Parallel Distance-*k* Coloring Algorithms for Numerical Optimization

Assefaw Hadish Gebremedhin[1], Fredrik Manne[1], and Alex Pothen[**2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{assefaw,fredrikm}@ii.uib.no

[2] Computer Science Department, Old Dominion University, Norfolk, VA 23529 USA
CSRI, Sandia National Labs, Albuquerque NM 87185 USA
ICASE, NASA Langley Research Center, Hampton, VA 23681-2199 USA
pothen@cs.odu.edu

Abstract. Matrix partitioning problems that arise in the efficient estimation of sparse Jacobians and Hessians can be modeled using variants of graph coloring problems. In a previous work [6], we argue that *distance-2* and *distance-*$\frac{3}{2}$ graph coloring are robust and flexible formulations of the respective matrix estimation problems. The problem size in large-scale optimization contexts makes the matrix estimation phase an expensive part of the entire computation both in terms of execution time and memory space. Hence, there is a need for both shared- and distributed-memory parallel algorithms for the stated graph coloring problems. In the current work, we present the first practical shared address space parallel algorithms for these problems. The main idea in our algorithms is to randomly *partition* the vertex set equally among the available processors, let each processor *speculatively* color its vertices using information about already colored vertices, detect eventual conflicts in parallel, and finally re-color conflicting vertices sequentially. *Randomization* is also used in the coloring phases to further reduce conflicts. Our PRAM-analysis shows that the algorithms should give almost linear speedup for sparse graphs that are large relative to the number of processors. Experimental results from our OpenMP implementations on a Cray Origin2000 using various large graphs show that the algorithms indeed yield reasonable speedup for modest numbers of processors.

1 Introduction

Numerical optimization algorithms that rely on derivative information often need to compute the Jacobian or Hessian matrix. Since this is an expensive part of the computation, efficient methods for estimating these matrices via finite differences (FD) or automatic differentiation (AD) are needed. It is known that the problem of minimizing the number of function evaluations (or AD passes)

** This author's research was supported by NSF grant DMS-9807172, DOE ASCI level-2 subcontract B347882 from Lawrence Livermore National Lab; and by DOE SCI-DAC grant DE-FC02-01ER25476.

required in the computation of these matrices can be formulated as variants of graph coloring problems [1–3, 8, 11]. The particular coloring problem differs with the optimization context: whether the Jacobian or the Hessian matrix is to be computed; whether a direct or a substitution method is employed; and whether only columns, or only rows, or both columns and rows are to be used to evaluate the matrix elements. In addition, the type of coloring problem depends on the kind of graph used to represent the underlying matrix. In [6], we provide an integrated review of previous works in this area and identify the *distance-2* (D2) graph coloring problem as a unifying, generic, and robust formulation. The D2-coloring problem has also noteworthy applications in other fields such as channel assignment [10] and facility location problems [12].

Large-scale PDE-constrained optimization problems can be solved only with the memory and time resources available on parallel computers. In these problems, the variables defined on a computational mesh are already distributed on the processors, and hence parallel coloring algorithms are needed for computing, for instance, the Jacobian. It turns out that the problems of efficiently computing the Jacobian and Hessian can be formulated as the D2- and D$\frac{3}{2}$-coloring problems, respectively. The latter coloring problem is a relaxed variant of the former, and will be described in Section 2.2. In these formulations, the bipartite graph associated with the rows and columns of the matrix is used for the Jacobian; in the case of the Hessian matrix, the adjacency graph corresponding to the symmetric matrix is used.

In this paper, we present several new deterministic as well as probabilistic parallel algorithms for the D2- and D$\frac{3}{2}$-coloring problems. Our algorithms are practical and effective, well suited for shared address space programming, and have been implemented in C using OpenMP primitives. We report results from experiments conducted on a Cray Origin2000 using large graphs that arise in finite element methods and in eigenvalue computations.

In the sequel, we introduce the graph problems in Section 2, present the algorithms in Section 3, discuss our experimental results in Section 4, and conclude the paper in Section 5.

2 Background

2.1 Matrix Partition Problems

An essential component of the efficient estimation of a sparse Jacobian or Hessian using FD or AD is the problem of finding a suitable *partition* of the columns and/or the rows of the matrix. The particular partition chosen defines a system of equations from which the matrix entries are determined. A method that utilizes a diagonal system is called a *direct* method, and one that uses a triangular system is called a *substitution* method. A direct method is more restrictive but the computation of matrix entries is straightforward and numerically stable. A substitution method, on the other hand, is less restrictive but it may be subject to approximation difficulties and numerical instability. Moreover, direct methods offer more parallelism than substitution methods. In this paper, we focus on direct methods that use column partitioning.

A partition of the columns of an unsymmetric matrix A is said to be *consistent* with the direct determination of A if whenever a_{ij} is a non-zero element of A then the group containing column j has no other column with a non-zero in row i [1]. Similarly, a partition of the columns of a symmetric matrix A is called *symmetrically consistent* with the direct determination of A if whenever a_{ij} is a non-zero element of A then *either* (i) the group containing column j has no other column with a non-zero in row i, or (ii) the group containing column i has no other column with a non-zero in row j [2]. From a given (symmetrically) consistent partition $\{C_1, C_2, \ldots, C_\rho\}$ of the columns of A, the nonzero entries can be determined with ρ function evaluations (matrix-vector products).

Thus we have the following two problems of interest. Given the sparsity structure of an unsymmetric $m \times n$ matrix A, find a consistent partition of the columns of A with the fewest number of groups. We refer to this problem as UNSYMCOLPART. Similarly, the problem that asks for a symmetrically consistent partition of a symmetric A is referred to as SYMCOLPART.

2.2 Graph Problems

In a graph, two distinct vertices are said to be *distance-k neighbors* if the shortest path connecting them consists of at most k edges. A *distance-k ρ-coloring* (or (k,ρ)-coloring for short) of a graph $G = (V,E)$ is a mapping $\phi : V \rightarrow \{1, 2, \ldots, \rho\}$ such that $\phi(u) \neq \phi(v)$ whenever u and v are distance-k neighbors. We call a mapping $\phi : V \rightarrow \{1, 2, \ldots, \rho\}$ a $(\frac{3}{2}, \rho)$-coloring of a graph $G = (V,E)$ if ϕ is a $(1,\rho)$-coloring of G and every path containing three edges uses at least three colors. Notice that a $(\frac{3}{2}, \rho)$-coloring is a restricted $(1,\rho)$-coloring, and a relaxed $(2,\rho)$-coloring, and hence the name. For instance, consider a path u, v, w, x in a graph. The assignment $2, 1, 2, 3$ to the respective vertices is a valid D1- and D$\frac{3}{2}$-coloring, but not a valid D2-coloring. The *distance-k graph coloring problem* asks for a (k,ρ)-coloring of a graph with the least possible value of ρ.

Let A be an $m \times n$ rectangular matrix with rows $r_1, r_2, \ldots, r_m$ and columns $a_1, a_2, \ldots, a_n$. We define the *bipartite graph* of A as $G_b(A) = (V_1, V_2, E)$ where $V_1 = \{r_1, r_2, \ldots, r_m\}$, $V_2 = \{a_1, a_2, \ldots, a_n\}$, and $(r_i, a_j) \in E$ whenever $a_{ij} \neq 0$, for $1 \leq i \leq m$, $1 \leq j \leq n$. When A is an $n \times n$ symmetric matrix with non-zero diagonal elements, the *adjacency graph* of A is defined to be $G_a(A) = (V,E)$ where $V = \{a_1, a_2, \ldots, a_n\}$ and $(a_i, a_j) \in E$ whenever a_{ij}, $i \neq j$, is a non-zero element of A. Note that the non-zero diagonal elements of A are not explicitly represented by edges in $G_a(A)$. In our work, we rely on the bipartite and adjacency graph representations. However, in the literature, an unsymmetric matrix A is often represented by its *column intersection graph* $G_c(A)$. In this representation, the columns of A constitute the vertex set, and an edge (a_i, a_j) exists whenever the columns a_i and a_j have non-zero entries at the same row position (i.e., a_i and a_j are not structurally orthogonal). As argued in [6], the bipartite graph representation is more flexible and robust than the 'compressed' column intersection graph.

Coleman and Moré [1] showed that problem UNSYMCOLPART is equivalent to the D1-coloring problem when the matrix is represented by its column intersection graph. We have shown [6] that the same problem is equivalent to a partial D2-coloring when a bipartite graph is used and discussed the relative merits of the two approaches. The word 'partial' reflects the fact that only the vertices corresponding to the columns need to be colored.

McCormick [11] showed that the approximation of a Hessian using a direct method is equivalent to a D2-coloring on the adjacency graph of the matrix. One drawback of McCormick's formulation is that it does not exploit symmetry. Later Coleman and Moré [2] addressed this issue and showed that the resulting problem (SYMCOLPART) is equivalent to the D$\frac{3}{2}$-coloring problem.

3 Parallel Coloring Algorithms

The distance-k coloring problem is NP-hard for any fixed integer $k \geq 1$ [11]. A proof-sketch showing that D$\frac{3}{2}$-coloring is NP-hard is given in [2]. Furthermore, Lexicographically First $\Delta + 1$ Coloring (LFC), the polynomial variant of D1-coloring in which the vertices are given in a predetermined order and the question at each step is to assign the vertex the smallest color not used by any of its neighbors, is P-complete [7]. The practical implication of this is that designing efficient fine-grained parallel algorithm for LFC is hard.

In practice, greedy sequential D1-coloring heuristics are found to be quite effective [1]. In a recent work [5], we have shown effective methods of parallelizing such greedy algorithms in a coarse-grained setting. Here, we extend this work to develop parallel algorithms for the D2- and D$\frac{3}{2}$-coloring problems.

Jones and Plassmann [9] describe a parallel distributed memory D1-coloring algorithm that uses randomization to assign priorities to the vertices, and then colors the vertices in the order determined by the priorities. There is no speculative coloring in their algorithm. It was reported that the algorithm slows down as the number of processors is increased. Finocchi et al.[4] suggest a parallel D1-coloring algorithm organized in several rounds; in each round, currently uncolored vertices are assigned a tentative pseudo-color without consulting their neighbors mapped to other processors; in a conflict resolution step, a maximal independent set of vertices in each color class is assigned these colors as final; the remainder of the vertices are uncolored, and the algorithm moves into the next round. However, they do not give any implementation and we believe that this algorithm incurs too many rounds, each with its synchronization and communication steps, for it to be practical on large graphs.

Our algorithm (in its generic form) may be viewed as a compromise between these algorithms, where we permit speculative coloring, but limit the number of synchronization steps to two in the whole algorithm. However, our current algorithms rely on the shared address space programming model; we will adapt our algorithms to distributed memory programming models in future work. We are unaware of any previous work on parallel algorithms for D2- and D$\frac{3}{2}$-coloring problems.

3.1 Generic Greedy Parallel Coloring Algorithm (GGPCA)

The steps of our generic parallel coloring algorithm can be summarized as shown below; refer to [5] for a detailed discussion of the D1-coloring case. Let $G = (V, E)$ be the input graph and p be the number of processors.

Phase 0: *Partition*
  Randomly partition V into p equal blocks $V_1 \dots V_p$. Processor P_i is responsible for coloring the vertices in block V_i.
Phase 1: *Pseudo-color*
  for $i = 1$ **to** p **do in parallel**
    for each $u \in V_i$ **do**
      assign the smallest available color to u, paying attention to already colored vertices (both local and non-local).
Phase 2: *Detect conflicts*
  for $i = 1$ **to** p **do in parallel**
    for each $u \in V_i$ **do**
      check whether the color of u is valid. If the colors of u and v are the same for some $(u, v) \in E$, then
        $L_i = L_i \cup min\{u, v\}$
Phase 3: *Resolve conflicts*
  Color the vertices in the conflict list $L = \cup L_i$ sequentially.

In Phase 0, the vertices are randomly partitioned into p equal blocks each of which is assigned to some processor. In Phase 1, when two adjacent vertices are on different processors, the two processors could color both simultaneously, possibly assign them the same value, and cause a conflict. The purpose of Phase 2 is to detect and store any such conflict vertices which are subsequently re-colored sequentially in Phase 3.

3.2 Simple Distance-2 (SD2) Coloring Algorithm

The meaning of 'available' color in GGPCA depends on the required coloring. In the case of a D2-coloring, a vertex is assigned the smallest color not used by any of its D2-neighbors. Let Δ denote the maximum degree in the graph. It can easily be verified that, in a D2-coloring, a vertex can always be assigned one of the colors from the set $\{1, 2, \dots, \Delta^2 + 1\}$. Moreover, since the D1-neighbors of a vertex are D2-neighbors with each other, the D2-chromatic number (the least number of colors required in a D2-coloring) is at least $\Delta + 1$. Thus, the greedy approach is an $O(\Delta)$-approximation algorithm. We refer to the variant of GGPCA that applies to D2-coloring as Algorithm SD2.

Note that the sequential time complexity of greedy D2-coloring is $O(\Delta^2|V|)$. The following results show that the number of conflicts discovered in Phase 2 of Algorithm SD2 is often small for sparse graphs, making the algorithm scalable when the number of processors $p = O(\sqrt{\frac{|V|^2}{\Delta|E|}})$. The proofs, which are essentially similar to those of the D1-coloring case given in [5], have been omitted for space considerations. Let $\overline{\delta} = 2|E|/|V|$ denote the average vertex degree in G.

Lemma 1 *The expected number of conflicts created at the end of Phase 1 of Algorithm SD2 is at most* $\approx \frac{\Delta\overline{\delta}(p-1)}{2}$.

Theorem 2 *On a CREW PRAM, Algorithm SD2 distance-2 colors the input graph consistently in expected time* $O(\Delta^2(\frac{|V|}{p}+\Delta\overline{\delta}p))$ *using at most* Δ^2+1 *colors.*

Corollary 3 *When* $p = O(\sqrt{\frac{|V|^2}{\Delta|E|}})$, *the expected runtime of Algorithm SD2 is* $O(\frac{\Delta^2|V|}{p})$.

The number of conflicts predicted by Lemma 1 is an over estimate. The analysis assumes that whenever two D2-adjacent vertices are colored simultaneously, they are assigned the same color, thereby resulting in a conflict. However, a more involved probabilistic analysis that takes the distribution of colors used into account may provide a tighter bound. Besides, the actual number of conflicts in an implementation could be significantly reduced by choosing a random color from the allowable set, instead of the smallest one as given in Phase 1 of GGPCA.

3.3 Improved Distance-2 (ID2) Coloring Algorithm

The number of colors used in Algorithm SD2 can be reduced using a 'two-round-coloring' strategy. The underlying idea in the D1-coloring case is due to Culberson and was used in the parallel algorithm of [5]. Here we extend the result to the D2-coloring case.

Lemma 4 *Let* ϕ *be a distance-2 coloring of a graph* G *using* χ *colors, and* π *a permutation of the vertices such that if* $\phi(v_{\pi(i)}) = \phi(v_{\pi(l)}) = c$, *then* $\phi(v_{\pi(j)}) = c$ *for* $i < j < l$. *Applying the greedy sequential distance-2 algorithm to* G *where the vertices have been ordered by* π *will produce a coloring* ϕ' *using* χ *or fewer colors.*

The idea is that if the greedy coloring algorithm is re-applied on a graph, with the vertices belonging to the same color class (in the original coloring) listed consecutively, then the new coloring obtained is better or at least as good as the original. One ordering (among many) that satisfies this condition, with a good potential for reducing the number of colors used, is to list the vertices consecutively for each color class in the reverse order of the introduction of the color classes. Based on Lemma 4, we modify Algorithm SD2 and introduce an additional parallel coloring phase between Phases 1 and 2. Algorithm ID2 below outlines the resulting 4-phase algorithm.

Phases 0 and 1. Same as Ph. 0 and 1 of GGPCA. Let s be the number of colors.
Phase 2. **for** $k = s$ **downto** 1 **do**
    Partition ColorClass(k) into p equal blocks $V'_1, \ldots, V'_p$
    for $i = 1$ **to** p **do in parallel**
        for each $u \in V'_i$ **do**
            assign the smallest available color to vertex u.
Phases 3 and 4. Same as Phases 2 and 3 of GGPCA, respectively.

In Phase 2, most of the vertices in a color class are at a distance greater than two edges from each other, and the exceptions arise from the conflict vertices colored incorrectly in Phase 1. Since the number of such conflict vertices from Phase 1 is low, the number of conflict vertices at the end of the re-coloring phase will be even lower. Phases 3 and 4 are included to detect and resolve any eventual conflicts not resolved in Phase 2.

3.4 Simple Distance-$\frac{3}{2}$ (SD$\frac{3}{2}$) Coloring Algorithm

Recall that a D$\frac{3}{2}$-coloring is a *relaxed* D2-coloring (see the example in Section 2.2). One way of relaxing the requirement for D2-coloring in GGPCA so as to obtain a valid D$\frac{3}{2}$-coloring is to let two vertices at a distance of *exactly* two edges from each other share a color as long as the vertex in between them is already colored with a (color of) lower value. We refer to the variant of GGPCA that employs this technique to achieve a distance-3/2 coloring as Algorithm SD$\frac{3}{2}$.

Note that both Algorithms ID2 and SD$\frac{3}{2}$ take asymptotically the same time as Algorithm SD2.

3.5 Randomization

The potential scalability of GGPCA depends on the number of conflicts discovered in Phase 2, since these are resolved sequentially in Phase 3. For dense graphs the number of conflicts could be large enough to destroy the scalability of the algorithm. To overcome this problem, we use randomization as a means for reducing the number of conflicts. In the randomized variants of Algorithms SD2, SD$\frac{3}{2}$, and ID2, a vertex is assigned the next available color with probability q, where $0 < q \leq 1$. The first attempt is made with the smallest available color, i.e., the color is chosen with probability q. An attempt is said to be successful if the vertex is assigned a color. If an attempt is not successful, then the next available color is tried with probability q, and so on, until the vertex gets a color. Algorithms SD2, SD$\frac{3}{2}$, and ID2, can be seen as the deterministic variants where $q = 1$. We refer to the randomized versions of the respective algorithms as RSD2, RSD$\frac{3}{2}$, and RID2.

Let u and v be two vertices with the same (infinite) set of allowable colors. If u and v are colored concurrently, it can be shown that the probability that u and v get the same color is $q/(2-q)$. This shows how randomization leads to a reduction in the number of conflicts; the lower the value of q, the lower chance for a conflict to arise. It should however be noted that a 'low' value of q may result in an increase in the number of colors used. Choosing the right value for q thus becomes a design issue.

4 Experimental Results and Discussion

Our test bed consists of graphs that arise from finite element methods and from eigenvalue computations [5]. Due to space limitations, we report on results obtained using two representative graphs, one from each field. Table 1 provides the test graphs' structural information. We point out that similar tendencies were observed in experiments using the other graphs from our test bed.

Table 1. Test graphs: the last 3 columns list the max., min., and average degree

Problem	$\|V\|$	$\|E\|$	Δ	δ	$\bar{\delta}$
fe144	144,649	1,074,393	26	4	14
ev02	19,845	3,353,890	749	78	338

Table 2 provides coloring and timing information of the different algorithms. Since the deterministic algorithms gave acceptable results on the relatively sparse graph fe144, the randomized variants were not run on this graph. Table 2 shows that Algorithm SD2 uses many fewer colors than the bound $\Delta^2 + 1$ and that Algorithm ID2 reduces the number of colors by up to 10% compared to SD2. The advantage of exploiting symmetry in Problem SYMCOLPART can be seen by comparing the number of colors used in SD2 and SD$\frac{3}{2}$; the latter can be as much as 37% fewer than the former. In general, the number of colors used in our deterministic algorithms increases only slightly with increasing p. For the randomized algorithms, the number of colors in some cases even decreases as p increases, but we think this is a random phenomenon.

Table 2. Results for fe144 (upper) and ev02 (lower) half. χ(alg x) and t(alg x) give the number of colors and the time in sec., respectively, used by alg x

p	χ(sd2)	χ(id2)	χ(sd$\frac{3}{2}$)	χ(rsd2)	χ(rid2)	χ(rsd$\frac{3}{2}$)	t(sd2)	t(id2)	t(sd$\frac{3}{2}$)	t(rsd2)	t(rid2)	t(rsd$\frac{3}{2}$)
1	41	37	35	-	-	-	6.8	9.6	8.1	-	-	-
12	43	38	36	-	-	-	1.1	1.6	1	-	-	-
1	4260	4016	2697	5564	4024	3142	255	456	247	257	380	257
12	5168	4455	3049	5536	4104	3081	214	112	45	31	106	29

Fig. 1 shows how the different phases of the algorithms scale as more processors are employed. For graph fe144, the time used in resolving conflicts sequentially is negligible compared to the overall time. Moreover, it can be seen that Phases 1 and 2 of Algorithms SD2 and SD$\frac{3}{2}$ scale rather well on this graph as the number of processors is increased. However, Phase 2 of Algorithm ID2 does not scale as well. This is due to the existence of many color classes with few vertices which entails extra synchronization and communication overhead in the parallel re-coloring phase.

The picture for the more dense graph ev02 is different: the time elapsed in Phase 3 of Algorithm SD2 is significant and increases as the number of processors is increased. The situation is somewhat better for SD3/2 and ID2. Note that ev02 is about 100 times denser than fe144 (where density = $\frac{|E|}{|V|^2}$), and the results in Table 2 and Fig. 1 agree well with the results in Corollary 3: when the average degree is high, we lose scalability.

Fig. 1d shows how using probabilistic algorithms solves the problem of high number of conflicts for ev02. The improvement in scalability comes at the expense

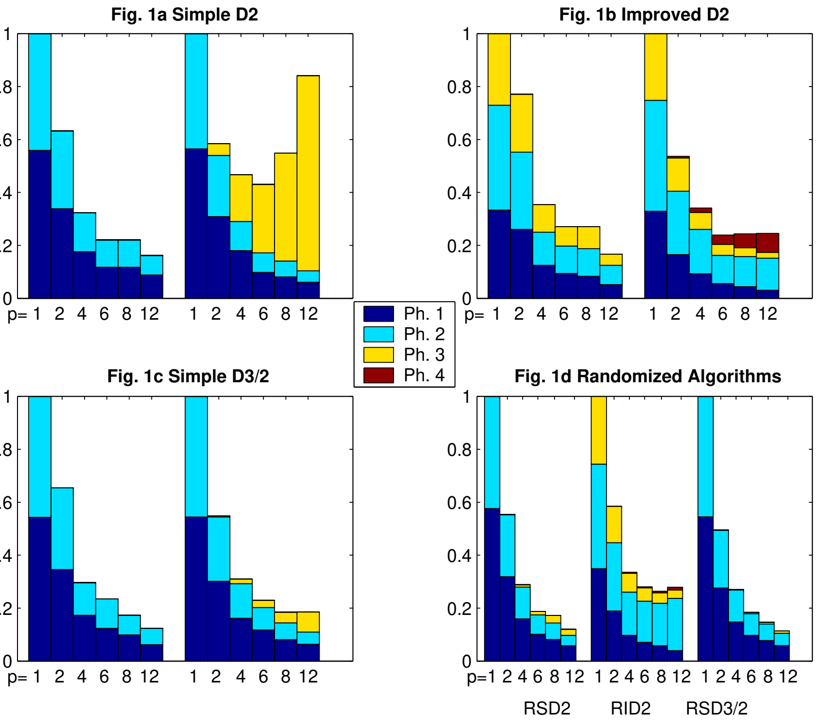

Fig. 1. a-c: Relative performance of deterministic algorithms on graphs fe144 (left-half on each sub-figure) and ev02 (right-half). Fig. 1d: Relative performance of randomized algorithms on ev02.

of increased number of colors used (see Table 2). We have experimented using different values for q and found good results when $q = 1/20$ for RSD2, and $q = 1/6$ for RID2 and RSD$\frac{3}{2}$.

It should be noted that the speedups observed in Fig. 1 are all relative to the respective parallel algorithm run with $p = 1$. A comparison against a sequential version (with no conflict detecting and resolving phase) would yield less speedup. In particular, relative to a sequential version, the ideal speedup obtained by Algorithms SD2 and ID2 is roughly $\frac{1}{2}p$ and $\frac{2}{3}p$, respectively.

Our algorithms in general did not scale well beyond around 16 processors. We believe this is due to, among other things, the relatively high cost associated with non-local physical memory accesses. It would be interesting to see how this affects the behavior of the algorithms on different parallel platforms.

5 Conclusion

We have presented several simple and effective parallel approximation algorithms as well as results from OpenMP-implementations for the D2- and D$\frac{3}{2}$-coloring problems. The number of colors produced by the algorithms in the case where $p = 1$ is generally good as it is typically off from the lower bound $\Delta+1$ of SD2 by a factor much less than the approximation ratio Δ. As more processors are employed, the algorithms provide reasonable speedup while maintaining the quality of the solution. In general, our deterministic algorithms seem to be suitable for sparse graphs and the probabilistic variants for more dense graphs. We believe the functionality provided by our algorithms is useful for many large-scale optimization codes, where parallel speedups while desirable, are not paramount, as long as running times for coloring are low relative to the other steps in the optimization computations. The three sources of parallelism in our algorithms – partitioning, speculation, and randomization – can be exploited in developing distributed parallel algorithms, but the algorithms would most likely differ significantly from the shared memory variants presented here.

References

1. T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, February 1983.
2. T. F. Coleman and J. J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.*, 28:243–270, 1984.
3. T. F. Coleman and A. Verma. The efficient computation of sparse Jacobian matrices using automatic differentiation. *SIAM J. Sci. Comput.*, 19(4):1210–1233, July 1998.
4. I. Finocchi, A. Panconesi, and R. Silvestri. Experimental analysis of simple, distributed vertex coloring algorithms. In *Proceedings of the Thirteenth ACM-SIAM Symposium on Discrete Algorithms (SODA 02)*, San Francisco, CA, 2002.
5. A. H. Gebremedhin and F. Manne. Scalable parallel graph coloring algorithms. *Concurrency: Pract. Exper.*, 12:1131–1146, 2000.
6. A. H. Gebremedhin, F. Manne, and A. Pothen. Graph coloring in optimization revisited. Technical Report 226, University of Bergen, Dept. of Informatics, Norway, January 2002. Available at: http://www.ii.uib.no/publikasjoner/texrap/.
7. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, 1995.
8. A.K.M. S. Hossain and T. Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optimization Methods and Software*, 10:33–48, 1998.
9. M. T. Jones and P. E. Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14(3):654–669, May 1993.
10. S. O. Krumke, M. V. Marathe, and S. S. Ravi. Approximation algorithms for channel assignment in radio networks. In *DIAL M for Mobility*, Dallas, Texas, 1998.
11. S. T. McCormick. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Math. Program.*, 26:153–171, 1983.
12. V. V. Vazirani. *Approximation Algorithms*. Springer, 2001. Chapter 5.