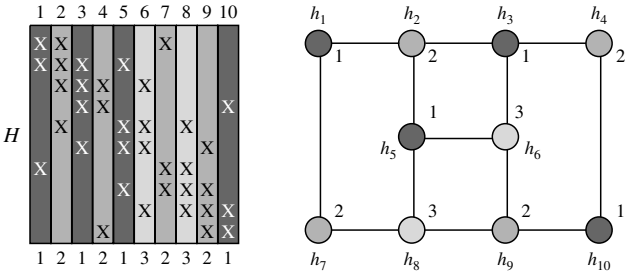
The recovery of Hessian entries in a direct method theoretically could be done more efficiently by using the set of two-colored stars defined by a star coloring of the adjacency graph. Specifically, DIRECTRECOVER2, the routine specified in Figure 3, shows that the recovery of the entries of the matrix H from the matrix B can be done in $O(|E|)$-time when the two-colored structures are readily available. As can be seen in the first for-loop, because adjacent vertices in a star coloring receive different colors, each diagonal element h_{ii} can be retrieved simply from $B[i, \text{color}(h_i)]$. The second (outer) for-loop shows that each off-diagonal element h_{ij} can be obtained by consulting the unique two-colored star to which the edge (h_i, h_j) belongs. The reader is encouraged to see the routine in Figure 3 in conjunction with the illustration in Figure 1. For example, one can see that all of the edges (off-diagonal nonzeros) that belong to the red-blue (color 1-color 2) star induced by the vertices $\{h_1, h_2, h_3, h_5\}$ can be obtained from the group that corresponds to the color of the hub vertex h_2 (i.e., column 2 of B_1). Note also that an edge such as (h_1, h_7) that belongs to a single-edge star can be obtained from either one of its endpoints.

3.2. Substitution Method

3.2.1. Matrix Partitioning. If the requirement that the entries of a Hessian be recovered directly from a compressed representation is relaxed, then the compression can be done much more compactly. One possibility here is to use what is called a substitutable partition. A partition of the columns of a symmetric matrix H is *substitutable* if there exists an ordering on the elements of H such that for every nonzero element h_{ij}, either (1) column h_j is in a group where all the nonzeros in row i, from other columns in the same group, are ordered before h_{ij}; or (2) column h_i is in a group where all the nonzeros in row j, from other columns in the same group, are ordered before h_{ij}. In the definition just stated, a nonzero entry h_{ij} of a symmetric matrix H is identified with the entry h_{ji}. A nonzero entry h_{ij} is said to be ordered before a nonzero entry $h_{i'j'}$ if h_{ij} is evaluated before $h_{i'j'}$.

3.2.2. Coloring Model. Fortunately, the rather clumsy notion of substitutable partition has a simple graph-coloring formulation: Coleman and Cai (1986) proved that an acyclic coloring of the adjacency graph of a Hessian induces a substitutable partition of its columns. Thus, the problem of finding a substitutable partition with the fewest groups reduces to the problem of finding an acyclic coloring with the fewest colors.

We use the illustration in Figure 4 to show how an acyclic coloring can be used to obtain a substitutable partition. The matrix H shown in Figure 4 has the same sparsity structure as the matrix in Figure 1. The



$$B_2 = HS_2 = \begin{pmatrix} h_{11} & h_{12}+h_{17} & 0 \\ h_{21}+h_{23}+h_{25} & h_{22} & 0 \\ h_{33} & h_{32}+h_{34} & h_{36} \\ h_{43}+h_{4,10} & h_{44} & 0 \\ h_{55} & h_{52} & h_{56}+h_{58} \\ h_{63}+h_{65} & h_{69} & h_{66} \\ h_{71} & h_{77} & h_{78} \\ h_{85} & h_{87}+h_{89} & h_{88} \\ h_{9,10} & h_{99} & h_{96}+h_{98} \\ h_{10,10} & h_{10,4}+h_{10,9} & 0 \end{pmatrix}$$

Figure 4 **Top—A Substitutable Partition of the Columns of a Hessian and Its Representation as an Acyclic Coloring of the Adjacency Graph; Bottom—The Compressed Matrix Obtained by Adding Together Columns that Belong to the Same Group in the Partitioning**

Note. The illustration uses three colors—red, blue, and yellow—that are also represented by the integers 1, 2, and 3, respectively, as shown at the bottom edge of the matrix as well as adjacent to the vertices of the graph.

coloring of the adjacency graph $G(H)$ depicted in Figure 4 is clearly acyclic. In the illustration, the columns of the matrix H have been painted in accordance with the shown acyclic coloring of $G(H)$. The lower row shows the compressed matrix obtained using the seed matrix S_2 defined by the depicted acyclic coloring.

3.2.3. Recovery Routine. We proceed to show how the acyclic coloring in Figure 4 will be used in recovering the entries of the matrix H from the compressed matrix $B_2 = HS_2$.

First, observe that every diagonal element h_{ii} is directly recoverable from the compressed matrix B_2, because h_{ii} appears "alone" in row i and column k of the matrix B_2, where k corresponds to the group to which column h_i belongs. This is a direct consequence of the fact that adjacent vertices in an acyclically colored graph, as in a star-colored graph, receive different colors.

Next, consider the task of determining the off-diagonal nonzero entries h_{ij}, $i \neq j$. Recall that each such matrix entry and its symmetric counterpart correspond to an edge in the adjacency graph, and an acyclic coloring partitions the edges of the graph into two-colored trees. We use each of these two-colored trees, *separately*, to define an *order* in which edges can be solved for.

In a two-colored tree, every edge incident on a leaf vertex can be determined directly because it is the only nonzero in a row of the group of columns to which its parent vertex belongs. As an example, consider the red-blue (color 1-color 2) tree in Figure 4 induced by the vertices $\{h_1, h_2, h_3, h_4, h_5, h_7, h_9, h_{10}\}$. In this tree, the vertex h_7 is a leaf, and the edge (h_7, h_1) can be immediately read from row 7 of the first column of B_2, the group to which column h_1 belongs. Similarly, the vertex h_9 is a leaf, and the edge (h_9, h_{10}) can be read from row 9 of the first column of B_2, again the group to which column h_{10} belongs. Likewise, edge (h_5, h_2) can be directly obtained from $B_2[5, 2]$.

Once the edges incident on leaf vertices have been determined, they can be deleted from the tree to create new leaves. The process can then be repeated to solve for edges incident on the new leaf vertices by using values computed for the leaf edges from earlier steps. The process terminates when the tree becomes empty, i.e., when all of the edges have been evaluated. In general, there are alternative ways in which an edge can be solved for, so the evaluation process is not unique.

Returning to our illustration, in the red-blue tree, once the edges (h_7, h_1), (h_9, h_{10}), and (h_5, h_2) have been evaluated and deleted, the path h_1–h_2–h_3–h_4–h_{10} remains. In this path, the edges (h_1, h_2) and (h_{10}, h_4) are incident on leaf vertices; the edge (h_1, h_2) can be evaluated using $B_2[1, 2]$ and the previously computed value for the edge (h_7, h_1), and the edge (h_{10}, h_4) can be evaluated using $B_2[10, 2]$ and the previously computed value for the edge (h_9, h_{10}). After this, the edges (h_1, h_2) and (h_4, h_{10}) can be deleted, leaving the path h_2–h_3–h_4 from which the edges (h_2, h_3) and (h_3, h_4) can be evaluated.

The red-blue tree in Figure 4 enabled the determination of 7 of the 13 distinct off-diagonal nonzero entries. The remaining six nonzeros are determined using the other two trees, the red-yellow (color 1-color 3) tree and the blue-yellow (color 2-color 3) tree.

Input: The adjacency graph $G(H)$ of a Hessian H of order n; a vertex-indexed integer array color specifying an acyclic coloring of $G(H)$; a set $\mathcal{T}$ of two-colored trees; a compressed matrix B representing H.
Output: Numerical values in H.

```
for i ← 1, 2, ..., n
  H[i, i] ← B[i, color[h_i]];
end-for
for each two-colored tree T ∈ 𝒯
  for each vertex h_j ∈ T
    storedValues[h_j] ← 0;
  end-for
  while the tree T is not empty
    Pick a leaf vertex h_i ∈ T;
    Let h_j be the parent of h_i in T;
    H[i, j] ← B[i, color[h_j]] − storedValues[h_i];
    storedValues[h_j] ← storedValues[h_j] + H[i, j];
    Delete vertex h_i (along with edge (h_i, h_j)) from T;
  end-while
end-for
```

Figure 5 **INDIRECTRECOVER—A Routine Using Two-Colored Trees for Recovering a Hessian from an Acyclic-Coloring-Based Compressed Representation**

We summarize the process we have been describing thus far in Figure 5, where we outline the routine INDIRECTRECOVER for evaluating the nonzeros of a Hessian from a compressed representation induced by an acyclic coloring. Note the resemblance between the routines DIRECTRECOVER2 and INDIRECTRECOVER. The first for-loops in each case correspond to the determination of diagonal nonzeros, and the second for-loops to the recovery of off-diagonal nonzeros from two-colored stars (respectively, trees). In INDIRECTRECOVER, the variable storedValues is used to store edge values that will be "substituted" in the determination of edges in later steps. Specifically, let (h_i, h_j) be a pair of child and parent vertices, respectively, in a two-colored tree T, and let $T(h_i)$ denote the subtree of T rooted at the vertex h_i that would remain if the edge (h_i, h_j) were to be removed from T. Then it is easy to see that the quantity stored in the variable storedValues at the index h_i is

$$\text{storedValues}[h_i] = \sum_{(h_r, h_s) \in E_{T(h_i)}} H[r, s], \qquad (1)$$

where $E_{T(h_i)}$ denotes the set of edges in the tree $T(h_i)$, and the entry $H[i, j]$ of the Hessian is computed using

the equation

$$H[i, j] = B[i, \mathsf{color}[h_j]] - \mathsf{storedValues}[h_i]. \quad (2)$$

Using appropriate data structures, the computational work associated with each two-colored tree in INDIRECTRECOVER can be performed in time proportional to the number of edges in the tree. Thus, the overall complexity of INDIRECTRECOVER on an adjacency graph $G(H) = (V, E)$ is $O(|E|)$.

3.3. Numerical Stability

How does the routine INDIRECTRECOVER compare with the routines DIRECTRECOVER1 or DIRECTRECOVER2 in terms of numerical stability? The answer to this question turns out to be quite positive: INDIRECTRECOVER for practical purposes is nearly as stable as DIRECTRECOVER. Our analytical justification for this claim is a natural extension of the works of Powell and Toint (1979) and Coleman and Cai (1986), who analyzed the error bounds associated with (specialized) substitution methods in the context of Hessian estimation using finite differences. In our context, because the compressed Hessian B is computed *analytically* using automatic differentiation (and thus *exactly* within machine precision), the associated numerical stability analysis is fundamentally different. Issues such as truncation error and choice of step length do not arise in our context.

The only arithmetic operations involved in INDIRECTRECOVER are subtraction and addition; the absence of division is highly favorable for numerical stability. Furthermore, the determination of nonzeros (edges) in one two-colored tree is entirely independent of the determination of edges in another two-colored tree, and therefore there is less opportunity for error magnification. As we shall show shortly, the error accumulation within a two-colored tree is in turn very limited.

Let (h_i, h_j) be an edge in the input graph $G(H) = (V, E)$ to INDIRECTRECOVER, and let $T = (V_T, E_T)$ be the two-colored tree to which the edge (h_i, h_j) belongs. Furthermore, as done earlier, let $T(h_i) = (V_{T(h_i)}, E_{T(h_i)})$ denote the subtree of T rooted at the vertex h_i that is obtained by removing the edge (h_i, h_j) from T. Let n, n_T, and $n_{T(h_i)}$ denote the number of vertices in $G(H)$, T, and $T(h_i)$, respectively. Our goal is to prove a bound on the accuracy of the Hessian entry $H[i, j]$ computed using Equation (2). First, following Powell and Toint (1979), we define the error matrix $\mathscr{E}$, the pointwise difference between the computed Hessian H and the analytic Hessian $\nabla^2 f$, as

$$\mathscr{E}[i, j] = H[i, j] - (\nabla^2 f(x))_{ij} \quad (3)$$

for all pairs i and j such that (h_i, h_j) is an edge in $G(H)$. In Theorem 3.1 we will show that $|\mathscr{E}[i, j]|$, the magnitude of the error associated with the computation of $H[i, j]$ by INDIRECTRECOVER, is bounded by the product of $n_{T(h_i)}$, the number of vertices in the *subtree* $T(h_i)$ of T, and a constant independent of T.

THEOREM 3.1. *The numerical value computed by* INDIRECTRECOVER *for each edge* (h_i, h_j) *in the input graph* $G(H)$ *is such that* $|\mathscr{E}[i, j]| \le n_{T(h_i)} \cdot \eta$, *where* η *is a positive constant*.

PROOF. Because the compressed Hessian B is computed in floating point arithmetic, it inevitably contains rounding errors. Let

$$\hat{B}[i, \mathsf{color}[h_j]] = B[i, \mathsf{color}[h_j]] + \varepsilon[i, \mathsf{color}[h_j]] \quad (4)$$

denote the computed matrix taking such errors into account. Thus, the actual value $H[i, j]$ evaluated using INDIRECTRECOVER is

$$H[i, j] = \hat{B}[i, \mathsf{color}[h_j]] - \mathsf{storedValues}[h_i]. \quad (5)$$

Analogous to the error matrix $\mathscr{E}$ associated with H, let the error matrix δ be the pointwise difference between the computed values contained in $\hat{B}$ and the corresponding values in the analytic Hessian $\nabla^2 f$; i.e., let

$$\begin{aligned}&\delta[i, \mathsf{color}[h_j]] \\ &:= \hat{B}[i, \mathsf{color}[h_j]] - \sum_{(h_r, h_s) \in E_{T(h_i)}} (\nabla^2 f(x))_{rs} - (\nabla^2 f(x))_{ij}. \quad (6)\end{aligned}$$

Using Equations (1) and (5), Equation (6) can be written as

$$\begin{aligned}\delta[i, \mathsf{color}[h_j]] &= H[i, j] - (\nabla^2 f(x))_{ij} \\ &\quad + \sum_{(h_r, h_s) \in E_{T(h_i)}} (H[r, s] - (\nabla^2 f(x))_{rs}) \\ &= \mathscr{E}[i, j] + \sum_{(h_r, h_s) \in E_{T(h_i)}} \mathscr{E}[r, s].\end{aligned}$$

It then follows that

$$\mathscr{E}[i, j] = \delta[i, \mathsf{color}[h_j]] - \sum_{(h_r, h_s) \in E_{T(h_i)}} \mathscr{E}[r, s].$$

Applying the same decomposition to each $\mathscr{E}[r, s]$ for $(h_r, h_s) \in E_{T(h_i)}$, one obtains

$$\mathscr{E}[i, j] = \delta[i, \mathsf{color}[h_j]] - \sum_{(h_r, h_s) \in E_{T(h_i)}} \delta[r, \mathsf{color}[h_s]].$$

Taking the absolute values of scalar quantities and noting that the tree $T(h_i)$ has $n_{T(h_i)} - 1$ edges,

$$\begin{aligned}|\mathscr{E}[i, j]| &\le |\delta[i, \mathsf{color}[h_j]]| \\ &\quad + \sum_{(h_r, h_s) \in E_{T(h_i)}} |\delta[r, \mathsf{color}[h_s]]| \quad (7) \\ &\le |\delta[i, \mathsf{color}[h_j]]| + (n_{T(h_i)} - 1) \\ &\quad \cdot \max_{(h_r, h_s) \in E_{T(h_i)}} |\delta[r, \mathsf{color}[h_s]]|. \quad (8)\end{aligned}$$

If we assume that the second derivatives of f are bounded, then there exists a positive constant M such that the maximum of the two terms in Equation (8), and indeed of similar terms in the entire graph $G(H)$, can be bounded as follows:

$$\eta := \max_{1\le r', s'\le n} \{|\delta[r', \mathsf{color}[h_{s'}]]|\}$$
$$\le M + \max_{1\le r', s'\le n} \{|\varepsilon[r', \mathsf{color}[h_{s'}]]|\}.$$

Thus, Equation (8) reduces to $|\mathscr{C}[i, j]| \le n_{T(h_i)} \cdot \eta$, which is what we wanted to show. □

Suppose the acyclic coloring used in the context of INDIRECTRECOVER is actually a star coloring. Then, clearly, for each edge (h_i, h_j), the two-colored tree to which the edge (h_i, h_j) belongs is a star; i.e., $T(h_i)$ is simply the vertex h_i. In such a case, in agreement with our expectations, Theorem 3.1 suggests that the error associated with the evaluation of (h_i, h_j) is bounded simply by η.

4. Coloring Chordal Graphs

The test suite in our experiments includes Hessian matrices with banded nonzero structures, whose adjacency graphs are *band graphs* (defined later in this section). Here, we present analytical results on the distance-k, star, and acyclic chromatic numbers of the larger class of *chordal* graphs, an important class of graphs with a wide range of applications (Brandstädt et al. 1999).

Let A be a symmetric matrix of order n with nonzero diagonal elements. The *lower bandwidth* of A is defined as $\beta_l(A) = \max\{|i-j|: i > j,\ a_{ij} \neq 0\}$, and the *bandwidth* of A is the quantity $\beta(A) = 2\beta_l(A) + 1$. The matrix A is *banded* if it is completely dense within the band; i.e., if for any pair of indices $1 \le i,\ j \le n$, $|i-j| \le \beta(A) \Leftrightarrow a_{ij} \neq 0$.

The bandwidth of a matrix has a twin concept in the adjacency graph.

Let $G = (V, E)$ be a graph on n vertices and let π be an ordering $v_1, v_2, \ldots, v_n$ of the vertices. The *bandwidth of the ordering* π in G is $\beta_\pi(G) = \max\{|i-j|: (v_i, v_j) \in E\}$, and the *bandwidth of the graph* G is $\beta(G) = \min\{\beta_\pi(G):\ \pi \text{ is an ordering of } V\}$. A graph G is a *band graph* if there exists an ordering $v_1, v_2, \ldots, v_n$ of its vertices such that for any pair of indices i, j drawn from the set $\{1, 2, \ldots, n\}$, $|i-j| \le \beta(G) \Leftrightarrow (v_i, v_j) \in E$; the order $v_1, v_2, \ldots, v_n$ is referred to as the *natural ordering* of the band graph G. For a general graph, finding a vertex ordering with the minimum bandwidth—computing the quantity $\beta(G)$—is an NP-complete problem (Papadimitriou 1976).

The bandwidth of a symmetric matrix A and that of the adjacency graph $G(A)$ are related in the following way: $\beta(G(A)) \le \beta_l(A) \equiv (\beta(A) - 1)/2$. In this relationship, equality holds when A is banded, in which case $G(A)$ is a band graph, and the natural ordering of the vertices of $G(A)$ corresponds to the given ordering of the columns of A. When A is not banded, there exists a permutation matrix P such that $\beta(G(A)) = \beta_l(PAP^T)$.

A graph G is *chordal* if every cycle in G of length at least four has a *chord*—an edge that connects two nonconsecutive vertices in the cycle. Clearly, a band graph is chordal. In fact, it is a highly structured, almost regular, chordal graph: the degree $d(v_i)$ of the ith vertex in the natural ordering of the vertices can be expressed as $d(v_i) = d(v_{n-i+1}) = \beta(G) + i - 1$ for $1 \le i \le \beta(G)$, and $d(v_i) = 2\beta(G)$ for $\beta(G) + 1 \le i \le n - \beta(G)$. A graph does not need to be this regular to be chordal. For example, the adjacency graph of a symmetric matrix in which rows are allowed to have variable number of nonzeros, but the nonzeros in every row are required to be consecutive, is chordal but not band.

In what follows we present results, which we believe to be new, concerning the relationships among the distance-k chromatic number $\chi_k(G)$, the star chromatic number $\chi_s(G)$, the acyclic chromatic number $\chi_a(G)$, the *clique number* $\omega(G)$—the size of the largest clique in G—and the bandwidth $\beta(G)$ of a chordal (not necessarily band) graph G. We begin with a simple observation that is true of any (not necessarily chordal) graph.

LEMMA 4.1. *For every graph* $G = (V, E)$, $\omega(G) \le \beta(G) + 1$, *and* $\omega(G^2) \le \min\{2\beta(G) + 1, |V|\}$.

PROOF. Let π be an ordering of the vertices of G such that $\beta_\pi(G) = \beta(G)$. Suppose there exists a clique Q in G of size greater than $\beta(G) + 1$. Then it means that there exists some pair of vertices v and w in the clique Q such that $|\pi(v) - \pi(w)| > \beta(G)$, a contradiction. Hence, the clique number of G cannot exceed $\beta(G) + 1$. The result for the square graph can be shown in an analogous fashion. □

LEMMA 4.2. *If a mapping* ϕ *is a distance-*1 *coloring for a chordal graph* G, *then* ϕ *is also an acyclic coloring for* G.

PROOF. Let ϕ be a distance-1 coloring for a chordal graph G. Consider any cycle C in G, and let l be its length. If $l = 3$, then ϕ clearly uses three colors. If $l \ge 4$, then C contains a chord, and therefore ϕ uses at least three colors. Hence, ϕ is an acyclic coloring for G. □

THEOREM 4.3. *For every chordal graph* G, $\chi_a(G) = \chi_1(G) = \omega(G) \le \beta(G) + 1$. *In the last relationship, equality holds when* G *is a band graph.*

PROOF. The equality $\chi_a(G) = \chi_1(G)$ follows from Lemma 4.2. Because a chordal graph is *perfect*, $\chi_1(G) = \omega(G)$ by the *perfect graph theorem* (Lovász 1972). The last inequality was proven in Lemma 4.1 for any

(including chordal) graph. When G is a band graph, clearly, $\omega(G) = \beta(G) + 1$. □

There exist chordal graphs where the last inequality in Theorem 4.3 is strict. For example, a star graph G on n vertices has $\omega(G) = 2$, but $\beta(G) = \lfloor n/2 \rfloor$.

THEOREM 4.4. *For every chordal graph* $G = (V, E)$,

$$\chi_s(G) \leq \chi_2(G) = \omega(G^2) \leq \min\{2\beta(G) + 1, |V|\}.$$

In both the first and the third relationship, equality holds when G is a band graph.

PROOF. The first inequality follows from Observation 2.1, and the third from Lemma 4.1. The square graph of a chordal graph is chordal and hence perfect. Therefore, $\chi_2(G) = \chi_1(G^2) = \omega(G^2)$. Turning to the special case of band graphs, Coleman and Moré (1984) have shown that for a band graph G with at least $3\beta(G) + 1$ vertices, $\chi_s(G) = \chi_2(G)$. For a band graph G, $\omega(G^2) = \min\{2\beta(G) + 1, |V|\}$. □

There exist chordal graphs where the first inequality in Theorem 4.4 is strict. An example, once again, is a star graph G on n vertices, which has $\chi_s(G) = 2$ but $\chi_2(G) = n$.

If symmetry were to be ignored, a *structurally orthogonal* partition of the columns of a Hessian—a partition in which no two columns in a group have nonzeros at a common row index—could be used to compress a Hessian in a direct method. As McCormick (1983) first showed, a structurally orthogonal partition of a Hessian can be modeled by a distance-2 coloring of its adjacency graph. In light of these facts, the result $\chi_s(G) = \chi_2(G) = 2\beta(G) + 1$ for a band graph G given in Theorem 4.4 is negative: it shows that exploiting symmetry in a direct computation of a banded Hessian matrix (star coloring) does not lead to fewer colors in comparison with a direct computation that ignores symmetry (distance-2 coloring). The result $\chi_1(G) = \chi_a(G) = \beta(G) + 1$ in Theorem 4.3, on the contrary, shows that symmetry exploitation in a banded matrix is worthwhile in a substitution method.

We conclude this section with some remarks on the performance of *greedy* coloring algorithms on chordal graphs. Recall that a greedy coloring algorithm processes vertices in some *order*, each time assigning a vertex the *smallest* allowable color subject to the conditions of the specific coloring problem. There exist several "degree"-based ordering techniques—including largest-degree-first, smallest-degree-last and incidence-degree—that have proven to be quite effective (but still suboptimal) for distance-k coloring of general graphs (Gebremedhin et al. 2005).

For chordal graphs, better ordering techniques exist. Given a graph $G = (V, E)$, an ordering $v_1, v_2, \ldots, v_n$ of the vertices in V is a *perfect elimination ordering* (*peo*) of G if for all $i \in \{1, 2, \ldots, n\}$, the vertex v_i is such that its neighbors in the subgraph induced by the set $\{v_i, \ldots, v_n\}$ form a clique. It is well known that a graph G is chordal if and only if it has a *peo*. It is also known that a greedy distance-1 coloring algorithm that uses the reverse of a *peo* of G gives an optimal solution, i.e., computes a coloring with $\chi_1(G)$ colors (Brandstädt et al. 1999). For the special case of a band graph, the natural ordering of the vertices, as well as its reverse, is a *peo*. Thus, a greedy distance-1 coloring algorithm that uses the natural ordering of the vertices would give an optimal solution. However, as Lemma 4.5 and its corollary will imply, an optimal coloring for a band graph can be obtained without actually executing the greedy algorithm.

LEMMA 4.5. *Let* $G = (V, E)$ *be any graph and let* $v_1, v_2, \ldots, v_n$ *be an ordering in which the bandwidth of G is attained. Then the mappings* $\phi_1(v_i) = i \bmod (\beta(G) + 1)$ *and* $\phi_2(v_i) = i \bmod (2\beta(G) + 1)$ *define a distance-*1 *coloring and a distance-*2 *coloring of G, respectively. If G is a band graph, then both of these colorings are optimal.*

COROLLARY 4.6. *For every graph* $G = (V, E)$, $\chi_1(G) \leq \beta(G) + 1$, *and* $\chi_2(G) \leq \min\{2\beta(G) + 1, |V|\}$. *In both relationships, equality holds when G is a band graph.*

The optimality of ϕ_1 and ϕ_2 in Lemma 4.5 in the case of band graphs, and the implied equalities in Corollary 4.6, follow from Theorems 4.3 and 4.4.

5. Numerical Experiments

In this section, we present experimental results concerning the following four steps involved in the efficient computation of the Hessian of a given function f.

Step S0: Detect the sparsity pattern of the Hessian

Step S1: Obtain a seed matrix S using an appropriate graph coloring

Step S2: Compute the Hessian-seed matrix product $B \equiv HS$

Step S3: Recover the nonzero entries of H from B

We use two different optimization problems as test cases: an electric power flow problem, representing a real-world application; and an unconstrained quadratic optimization problem, a synthetic case chosen for a detailed performance analysis. The underlying test function f in both test cases is specified in the programming language C. The coloring and recovery codes (steps S1 and S3) are written in C++ as part of the software package ColPack (Gebremedhin et al. 2009) and are incorporated into ADOL-C. For step S3, in the direct case, the routine DIRECTRECOVER1 is used. For the step S2, the *second-order adjoint mode* in the latest version of ADOL-C, which is significantly faster than previous versions, is used. When needed, the sparsity structure of the Hessian (step S0)

Table 1 Absolute Runtimes in Seconds for the Evaluation of the Function f and the Steps S0, S1, S2, and S3 for Test Hessians in the Optimal Power Flow Problem

			Direct				Indirect				
n	Eval(f)	S0	S1	S2	S3	Tot	S1	S2	S3	Tot	Dense
72	0.000008	0.001	0.0006	0.007	0.00003	0.008	0.0004	0.005	0.0004	0.005	0.08
3,760	0.000587	0.247	0.0252	0.805	0.00248	0.833	0.0464	0.646	0.0617	0.755	201.32
4,472	0.000717	0.372	0.0329	0.899	0.00307	0.935	0.0601	0.561	0.0791	0.700	257.99
9,932	0.001933	6.212	0.0929	2.839	0.00719	2.939	0.2220	1.159	0.3049	1.686	1315.43
22,540	0.002118	22.662	0.2170	20.162	0.01741	20.396	1.0402	12.531	1.2981	14.869	***

Notes. The last column shows runtime for the computation of a Hessian without exploiting sparsity. The asterisks indicate that space could not be allocated for the full Hessian.

is determined using the recently added functionality in ADOL-C (Walther 2008). The experiments on the power flow problem are performed on a Linux system with an Intel Xenon 1.5 GHz processor and 1 GB RAM, and those on the synthetic problem are performed on a Fedora Linux system with an AMD Athlon XP 1.666 GHz processor and 512 MB RAM. In both cases, the gcc 4.1.1 compiler with -02 optimization is used.

5.1. Optimal Power Flow Problem

5.1.1. Description. This problem is concerned with the management of power transmission over a network that has observable parts, where measured data are available, and unobservable parts (Dancre et al. 2002). For an unobservable part, one usually has estimations of the data, for example, from the past. For a proper management of the network, however, a complete and actual database for the entire network is needed. Therefore, one relies on *computed* data in all parts of the network. In the observable areas, the computed data should be as close to the measured data as possible. In addition, one needs to minimize the *weighted least-squares* distance between the computed data and the estimated data in unobservable parts and boundary areas. This gives a nonlinear optimization problem of the form

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad g(x)=0, \quad l \le h(x) \le u, \qquad (9)$$

with the objective function $f\colon \mathbb{R}^n \to \mathbb{R}$, the equality constraints $g\colon \mathbb{R}^n \to \mathbb{R}^m$, and the inequality constraints $h\colon \mathbb{R}^n \to \mathbb{R}^p$ being twice continuously differentiable. The optimization problem (9) can be solved using an interior-point based tool such as LOQO or IPOPT (Vanderbei and Shanno 1999, Wächter and Biegler 2006). These solvers require the provision of the Jacobian matrices $\nabla g(x)$ and $\nabla h(x)$ as well as the Hessian of the Lagrange function $L(x,\lambda,\mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$ with respect to x in sparse format. We report on runtimes for this Hessian computation using actual problem instances.

5.1.2. Results and Discussion. Table 1 lists the absolute runtimes in seconds spent in the various steps for the five Hessians considered in our experiments. The first two columns of Table 2 show the number of rows and the average number of nonzeros per row in the Hessians used in the experiments; the next two columns show the number of colors used in the direct and indirect cases; and the last four columns show timing results for various steps, each normalized relative to the time needed to evaluate the underlying function f.

The results in Tables 1 and 2 clearly show that employing coloring in Hessian computation enables one to solve large-size problems that could not otherwise have been solved. For problem sizes where dense computation is possible, the results show that sparsity exploitation via coloring yields huge savings in runtime. Furthermore, it can be seen that indirect computation using acyclic coloring is faster than direct computation using star coloring, considering overall runtime. Comparing the steps S1, S2, and S3 against each other, as can be seen from Figure 6, the coloring (S1) and recovery (S3) steps are almost negligible compared with the step in which the Hessian-seed matrix product is computed (S2), both in the direct and indirect methods.

Numerically, we observed that indirect recovery gave Hessian entries of the same accuracy as direct recovery. This experimental observation agrees well with the analysis in §3.3.

Table 2 Matrix Structural Data, Number of Colors, and Normalized Runtime Relative to Function Evaluation for Test Hessians in the Optimal Power Flow Problem

		Colors			Total (S1–S3)		
n	$\bar{\rho}$	Direct	Indirect	S0	Direct	Indirect	Dense
72	3.72	9	6	125	954	725	10,000
3,760	4.11	15	12	421	1,419	1,285	342,964
4,472	3.99	16	10	519	1,304	976	359,819
9,932	3.91	22	9	3,213	1,520	872	680,512
22,540	3.94	16	10	10,700	9,630	7,020	***

Note. The asteriks indicate that space could not be allocated.

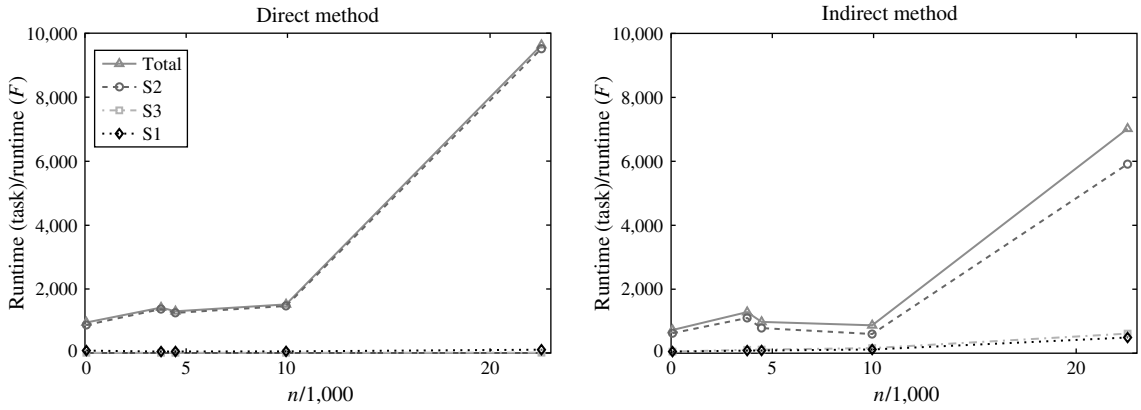


Figure 6 **Runtimes of the Various Steps Normalized by the Runtime of Function Evaluation for the Power Flow Problem**

A final point to be noted from Table 2 is that the runtime of the sparsity detection routine is relatively large in comparison with the routines in the other steps. In future work, we plan to explore ways in which this can be reduced.

5.2. Unconstrained Quadratic Optimization Problem

5.2.1. Description. The sizes and structures of the Hessians from the optimal power flow problem that we could include in our experiments were quite limited. To be able to study the performance of the various steps in a systematic fashion, we considered a synthetic problem in which we have a direct control over the size and structure of the Hessians. In particular, we used the unconstrained quadratic optimization problem $\min_{x\in\mathbb{R}^n} f(x)$ with $f(x) = x^T Cx + a^T x$, $C \in \mathbb{R}^{n\times n}$ and $a^T = (10, \ldots, 10) \in \mathbb{R}^n$, where the Hessian is simply the matrix C. We considered two kinds of sparsity structures for the matrix C: *banded* (bd) and *random* (rd). Furthermore, the test matrices were designed in such a way that:

(i) the number of nonzeros per row in a banded matrix (denoted by ρ) is nearly the same as the number of nonzeros per row in a random matrix (denoted by $\bar{\rho}$), and

(ii) the value for ρ, or $\bar{\rho}$, remains constant as the problem dimension n is varied.

In our experiments, we used the values $\rho \in \{10, 20\}$, $\bar{\rho} \in \{10.98, 20.99\}$, and $n/1{,}000 \in \mathcal{I} \equiv \{5, 10, 20, 40, 60, 80, 100\}$.

5.2.2. Results and Discussion.

Number of Colors. Table 3 provides a summary of the numbers of colors used by the star-coloring algorithm (direct method) and the acyclic-coloring algorithm (indirect method) for all the sparsity structures and input sizes considered in the experiments. Two observations can be made from this table.

First, for the banded structure, the acyclic- and star-coloring algorithms invariably used $\lfloor \rho/2 \rfloor + 1$ and $2\lfloor \rho/2 \rfloor + 1$ colors, respectively, regardless of the value of n. In view of Theorems 4.3 and 4.4, and noting that $\lfloor \rho/2 \rfloor$ is the bandwidth of the corresponding graphs, we see that both algorithms find *optimal* solutions for band graphs. Both algorithms are greedy, and vertices were colored in the natural ordering of the graphs. Hence, the observed phenomenon agrees with the theory of distance-1 coloring discussed in the last paragraph of §4.

Second, in both the star- and the acyclic-coloring cases, the numbers of colors required by the random structures were observed to be nearly twice the corresponding numbers in the banded structures. Moreover, the numbers of colors varied only slightly as the problem dimension n was varied.

Runtime. Table 4 lists the absolute runtime in seconds spent in the various steps while using a direct and an indirect method. The information in Table 4 is analogous to that presented in Table 1 for the optimal power flow problem. The general conclusion to be drawn from Table 4 in terms of the enabling power of the coloring techniques in the overall computation is similar to that drawn from the optimal power flow problem. Our objective here is to show how the execution time for each step grows as a function of the input size.

Figure 7 shows a collection of *normalized* runtime versus problem dimension (n) plots. In particular, the

Table 3 **Number of Colors Used by the Star- and Acyclic-Coloring Algorithms for *All* Problem Dimensions n in the Set $n/1{,}000 \in \mathcal{I} \equiv \{5, 10, 20, 40, 60, 80, 100\}$**

	$\rho, \bar{\rho}$			
	10, 10.98		20, 20.99	
	Star	Acyclic	Star	Acyclic
bd	11	6	21	11
rd	21–24	9–11	50–56	18–19

Table 4 **Absolute Runtimes in Seconds for the Evaluation of the Function f and the Steps S1, S2, and S3 in the Quadratic Optimization Problem**

		Direct				Indirect				
n/1,000	Eval(f)	S1	S2	S3	Tot	S1	S2	S3	Tot	Dense
bd:										
5	0.0006	0.05	0.20	0.006	0.25	0.08	0.11	0.05	0.24	86.1
10	0.0010	0.11	0.40	0.012	0.52	0.25	0.22	0.10	0.57	342.0
20	0.0019	0.23	0.79	0.023	1.05	0.81	0.43	0.21	1.45	***
40	0.0035	0.55	1.61	0.046	2.21	2.86	0.86	0.46	4.18	***
60	0.0049	0.81	2.31	0.066	3.19	6.12	1.25	0.73	8.10	***
80	0.0062	1.03	3.06	0.088	4.18	11.03	1.65	0.99	13.67	***
100	0.0077	1.31	3.83	0.110	5.25	16.33	2.08	1.23	19.64	***
rd:										
5	0.0005	0.07	0.41	0.007	0.49	0.10	0.17	0.11	0.38	147.9
10	0.0009	0.19	0.90	0.014	1.10	0.30	0.38	0.30	0.98	589.4
20	0.0020	0.46	2.01	0.029	2.50	0.95	0.83	0.87	2.65	***
40	0.0043	1.08	4.51	0.064	5.66	3.21	1.84	2.67	7.73	***
60	0.0082	1.77	7.54	0.103	9.41	6.75	3.13	5.25	15.13	***
80	0.0125	2.50	10.78	0.140	13.41	11.52	4.54	8.66	24.71	***
100	0.0170	3.26	13.16	0.174	16.60	17.52	5.55	13.01	36.07	***
bd:										
5	0.0008	0.16	0.63	0.015	0.81	0.14	0.33	0.12	0.59	84.0
10	0.0015	0.36	1.27	0.030	1.66	0.37	0.67	0.23	1.28	362.8
20	0.0029	0.84	2.57	0.061	3.47	1.07	1.34	0.51	2.91	***
40	0.0056	1.68	5.14	0.118	6.93	3.40	2.61	1.10	7.12	***
60	0.0084	2.56	7.46	0.176	10.19	6.93	3.91	1.66	12.51	***
80	0.0109	3.54	10.47	0.234	14.24	11.85	5.47	2.31	19.63	***
100	0.0135	4.44	12.95	0.300	17.70	17.74	6.81	3.07	27.61	***
rd:										
5	0.0008	0.29	1.66	0.017	1.97	0.20	0.55	0.24	0.99	151.0
10	0.0015	0.70	3.49	0.035	4.23	0.52	1.15	0.65	2.31	594.9
20	0.0033	1.65	8.14	0.071	9.86	1.44	2.67	1.87	5.98	***
40	0.0087	3.91	19.99	0.156	24.06	4.35	6.43	5.59	16.36	***
60	0.0150	6.04	29.91	0.234	36.19	8.56	9.89	10.81	29.26	***
80	0.0246	8.91	45.12	0.338	54.36	14.37	15.00	18.03	47.40	***
100	0.0317	11.93	54.71	0.413	67.07	21.34	19.00	26.91	67.25	***

Notes. The upper half of the table shows results for $\rho = 10$ (banded) and $\bar{\rho} = 10.98$ (random) and the lower half for $\rho = 20$, and $\bar{\rho} = 20.99$. All runtimes are averages of five runs. For the random structures, the runtimes are in addition averaged over five randomly generated matrices.

vertical axis in each subfigure shows the execution time of a specific step divided by the time needed to evaluate the function f being differentiated; note that the *scales* on the axes differ from subfigure to subfigure. Below, we discuss the runtime behavior of the various steps turn by turn; but first, we look at how the normalizing quantity, the time needed for evaluating f, itself grows as a function of n.

Time for Evaluating f. Because the number of nonzeros per row (column) in the structures we considered is constant, the time needed to evaluate the function f theoretically is expected to be linear in the number of rows (columns) n. Figure 8 shows that the practically observed execution times are roughly linear in n across the structures we considered. For the banded structures, the growth is actually slightly sublinear. The growth is somewhat superlinear for the random structures, especially for the cases where $\bar{\rho} = 20.99$. This is due mainly to the irregular memory accesses involved and the associated nonuniform costs in hierarchical memory.

Step S1: Coloring and Generation of Seed Matrix. Recall from §2 that the complexity of the star-coloring algorithm for a graph on n vertices is $O(n\bar{d}_2^2)$ and that of the acyclic-coloring algorithm is $O(n\bar{d}_2 \cdot \alpha)$, where α is the inverse of Ackermann's function. For the banded sparsity structures, the quantity $\bar{d}_2$ in the associated adjacency graphs is nearly ρ^2, independent of the parameter n. In light of these facts, the trends observed in the various cases in Figure 7 are in agreement with theoretical analyses. For the banded structures (the top two rows), it can be seen that the runtime of the star-coloring algorithm grows linearly with n (left column), whereas the runtime of the acyclic-coloring algorithm is slightly superlinear (right column). The general trend in the random structures is very similar, but slightly more erratic, again due to irregular memory accesses.

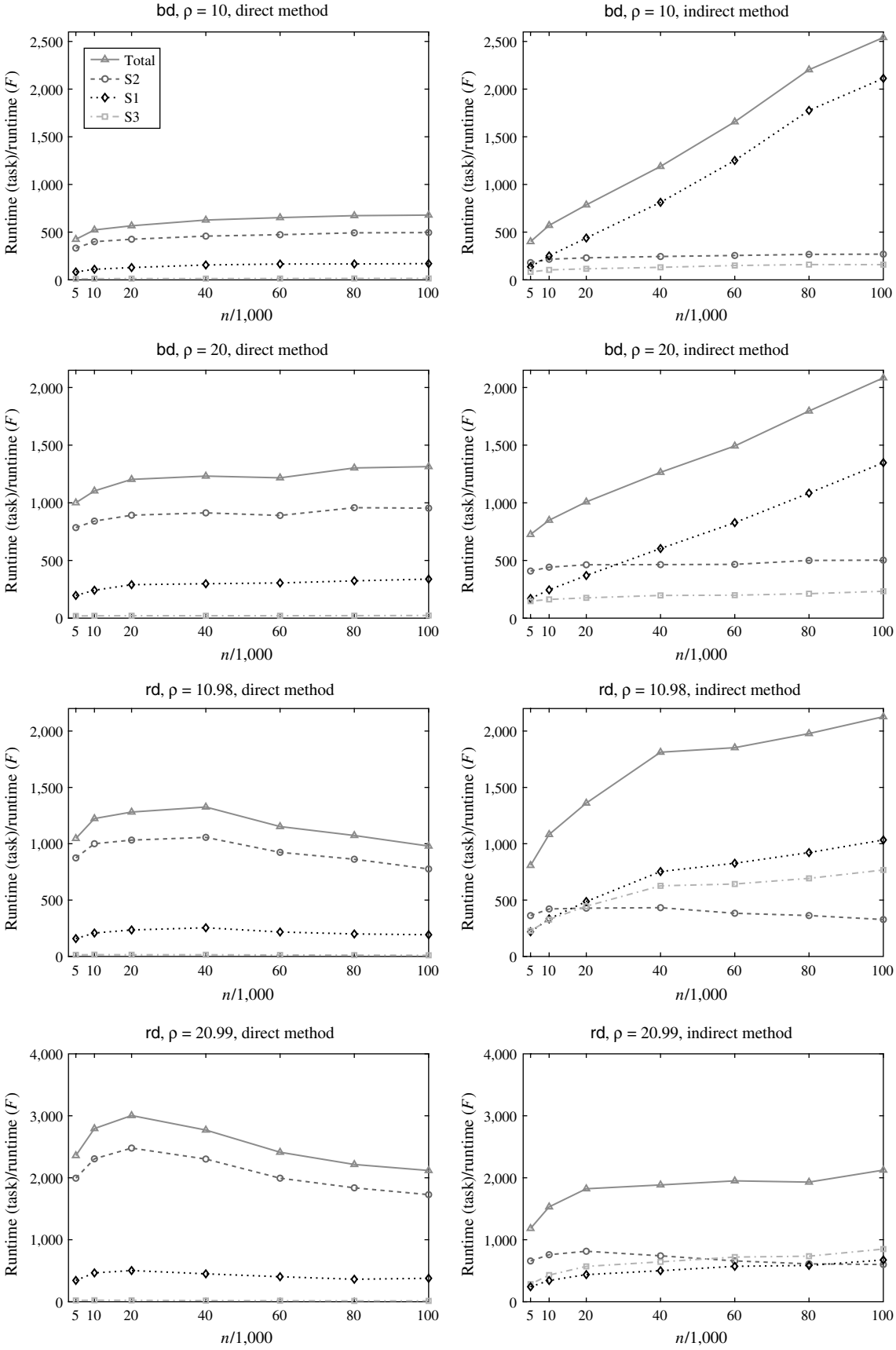


Figure 7 Execution Time of the Various Steps Normalized by the Time Needed for Function Evaluation vs. Problem Size

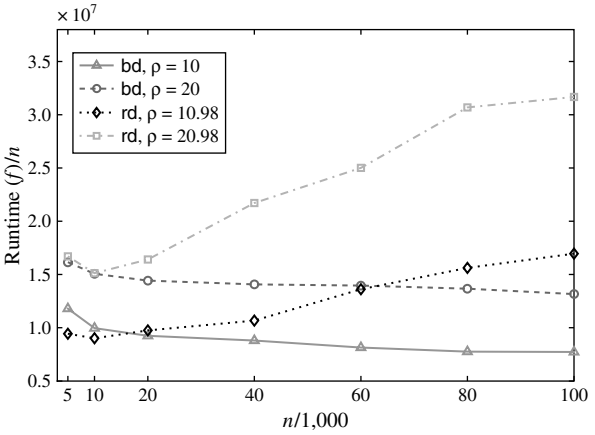


Figure 8 **Runtime of Function Evaluation in Seconds Normalized by Input Size n vs. n**

Step S2: Computation of the Compressed Hessian. Figure 7 shows that the time for the step in which the compressed Hessian HS is computed is linear in the problem dimension n. The analytical justification for this behavior stems from two sources. First, as mentioned earlier, the time needed for evaluating the function f is linear in n. Second, the number of columns p in a seed matrix (the number of colors used) remained constant or nearly constant as the problem dimension n in our experiments was varied, both in a direct and an indirect method. Theoretically, the complexity of computing the Hessian-seed vector product using AD is known to be a small constant (in the order of 10) times the time need to evaluate the function being differentiated (Griewank 2000). Hence, the fact that the observed runtime grew linearly with n for both structures is consistent with theoretical analyses.

Step S3: Recovery of the Original Hessian Entries. As discussed in §3, the complexity of DIRECTRECOVER1 is $O(m\bar{\rho})$, where m is the number of nonzeros in the Hessian. The constant hidden in this expression is rather small, because the computation involved is fairly easy. Similarly, the complexity of INDIRECTRECOVER, which relies on the use of two-colored trees, was shown to be $O(m)$. Due to the overhead associated with the management of nontrivial data structures, the hidden constant here is expected to be considerably larger, to the extent that the execution time of the routine in practice becomes more than the corresponding time for DIRECTRECOVER1. The observed runtimes in Figure 7 clearly reflect these facts. For a similar reason, even though DIRECTRECOVER2 theoretically is faster than DIRECTRECOVER1, we used the latter in our experiments because it is likely to be faster in practice.

Overall Runtime. Considering all the steps together, is a direct method faster or slower than an indirect method? The results in Figure 7 show that the answer depends on the size and structure of the Hessian being computed. For the random structure with nearly 20 nonzeros per row, an indirect method is consistently observed to be faster than a direct method. A similar statement can be made for the banded structures of relatively small size (n up to 20,000). For larger-size banded problems and for many of the random matrices with nearly 10 nonzeros per row, a direct method was observed to be faster. These observations are in contrast to those in the optimal power flow problem where an indirect method was always found to be faster. Comparing the relative contribution of the various steps with the total runtime, we observe that the Hessian-seed matrix product step (S2) is by a large margin the most expensive in a direct method, whereas the coloring step (S1) is slightly the dominant step in an indirect method.

Numerical Accuracy. As in the optimal power flow problem, here again, the numerical values of the Hessian entries obtained using INDIRECTRECOVER were observed to be of the same accuracy as the values obtained using DIRECTRECOVER1—a typical pair of values obtained using the two methods matched in all of the computed digits in double precision.

6. Conclusion

We studied compression-based calculation of sparse Hessians using automatic differentiation. We considered the case where a matrix is compressed such that the recovery is direct (star coloring) and the case where the recovery requires additional arithmetic work (acyclic coloring). Our experimental results showed that sparsity exploitation via star and acyclic coloring enables one affordably to compute Hessians of dimensions that could not have been computed otherwise. For sizes where a computation that does not exploit sparsity is at least possible, the results showed that the techniques render dramatic savings in runtime. We believe savings of similar magnitude would be attained should an AD tool other than ADOL-C be used, because the execution time for the Hessian-seed matrix product is likely to dominate the overall runtime for any reasonable function. The experimental results also showed that, for real-world optimization problems, an acyclic-coloring-based method is faster than a star-coloring-based method, considering the overall process. Furthermore, we showed, both analytically and experimentally, that indirect recovery using two-colored trees is numerically as stable as direct recovery.

Acknowledgments

The authors thank Dr. Fabrice Zaoui of EDF R&D MOSSE, Clamart, France for helping with the experiments on the power flow problem. They also thank the anonymous

referees for their valuable comments, which helped them improve the quality of the paper. This work was supported by the Office of Science of the U.S. Department of Energy under the Scientific Discovery Through Advanced Computing (SciDAC) program through Grant DE-FC-0206-ER-25774 awarded to the CSCAPES Institute, by the U.S. National Science Foundation Grant ACI 0203722, and by the German Research Foundation Grant Wa 1607/2-1. A. H. Gebremedhin's current affiliation is Department of Computer Science and Computing Research Institute, Purdue University, West Lafayette, IN 47907.

References

Brandstädt, A., V. B. Le, J. P. Spinrad. 1999. *Graph Classes: A Survey: Monographs on Discrete Mathematics and Applications*. SIAM, Philadelphia.

Büskens, C., H. Maurer. 2001. Sensitivity analysis and real-time optimization of parametric nonlinear programming problems. M. Gröschel, S. Krumke, J. Rambau, eds. *Online Optimization of Large Scale Systems*. Springer, Berlin, 3–16.

Coleman, T. F., J. Cai. 1986. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM J. Algebraic Discrete Methods* **7**(2) 221–235.

Coleman, T. F., J. J. Moré. 1984. Estimation of sparse Hessian matrices and Graph coloring problems. *Math. Programming* **28** 243–270.

Dancre, M., P. Tournebise, P. Panciatici, F. Zaoui. 2002. Optimal power flow applied to state estimation enhancement. *Proc. 14th Power Systems Comput. Conf.*, Sevilla, Spain, Paper 3, Session 37.

Dixon, L. 1991. Use of automatic differentiation for calculating Hessians and Newton steps. A. Griewank, G. Corliss, eds. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM, Philadelphia, 114–125.

Gay, D. 1996. More AD of nonlinear AMPL models: Computing Hessian information and exploiting partial separability. M. Berz, C. Bischof, G. Corliss, A. Griewank, eds. *Computational Differentiation: Techniques, Applications, and Tools*. SIAM, Philadelphia, 173–184.

Gebremedhin, A. H., F. Manne, A. Pothen. 2005. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Rev.* **47**(4) 629–705.

Gebremedhin, A. H., A. Tarafdar, D. Nguyen, A. Pothen. 2009. ColPack: A graph coloring package for supporting sparse derivative matrix computation. *ACM Trans. Math. Software.* In preparation.

Gebremedhin, A. H., A. Tarafdar, F. Manne, A. Pothen. 2007. New acyclic and star coloring algorithms with application to computing Hessians. *SIAM J. Sci. Comput.* **29** 1042–1072.

Griewank, A. 2000. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Frontiers in Applied Mathematics, Number 19. SIAM, Philadelphia.

Griewank, A., D. Juedes, J. Utke. 1996. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software* **22** 131–167.

Lin, Y., S. S. Skiena. 1995. Algorithms for square roots of graphs. *SIAM J. Discrete Math.* **8** 99–118.

Lovász, L. 1972. A characterization of perfect graphs. *J. Combin. Theory* **13** 95–98.

McCormick, S. T. 1983. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Math. Programming* **26** 153–171.

Papadimitriou, C. 1976. The NP-completeness of the bandwidth minimization prolem. *Computing* **16** 263–270.

Powell, M. J. D., P. L. Toint. 1979. On the estimation of sparse Hessian matrices. *SIAM J. Numer. Anal.* **16**(6) 1060–1074.

Vanderbei, R., D. Shanno. 1999. An interior-point algorithm for nonconvex nonlinear programming. *Comput. Optim. Appl.* **13** 231–252.

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

Walther, A. 2008. Computing sparse Hessians with automatic differentiation. *ACM Trans. Math. Software* **34**(1) Paper 3.