Efficient Computation of Sparse Hessians using Automatic Differentiation

Assefaw H. Gebremedhin, Alex Pothen, Arijit Tarafdar
Department of Computer Science and Center for Computational Sciences
Old Dominion University, Norfolk, VA, USA
{assefaw@cs.odu.edu, pothen@cs.odu.edu, tarafdar@cs.odu.edu}

Andrea Walther
Institute of Scientific Computing, Technische Universität Dresden
D-01062 Dresden, Germany
Andrea.Walther@tu-dresden.de

Extended Abstract

Interior-point methods for solving nonlinear optimization problems require second-order derivatives of the Lagrangian function, and exact Hessians are needed in parametric sensitivity analysis. A Hessian can be computed accurately using automatic differentiation (AD), and when the sparsity structure of the Hessian is known, the computation can be made efficient using the following three-steps procedure.

1. Obtain a *seed* matrix S, which defines a column partition of a Hessian matrix H, via a specialized graph coloring.
2. Compute the numerical values of the entries of the *compressed* Hessian $B \equiv HS$ using AD.
3. Recover the numerical values of the entries of the original matrix H from the compressed representation B.

The coloring variant used in Step 1 depends on whether the recovery in Step 3 is *direct* (requires no further arithmetic) or *substitution-based* (relies on solving a set of simple triangular systems of equations): A direct method uses *star coloring* whereas a substitution method uses *acyclic coloring*. In both star and acyclic coloring, adjacent vertices are required to receive distinct colors. In addition, in a star coloring, every path on four vertices is required to use at least three colors; in an acyclic coloring, every cycle should use at least three colors. The objectives here are to use as few colors as possible, since the computational effort involved in employing AD in Step 2 is directly proportional to the number of colors used.

In a recent work we have designed and implemented novel and highly effective heuristic algorithms for these two NP-hard coloring problems [2]. The common key idea in these algorithms is the utilization of the structure of *two-colored induced subgraphs*, a collection of stars in the case of star coloring and a collection of trees in the case of acyclic coloring. With a careful choice of data

structures, the time complexity of these algorithms for a graph $G = (V, E)$ is within $O(|V|\overline{d}_2 \cdot \alpha)$, where $\overline{d}_2$ is the average degree-2 in G, the number of edges that are incident either on a vertex v or vertices adjacent to v, and α is the inverse of Ackermann's function. The acyclic coloring algorithm in particular makes use of the *union-find* (also known as *disjoint-set*) data structure. In an even more recent work, we have integrated our implementations of these coloring algorithms and the associated recovery routines with the AD software tool ADOL-C developed at the Technical University of Dresden, Germany [1]. The runtime of our recovery routines is linear in the number of edges in a graph (the number of nonzeros in the corresponding Hessian matrix).

In this talk we will discuss the algorithmic aspects of the three computational steps outlined earlier. We will also present experimental results that 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. The results we obtained show that the coloring techniques render enormous savings in runtime and make the computation of Hessians of very large size feasible (see the tables in the appendix). The results also show that a substitution-based evaluation that uses acyclic coloring leads to faster computation of Hessians than a direct method that relies on star coloring, considering the overall process. Remarkably, this speedup is achieved without compromising numerical accuracy, an advantage attributed to our careful use of two-colored trees in the recovery step.

References

[1] A.H. Gebremedhin, A. Pothen, A. Tarafdar, and A. Walter. Efficient Computation of Sparse Hessians: An Experimental Study using ADOL-C. 25 pages, Oct 2006. Submitted to INFORMS Journal on Computing.

[2] A.H. Gebremedhin, A. Tarafdar, F. Manne, and A. Pothen. New Acyclic and Star Coloring Algorithms with Application to Computing Hessians. 30 pages, 2006. Accepted by SIAM J. Sci. Comput.

Appendix: Computational Results

$\frac{n}{1000}$	dir S1	dir S2	dir S3	dir tot	sub S1	sub S2	sub S3	sub tot	dense
bd:									
1	0.01	2.1	0.001	2.1	0.01	1.1	0.01	1.2	80.0
5	0.03	5.2	0.006	5.2	0.06	2.9	0.06	3.1	1891.1
10	0.07	9.2	0.013	9.3	0.17	4.9	0.11	5.2	6725.5
20	0.13	17.4	0.026	17.6	0.56	9.5	0.23	10.2	***
40	0.27	33.4	0.051	33.7	2.02	18.2	0.48	20.7	***
60	0.39	49.0	0.077	49.5	4.26	26.4	0.72	31.4	***
80	0.53	65.8	0.103	66.4	7.43	35.5	0.97	43.9	***
100	0.66	81.4	0.128	82.2	11.44	44.3	1.22	57.0	***
rd:									
1	0.01	3.7	0.001	3.7	0.01	1.7	0.01	1.7	77.6
5	0.06	9.5	0.008	9.6	0.10	4.3	0.14	4.5	1740.2
10	0.19	18.2	0.016	18.4	0.28	7.8	0.41	8.5	6631.4
20	0.47	28.9	0.034	29.4	0.56	12.2	1.29	14.4	***
40	1.40	57.8	0.070	59.3	3.06	24.1	4.20	31.4	***
60	2.63	85.8	0.112	88.5	6.41	36.3	8.62	51.3	***
80	4.15	117.4	0.156	121.7	11.43	48.2	14.38	74.0	***
100	5.70	151.6	0.212	157.5	16.90	60.4	21.97	99.2	***

Table 1: Runtimes in seconds of the steps S1, S2, and S3. Results shown for two sparsity structures: a banded (bd) matrix of bandwidth $\rho = 11$, and a random (rd) matrix with average number of nonzeros per row $\overline{\rho} = 10.98$. The number of columns (n) in the Hessians considered range from 1000 to 100,000. The last column shows runtimes for computation without exploiting sparsity; the asterisks *** indicate that memory could not be allocated for the full Hessian.

ρ, $\overline{\rho}$	11, 10.98		21, 20.99	
	star	acyclic	star	acyclic
bd	11	6	21	11
rd	21 – 24	9 – 11	50 – 56	18 – 19

Table 2: Number of colors used by the star and acyclic coloring algorithms for *all* the problem dimensions n listed in Table 1. For the banded structure, both the star and the acyclic coloring algorithm we used find *optimal* solutions.