Algorithmic Differentiation Through Automatic Graph Elimination Ordering (ADTAGEO)

Jan Riehme[1], Andreas Griewank[2]

[1] Department of Computer Science, University of Hertfordshire, UK,
riehme@stce.rwth-aachen.de
[2] Department of Mathematics, Humboldt Universität zu Berlin, Germany,
griewank@mathematik.hu-berlin.de

Keywords. Automatic Differentiation, Instant Elimination, Live-DAG, symmetric Hessian-DAG, forward mode, reverse mode, checkpointing, ADTAGEO

Algorithmic Differentiation Through Automatic Graph Elimination Ordering (ADTAGEO) is based on the principle of *Instant Elimination*: Instead of storing a representation of the complete Computational Graph (*tape*-based approach of ADOL-C [1]) during the execution of the code to be differentiated and applying some elimination sequence afterwards, we dynamically maintain a DAG representing only active variables that are alive at any point in time. Whenever an active variable is deallocated or its value is overwritten the corresponding vertex in the Live-DAG will be eliminated immediately by the well-known vertex elimination rules [2].

Consequently, the total memory requirement is similar to that of the sparse forward mode. However, if local variables are destructed in the opposite order of their construction (as in `C++`), single assignment sequences of a code are in effect differentiated in reverse mode. Especially if compiler generated temporaries are destroyed in this way Instant Elimination leads to statement level reverse mode of ADIFOR [3] naturally.

More generally, the user determines the elimination order intentionally (or unintentionally) by the order in which he declares variables. This opens the opportunity to overcome the categorical separation between forward and reverse mode by creating hybrid modes by splitting the computation of derivatives in forward and reverse parts. So a trade-off can be achieved between the time consuming forward mode and the memory intensive reverse mode of AD.

If the subgraph spanned between the independent and dependent variables becomes bipartite by Instant Elimination than the desired derivatives are full accumulated in the Live-DAG. Thus no explicit initiation of sweeps is required. Note that a bipartite Live-DAG can be achieved often easily by restructuring the source code: Define a top-level routine with independents and dependents only as arguments and all other active variables locally in that top-level routine. Then Instant Elimination makes sure that all vertices located between independent and dependent variables will be removed on termination of the top-level routine execution.

Dagstuhl Seminar Proceedings 09061
Combinatorial Scientific Computing
http://drops.dagstuhl.de/opus/volltexte/2009/2085

On the other hand propagation routines can be implemented that allow to send initial tangents of independents (forward propagation), or initial adjoints of dependent variables (reverse propagation) through non-bipartite Live-DAGs (or subgraph only) at any point in time. Since a Live-DAG contains only variables alive propagation in a Live-DAG is much cheaper than a propagation within the complete Computational Graph. Note that a Live-DAG is build for the given evaluation point only, since Instant Elimination will incorporate the evaluation point into local partial derivatives determined by the vertex elimination rules. Nevertheless repeated Jacobian-vector products for a fixed evaluation point can be computed efficiently by propagation through the Live-DAG.

Second-order models of a function $Y = F(X)$ can be evaluated by extending the vertices of the Live-DAG with local Hessians and applying second-order elimination rules during vertex elimination. A Hessian-vector product $\bar{Y}F''\dot{X}$ can be computed by an initial forward propagation of tangent $\dot{X}$ in the Live-DAG resulting in $\dot{Y} = F'\dot{X}$. Afterwards a reverse sweep has to propagate both the given adjoint $\bar{Y}$ and the computed tangent $\dot{Y}$ to accumulate the Hessian-vector product. Note that the Live-DAG annotated with local Hessians stores one half of the symmetric Hessian graph only as suggested in [2].

Solving a least squares problem $\min_{(y,u)} \frac{1}{2}\|y - y^*\|^2$ s.t. $F(y,u) = 0$ with a Gauss-Newton algorithm needs derivatives at two points: Solving the system $F(y,u) = 0$ for fixed control u with Newton's method requires $\partial F(y,u)/\partial y$ (inner iteration). The system has to be solved at every step of the Gauss-Newton method (outer iteration), which itself requires the derivatives $\partial y(u)/\partial u$ to update the control u. Computing the nested derivatives $\partial y(u)/\partial u$ and $\partial F(y,u)/\partial y$ within a Live-DAG is absolut painless since sensitivities between live variables can be determined at any point in time.

The concept of maintaining a Live-DAG is very simple and fits optimally into the strategy of overloaded operators for classes, it is a very natural example of Object Oriented Programming. **ADTAGEO** is a proof of concept implementation in C++ using the `map`-class of the Standard Template Library (contact first author to get a copy).

Maintaining a Live-DAG is clearly a runtime issue. Nevertheless source transformation AD-tools might augment the generated AD-code with calls of runtime support routines that handle a graph structure and perform Instant Elimination. Thus, the concept of Live-DAGs is not restricted to AD-tools based on overloading.

References

1. Griewank, A., Juedes, D., Utke, J.: ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. ACM Trans. Math. Soft. **22** (1996) 131–167
2. Griewank, A.: Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation. SIAM (2000)
3. Bischof, C.H., Carle, A., Khademi, P., Mauer, A.: ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. IEEE Computational Science & Engineering **3** (1996) 18–32