Statement-Level Optimality of Tangent-Linear and Adjoint Models[1]

Uwe Naumann[2]

Abstract

We discuss a combinatorial problem that arises in the optimization of first-derivative code generated by exploiting the associativity of the chain rule. Our objective is to minimize the arithmetic complexity of such codes. A hierarchical approach is taken that ensures optimality locally at the level of single scalar assignments. New optimal algorithms are presented for three important special cases that represent the building blocks of any derivative code.

1 Introduction

We consider computer programs that implement mathematical models for simulating real-world processes in, for example, science, engineering, or economics. Such programs represent vector functions $F : I\!R^n \rightarrow I\!R^m$ that determine the numerical behavior of a set of outputs $\mathbf{y} = F(\mathbf{x})$ depending on the values of the inputs $\mathbf{x}$. The Jacobian of F is defined as the matrix of the partial derivatives of the outputs with respect to the inputs, that is,

$$F' = F'(\mathbf{x}) \equiv \left(\frac{\partial y_j}{\partial x_i}\right)_{i=1,\ldots,n}^{j=1,\ldots,m} \quad .$$

The *optimal Jacobian accumulation* problem [**?**] is to minimize the number of *fused multiply-add* operations $a + b \cdot c$ that are the main building blocks of any code for computing F', as shown in Section 2.

Tangent-linear and *adjoint* models are widely used to make the transition from the pure simulation to a possible optimization of objectives $\mathbf{y}$ with respect to parameters $\mathbf{x}$ [**?**, **?**, **?**]. A semantical source transformation technique known as *automatic differentiation* (AD) [**?**] can generate programs that implement the tangent-linear and adjoint models automatically

[1]This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38. Support was also provided by the National Science Foundation's Information Technology Research Program under Contract OCE-0205590 ("Adjoint Compiler Technology & Standards").

[2]Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439–4844, USA; naumann@mcs.anl.gov

from the original program. The *tangent-linear* model of F is defined as $\dot{\mathbf{y}} = F'(\mathbf{x}) \cdot \dot{\mathbf{x}}$, where $\dot{\mathbf{x}} \in I\!R^n$ is a direction in the input space of F. Similarly, the *adjoint* model of F is $\bar{\mathbf{x}} = (F'(\mathbf{x}))^T \cdot \bar{\mathbf{y}}$ with a vector of adjoints $\bar{\mathbf{y}} \in I\!R^m$ in the output space of F. A large part of the ongoing research in the field of AD is aimed at improving the efficiency of these derivative codes. The corresponding algorithms often require the solution of various discrete and combinatorial problems, including coloring [**?**, **?**, **?**] and hard elimination [**?**] problems.

In this paper we prove new results in the context of elimination techniques used in the evaluation of tangent-linear and adjoint models. The paper is structured as follows. In Section 2 we introduce the elimination problem, and we present a summary of the results. In Section 3 we present the technical details and we sketch the ideas of the corresponding proofs. In Section 4 we draw conclusions about the impact of this work.

2 Problem Description and Summary of Results

We assume that the program that implements F can be decomposed into a sequence of scalar assignments of the results of all arithmetic operations and calls of intrinsic functions to unique intermediate variables. Furthermore, we assume that these *elemental* functions φ_j, $j = 1, \ldots, q$, take at most two arguments which applies to most arithmetic operations and intrinsic functions of the high-level programming languages that are used to implement numerical simulations. The *code list* $v_j = \varphi_j(v_i)_{i \prec j}$, where $i \prec j$ if v_i is an argument of φ_j, can be *linearized* by computing local partial derivatives

$$c_{j,i} \equiv \frac{\partial \varphi_j(v_k)_{k \prec j}}{\partial v_i} \tag{1}$$

of φ_j with respect to all v_i with $i \prec j$ for $j = 1, \ldots, q$. This assumes that all elemental functions are jointly continuously differentiable in some neighborhood of the current argument. We set $c_{i,i} = 0$ for $i = 1 - n, \ldots, q$. The numbering of the code list variables is as follows: $x_i = v_{i-n}$ for $i = 1, \ldots, n$, $z_k = v_k$ for $k = 1, \ldots, p$, and $y_j = v_{p+j}$ for $j = 1, \ldots, m$. There are p *intermediate* variables, and we set $p + m = q$. The corresponding index sets are denoted by $X = \{1 - n, \ldots, 0\}$, $Z = \{1, \ldots, p\}$, and $Y = \{p + 1, \ldots, q\}$.

For example, consider a function $F : I\!R^3 \to I\!R$ that is implemented as the following scalar assignment:

$$y = \sin(x_1)^{x_1} \cdot x_1^{x_2} \cdot \frac{x_2}{x_3} \quad . \tag{2}$$

The corresponding code list $z_1 = v_1 = \sin(x_1)$; $z_2 = v_2 = x_1^{x_2}$; $z_3 = v_3 = \frac{x_2}{x_3}$; $z_4 = v_4 = v_1^{x_1}$; $z_5 = v_5 = v_2 \cdot v_4$; $y = v_6 = v_3 \cdot v_5$ is linearized by expressions for the local partial derivatives of the elemental functions, for example, $c_{1,-2} = \cos(x_1)$ and $c_{5,2} = v_4$.

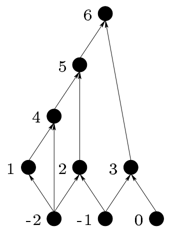

Figure 1: **G**

A directed acyclic *computational graph* $\mathbf{G} = (V, E)$ is induced by the code list. We set $V = X \cup Z \cup Y$ and $E = \{(i,j), i, j \in V : i \prec j\}$. No parallel edges sharing the same source as well as the same target are allowed in **G**. This graph is linearized by attaching the local partial derivatives to the corresponding edges; that is, $(i,j) \in E$ is labeled with $c_{j,i}$. For the purpose of this paper it is sufficient to consider **G** from a purely structural point of view. We refer to the corresponding linearized computational graphs as *c-graphs.* The c-graph for Equation (2) is shown in Figure 1.

The *extended Jacobian* C of F is defined as

$$C = (c_{j,i})_{i=1-n,\ldots,p}^{j=1,\ldots,q} \quad .$$

For Equation (2) we get

$$C = \begin{pmatrix} c_{1,-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_{2,-2} & c_{2,-1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_{3,-1} & c_{3,0} & 0 & 0 & 0 & 0 & 0 \\ c_{4,-2} & 0 & 0 & c_{4,1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{5,2} & 0 & c_{5,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{6,3} & 0 & c_{6,5} \end{pmatrix} \quad .$$

The sparsity pattern of C corresponds exactly to the adjacency matrix of **G**. The nonzero entries in C are the local partial derivatives labeling the edges in **G**. The extended Jacobian is the matrix representation of the c-graph. For every result that holds on **G** there is an equivalent formulation for C in terms of linear algebra. The results presented in this paper are derived by using graph terminology.

With the *tangent-linear system* defined as

$$\begin{pmatrix} \dot{\mathbf{z}} \\ \dot{\mathbf{y}} \end{pmatrix} = C \cdot \begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \end{pmatrix} \tag{3}$$

the accumulation of F' can be regarded as its solution for $\dot{\mathbf{y}}$ in terms of $\dot{\mathbf{x}}$ as outlined in [**?**]. Similarly, the *adjoint system*

$$\begin{pmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{z}} \end{pmatrix} = C^T \cdot \begin{pmatrix} \bar{\mathbf{z}} \\ \bar{\mathbf{y}} \end{pmatrix} \quad . \tag{4}$$

can be solved for $\bar{\mathbf{x}}$ in terms of $\bar{\mathbf{y}}$. Again, we have the choice of presenting the results in this paper based on either the tangent-linear or the adjoint system. W.l.o.g. we use the tangent-linear system.

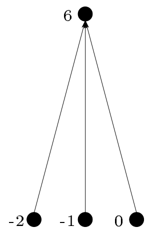

Figure 2: $\mathbf{G}'$

The solution of Equation (3) for $\dot{\mathbf{y}}$ in terms of $\dot{\mathbf{x}}$ is equivalent to transforming $\mathbf{G}$ into a bipartite form such that the labels on the remaining edges are exactly the nonzero entries of the Jacobian [?]. Figure 2 shows the bipartite c-graph $\mathbf{G}'$ that corresponds to the Jacobian of Equation (2). *Vertex* and *edge elimination* techniques have been developed to perform this transformation at a near-optimal computational cost [?]. Analogous elimination techniques can be defined on the extended Jacobian as shown in [?].

In $\mathbf{G}$ a vertex $i \in V$ is eliminated by connecting its predecessors with its successors [?]. An edge (i, k) with $i \prec j$ and $j \prec k$ is labeled with $c_{k,i} + c_{k,j} \cdot c_{j,i}$ if it existed before the elimination of j. We say that *absorption* takes place. Otherwise, (i, k) is generated as *fill-in* and labeled with $c_{k,j} \cdot c_{j,i}$ The vertex j is removed from $\mathbf{G}$ together with all incident edges. The procedure is illustrated in Figure 3(b), which shows the structural modifications resulting from the elimination of 3 in Figure 3(a). The new edge labels are $c_{4,1} = c_{4,1} + c_{4,3} \cdot c_{3,1}$, $c_{4,2} = c_{4,3} \cdot c_{3,2}$, $c_{5,1} = c_{5,3} \cdot c_{3,1}$, and $c_{5,2} = c_{5,3} \cdot c_{3,2}$. Vertex elimination in $\mathbf{G}$ is equivalent to *dyadic pivoting* in C [?].

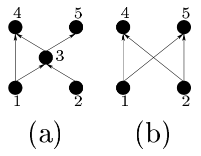

Figure 3: Vertex Elimination

An edge (i, j) is *front eliminated* by connecting i with all successors of j followed by removing (i, j) [?]. The corresponding structural modifications of the c-graph in Figure 3(a) are shown in Figure 4(a). The new edge labels are $c_{4,1} = c_{4,1} + c_{4,3} \cdot c_{3,1}$ and $c_{5,1} = c_{5,3} \cdot c_{3,1}$. Front elimination of edges in $\mathbf{G}$ is equivalent to *row pivoting* in C [?].

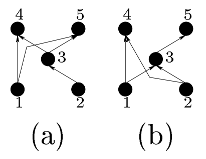

Figure 4: Edge Elimination

Similarly, $(i, j) \in E$ is *back eliminated* by connecting all predecessors of i with j [?]. The edge (i, j) itself is removed from $\mathbf{G}$. In Figure 4(b) the two in-edges of 4 are modified as $c_{4,1} = c_{4,1} + c_{4,3} \cdot c_{3,1}$ and $c_{4,2} = c_{4,3} \cdot c_{3,2}$. Back elimination of edges in $\mathbf{G}$ is equivalent to *column pivoting* in C [?].

Edge elimination eventually leads to intermediate vertices in $\mathbf{G}$ becoming *isolated*, that is, these vertices do not have any predecessors or successors anymore. Isolated vertices are simply removed from $\mathbf{G}$ together with all incident edges.

The transformation of $\mathbf{G}$ into $\mathbf{G}'$ using an elimination sequence σ yields the *Jacobian accumulation code*

$$c_{k_0,i_0} = \begin{cases} \frac{\partial \varphi_{k_0}}{\partial x_{i_0}} \text{ (as in Equation (1))} & \text{if } i_0 \prec k_0 \\ 0 & \text{otherwise} \end{cases}$$
$$c_{k_{\nu+1},i_{\nu+1}} = c_{k_\nu,i_\nu} + c_{k_\nu,j_\nu} \cdot c_{j_\nu,i_\nu} \quad \text{for } \nu = 0, \ldots, M(\sigma)-1 \quad .$$

$M(\sigma)$ denotes the number of fused multiply-add (`fma`) operations performed. This number is essentially equal to the number of multiplications performed, and it represents the objective with respect to which the Jacobian accumulation code is optimized. We cannot give a formal proof for the complexity of solving the tangent-linear system by using a minimal number of `fma`'s yet. However, we conjecture that this *c-graph elimination* (CGE) problem is NP-complete. So far, no polynomial algorithm exists that solves the problem exactly in general. The closely related problem of finding an elimination sequence that minimizes the fill-in for a pure vertex elimination strategy was shown in [**?**] to be NP-complete. Various methods for solving the CGE problem have been proposed, including heuristics [**?**], dynamic programming [**?**], and simulated annealing [**?**]. A generalization of the elimination techniques to the *dual c-graph* is presented in [**?**].

The evaluation of the tangent-linear model $\dot{\mathbf{y}} = F' \cdot \dot{\mathbf{x}}$ can be regarded as the solution of the *tangent-augmented tangent-linear system*

$$\begin{pmatrix} \tilde{\dot{\mathbf{x}}} \\ \dot{\mathbf{z}} \\ \dot{\mathbf{y}} \end{pmatrix} = \tilde{C} \cdot \begin{pmatrix} \tilde{x} \\ \tilde{\dot{\mathbf{x}}} \\ \dot{\mathbf{z}} \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{x}} & 0 \\ 0 & C \end{pmatrix} \cdot \begin{pmatrix} \tilde{x} \\ \tilde{\dot{\mathbf{x}}} \\ \dot{\mathbf{z}} \end{pmatrix} \tag{5}$$

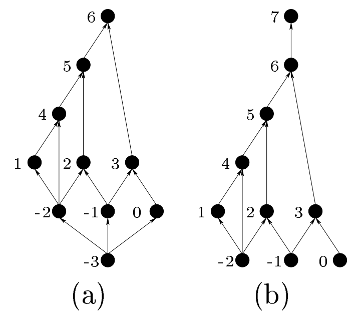

Figure 5: Augmentation

for $\dot{\mathbf{y}}$ in terms of $\tilde{x} \in I\!R$. A new auxiliary intermediate variable $\tilde{\dot{\mathbf{x}}} \in I\!R^n$ is introduced that corresponds to the original independent variable $\dot{\mathbf{x}}$ whose values now become entries of the new extended Jacobian $\tilde{C}$. In the *tangent-augmented c-graph* $\tilde{G} = (\tilde{V}, \tilde{E})$ a new independent vertex is introduced for $\tilde{x}$. New edges connect it with the vertices that represent the elements of $\tilde{\dot{\mathbf{x}}}$. These edges are labeled with the values of the corresponding elements of $\dot{\mathbf{x}}$. Figure 5(a) depicts the structure of this graph for Equation (2).

The adjoint model $\bar{\mathbf{x}} = (F')^T \cdot \bar{\mathbf{y}}$ can be evaluated by solving the *adjoint-augmented tangent-linear system*

$$\begin{pmatrix} \dot{\mathbf{z}} \\ \hat{\dot{\mathbf{y}}} \\ \hat{y} \end{pmatrix} = \hat{C} \cdot \begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \\ \hat{\dot{\mathbf{y}}} \end{pmatrix} = \begin{pmatrix} C & 0 \\ 0 & \bar{\mathbf{y}} \end{pmatrix} \cdot \begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \\ \hat{\dot{\mathbf{y}}} \end{pmatrix} \tag{6}$$

for $\hat{y} \in I\!R$ in terms of $\dot{\mathbf{x}}$. Again, a new auxiliary intermediate variable $\hat{\dot{\mathbf{y}}} \in I\!R^m$ is introduced together with the new dependent variable $\hat{y}$. In the *adjoint-augmented c-graph* $\hat{G} = (\hat{V}, \hat{E})$ the edges connecting the elements of $\hat{\dot{\mathbf{y}}}$ with $\hat{y}$ are labeled with the corresponding values of $\bar{\mathbf{y}}$. The structure of this graph for Equation (2) is shown in Figure 5(b).

To find a good approximation for the solution of the CGE problem, we choose a hierarchical approach [**?**] that ensures local optimality at the level of single assignment statements $y = F(\mathbf{x})$, $y \in I\!R$, while applying some approximation methods globally. For the solution of the augmented tangent-linear systems the question arises whether the optimal preaccumulation of all local gradients followed by the propagation of tangents or adjoints is in fact optimal. The answer is "no."

In scalar assignments, such as Equation (2), any intermediate value that is computed cannot be used more than once. In other words, all intermediate vertices in the c-graph have exactly one successor. This *single expression use* property is exploited in Section 3 to derive the main results of this paper. Under the assumption that all elemental functions are at most binary, we prove the optimality of the following three algorithms.

Algorithm A : Optimal Gradients of Scalar Assignments The local gradient of a scalar assignment is preaccumulated optimally as follows: (a) Eliminate all intermediate vertices j with a single predecessor for $j = 1, \ldots, p$. (b) Eliminate the remaining vertices in reverse order, that is, for $j = p, \ldots, 1$.

The resulting bipartite graphs of all scalar assignments in the program form a reduced c-graph of the entire function at a given point. Some approximation algorithm can be applied to find a near-optimal elimination sequence that transforms this c-graph into a bipartite form.

An optimal elimination sequence for the c-graph of Equation (2) is $(1, 4, 5, 3, 2)$. Its cost adds up to eight `fma`'s. A pure forward or backward vertex elimination sequence performs nine `fma`'s, respectively.

Algorithm B : Optimal Adjoints of Scalar Assignments Algorithm A can also be used to solve the adjoint-augmented tangent-linear system optimally for scalar assignments. Adjoints are propagated backward in the c-graph according to

$$\bar{\mathbf{x}} = \frac{\partial \hat{y}}{\partial \dot{\mathbf{x}}} \quad .$$

Note that the local gradient of the scalar assignment is not computed explicitly. For example, an optimal elimination sequence for the adjoint-augmented c-graph in Figure 5(b) is $(1, 4, 6, 5, 3, 2)$.

Algorithm C : Optimal Tangents of Scalar Assignments Directional derivatives can be propagated forward based on scalar assignments as

$$\dot{y} = \frac{\partial \dot{y}}{\partial \tilde{x}} \quad .$$

The number of `fma`'s required locally is minimized by the following algorithm: (a) Recursively, eliminate all intermediate vertices in $\tilde{\mathbf{G}}$ that have both a single predecessor and a single successor. Repeat for as long as such vertices exist in $\tilde{\mathbf{G}}$. (b) Eliminate the remaining intermediate vertices for $j = 1, \ldots, p$.

Again, the algorithm does not compute the local gradient of the scalar assignment. The optimal elimination sequence that is computed for the tangent-augmented c-graph in Figure 5(a) is $(0, 1, 4, -2, -1, 2, 3, 5)$.

3 Elimination Algorithms

Following the introduction of some notation, we prove two lemmas that are crucial for any further development.

For a given $\mathbf{G}$ we denote the cost of an optimal elimination sequence by $\underline{M}(\mathbf{G})$. The graph resulting from $\mathbf{G} = (V, E)$ after the insertion of some edge (i, j), $i, j \in V$, is denoted by $\mathbf{G} + (i, j)$. $\mathbf{G} - j$ denotes $\mathbf{G}$ after the elimination of a vertex $j \in V$. Similarly, $\mathbf{G} - H$ denotes $\mathbf{G}$ after the elimination of a set of vertices $H \subseteq V$. As before, the cost of an elimination sequence σ is denoted by $M(\sigma)$.

Lemma 1 *Let $\mathbf{G} = (V, E)$ be a linearized computational graph.*

1. $\underline{M}(\mathbf{G} + (i, j)) \geq \underline{M}(\mathbf{G})$ *for* $i, j \in V$.
2. *For* $i \notin X$ *or* $j \notin Y$ $\quad \underline{M}(\mathbf{G} + (i, j)) = \underline{M}(\mathbf{G}) \;\Leftrightarrow\; \exists \sigma : M(\sigma) = \underline{M}(\mathbf{G})$ *and σ generates (i, j) as fill-in.*

Proof First, we show that the insertion of a new edge (i, j) can never decrease the optimal cost. Therefore, let σ be an optimal elimination sequence for $\mathbf{G}$. Furthermore, let σ' be an elimination sequence for $\mathbf{G} + (i, j)$ such that $M(\sigma') < M(\sigma)$. Set $c_{j,i} = 0$ to make σ' an elimination sequence for $\mathbf{G}$. Its cost undercuts the cost of σ, thereby contradicting the assumption that σ is optimal for $\mathbf{G}$.

For the second part of the lemma we note that, obviously, inserting an edge (i, j) with $i \in X$ and $j \in Y$ does not increase the optimal cost.

" $\Leftarrow$ " : Let σ be an elimination sequence for $\mathbf{G}$ that generates (i, j) as fill-in. The application of σ to $\mathbf{G} + (i, j)$ adds a single scalar addition to the operations count. The length of the Jacobian code remains unchanged. Hence, σ is also an optimal elimination sequence for $\mathbf{G} + (i, j)$.

" $\Rightarrow$ " : Suppose that none of the optimal elimination sequences for $\mathbf{G}$ generates (i, j) as fill-in. Let σ be one of them. Furthermore, let σ' be an optimal elimination sequence for $\mathbf{G} + (i, j)$ such that $M(\sigma') = M(\sigma)$. Obviously, σ' must contain at least one statement that involves $c_{j,i}$ as a factor. Set $c_{j,i} = 0$ in σ'. The resulting elimination sequence is an elimination sequence for $\mathbf{G}$ that undercuts the cost of σ by at least one. This contradicts the assumption that σ is optimal for $\mathbf{G}$. $\square$

For $j \in V$ we define $P_j = \{i : i \prec j\}$ and $S_j = \{k : j \prec k\}$. We use $|S|$ to denote the cardinality of a set S.

Lemma 2 *Let $j \in V$ be such that $|P_j| \cdot |S_j| = 1$. Then $\underline{M}(\mathbf{G}) = \underline{M}(\mathbf{G} - j) + 1$.*

Proof Let $i \prec j \prec k$ in $\mathbf{G}$, and let the cost of eliminating j be equal to $|P_j| \cdot |S_j| = 1$. Consider any optimal elimination sequence σ for $\mathbf{G} - j$. Obviously, neither (i, j) nor (j, k) can be generated as fill-edges by σ because $j \notin \mathbf{G} - j$. By Lemma 1 we must have $\underline{M}(\mathbf{G}) > \underline{M}(\mathbf{G} - j)$. $\underline{M}(\mathbf{G} - j) + 1$ is the lowest possible value satisfying this inequality. $\square$

We say that the elimination of vertices j with $|P_j| \cdot |S_j| = 1$ *preserves optimality.*

3.1 Optimal Gradients of Scalar Assignments

Proposition 1 *Let $\mathbf{G}$ be a c-graph such that $\forall j \in Z : |P_j| \leq 2$ and $|S_j| = 1$. Let $\underline{M}_g$ be the cost of an elimination sequence that solves the CGE problem for $\mathbf{G}$. Then $\underline{M}_g = 2p - |H|$, where the set H is defined recursively as follows:*

1. *$H = \{i \in \mathbf{G} : |P_i| \cdot |S_i| = 1\}$;*

2. *if $(H = \varnothing)$ then* **exit** *else $\mathbf{G} = \mathbf{G} - H$; goto 1.*

Proof The recursive elimination of all vertices in H preserves optimality by Lemma 2. It remains to be shown that none of the vertices left can be eliminated at the cost of one `fma`. Eliminating them in reverse order as in Algorithm A ensures their elimination at a cost of two `fma`'s, and the proof of the proposition follows immediately.

Consider an arbitrary intermediate vertex j in $\mathbf{G} - H$. It must have exactly two predecessors. The elimination of either of its predecessors results in a decrease of the indegree of j only if this predecessor has a single predecessor itself. This is cannot be the case in $\mathbf{G} - H$. $\square$ Moreover, the elimination of some $i \in \mathbf{G}$ with $|P_i| \cdot |S_i| = 1$ can only lead to a decrease of the indegree of its successor. Consequently, we can perform a forward elimination of all vertices in H as in step 1 of Algorithm A.

3.2 Optimal Adjoints of Scalar Assignments

Proposition 2 *Let* $\mathbf{G}$ *be a c-graph such that* $\forall j \in Z : |P_j| \leq 2$ *and* $|S_j| = 1$. *Let* $\underline{M}_a$ *be the cost of an elimination sequence that solves the CGE problem for the adjoint-augmented c-graph* $\hat{\mathbf{G}}$. *Then* $\underline{M}_a = 2(p+1) - |H|$, *where the set* H *is defined for* $\hat{\mathbf{G}}$ *as in Proposition 1.*

The adjoint-augmented c-graph exhibits the single expression use property. Hence, the proof of Proposition 2 follows immediately.

3.3 Optimal Statement-Level Tangents

In general, tangent-augmented tangent-linear systems do not have the single expression use property.

Proposition 3 *Let* $\mathbf{G}$ *be a c-graph such that* $\forall j \in Z : |P_j| \leq 2$ *and* $|S_j| = 1$. *Let* $\underline{M}_t$ *be the cost of an elimination sequence that solves the CGE problem for the tangent-augmented c-graph* $\tilde{\mathbf{G}}$. *Then*

$$\underline{M}_t = |H| + \sum_{i \in X \setminus H} |S_i| + |Z \setminus H| \quad , \tag{7}$$

where the set H *is defined for* $\tilde{\mathbf{G}}$ *as in Proposition 1.*

Proof First we note that Equation (7) is equal to the cost yielded by Algorithm C. The minimal possible cost at which any vertex in $\tilde{\mathbf{G}}$ can be eliminated is one. Therefore, the number of intermediate vertices in $\tilde{\mathbf{G}}$ that is equal to $|\tilde{Z}| = |X| + |Z|$ represents a lower bound for the cost of solving the tangent-augmented tangent-linear system. This value can actually be achieved if $|S_i| = 1$ for all $i \in X$ because then $H = X \cup Z$. Starting from this situation, let us add some edge (i, j) to $\tilde{\mathbf{G}}$. W.l.o.g., let $i \in X$.[3] For the target j we can either have $j \in Y$ or $j \in Z$ such that $|P_j| = 1$ in $\tilde{\mathbf{G}}$.

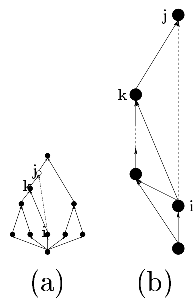

Figure 6: Proof

By Lemma 1, inserting (i, j) does not change the optimal cost only if (i, j) is generated as fill-in by an optimal elimination sequence for $\tilde{\mathbf{G}}$.

Let $j \in Y$. In order to generate (i, j) as fill-in an edge (k, j) with $(i, k) \in E$ needs to be back eliminated by some optimal elimination sequence for $\tilde{\mathbf{G}}$. However, any optimal elimination sequence for $\tilde{\mathbf{G}}$ eliminates i before k. Therefore, the optimal cost must be increased as a result of inserting (i, j). Algorithm C yields the minimal increase possible (an increase by one).

Let $j \in Z$ such that $|P_j| = 1$ in $\tilde{\mathbf{G}}$. The recursive elimination of all vertices in H preserves optimality by Lemma 2. $\tilde{\mathbf{G}} - H + (i, j)$ is depicted in Figure 6 (a). Obviously, $\underline{M}(\tilde{\mathbf{G}} - H + (i, j)) = 3$, and, again, the optimal cost is increased by one. Algorithm C yields the same cost.

Suppose that l edges have been inserted and that Equation (7) holds for the resulting graph. Insert (i, j) as the $(l + 1)$th edge such that, again and w.l.o.g., $i \in X$.

If $j \in Y$, then the recursive elimination of all vertices in H leads to the graph shown in Figure 6 (b). An optimal elimination sequence for $\tilde{\mathbf{G}} - H$ that creates (i, j) as fill-in does not exist. This observation is the result of generalizing the argument lead for the first edge that was inserted. By Lemma 1 the insertion of (i, j) must increase the optimal cost. Again, the application of Algorithm C yields a minimal increase of one.

For $j \in Z$ such that $|P_j| = 1$, $k \in P_j$ in $\tilde{\mathbf{G}}$ and $k \in \tilde{\mathbf{G}} - H$ we find by a similar argument that in an optimal elimination sequence i is always eliminated before k. Consequently, no optimal elimination sequence for $\tilde{\mathbf{G}}$ generates (i, j) as fill-in. The optimal cost must be increased, and this increase becomes minimal when applying Algorithm C. $\square$

[3] Alternatively, i could be equal to the single vertex in $\tilde{X}$. It is easily seen, however, that this situation is not relevant for the proof because the minimal number of `fma`'s is still equal to $|X| + |Z|$ in this case.

4 Conclusion

The efficiency of automatically generated derivative code plays an important role in modern numerical computing. One important contributing factor is the arithmetic complexity that is defined as the number of floating point operations performed by the derivative code. A variety of combinatorial optimization problems arise in this context. Polynomial algorithms are known for only few of them.

We presented three new algorithms for the optimal accumulation of derivative information at the level of scalar assignments. These results are being exploited as part of a hierarchical approach to the optimization of derivative code generated by next generation software tools for automatic differentiation. Their development is part of a collaborative research effort between MIT., Rice University, and the University of Chicago/Argonne National Laboratory that has been funded under NSF's ITR program. We expect the theoretical results of this paper to have a positive effect on the quality of these tools.