

An Integer Programming Approach to Optimal Derivative Accumulation

Jieqiu Chen, Paul Hovland, Todd Munson, and Jean Utke

Abstract In automatic differentiation, vertex elimination is one of the many methods for Jacobian accumulation. However, finding the optimal vertex elimination sequence of a computational graph is a hard combinatorial optimization problem. In this paper, we propose an integer programming (IP) technique to tackle this problem, and we develop an IP formulation for it. This enables us to use a standard integer optimization solver to find an optimal vertex elimination strategy. We hope to use the optimization formulation to evaluate the effectiveness of heuristics and find an optimal strategy for certain key computational kernels. In addition, we have developed several lower bound and symmetry-breaking constraints to strengthen the basic IP formulation. We demonstrate the effectiveness of these enhancements through computational experiments. We also consider the scarcity of a graph in the context of vertex elimination. A basic IP formulation for the scarcity problem and some preliminary computational results are presented.

Key words: Vertex-elimination, scarcity, integer programming

Jieqiu Chen

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. jieqchen@mcs.anl.gov

Paul Hovland

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. hovland@mcs.anl.gov

Todd Munson

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. tmunson@mcs.anl.gov

Jean Utke

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. utke@mcs.anl.gov

1 Introduction

Automatic differentiation (AD) is a family of methods for obtaining the derivatives of functions computed by a program [4]. AD couples rule-based differentiation of language intrinsics with derivative accumulation according to the chain rule. The associativity of the chain rule leads to many possible “modes” of combining partial derivatives, such as the forward mode and reverse mode. Exponentially many hybrid, or cross-country modes are possible, and finding the optimal Jacobian accumulation strategy is NP-hard [8]. Therefore, all AD tools employ some sort of heuristic strategy. The most popular heuristics are pure forward mode, pure reverse mode, and a hierarchical strategy using the forward mode overall but “preaccumulating” the derivatives of small program units (often statements or basic blocks).

Algorithms for automatic differentiation are often expressed in terms of the computational graph. If the edges of the computational graph are assigned weights equal to partial derivatives, then the optimal Jacobian accumulation problem is reduced to finding an optimal order in which to combine edge weights. A simplified version of this problem is to find an optimal vertex elimination strategy, where a vertex is eliminated by combining all in-edges with all out-edges, requiring $|\text{in}| \times |\text{out}|$ multiplications; see Sect. 2.1 for more details. The optimal Jacobian accumulation problem requires eliminating all intermediate vertices of the computational graph. A closely related problem is to find the *scarcity* of a computational graph, which arises when Jacobian-vector multiplication is needed and forming the full Jacobian matrix is not desired; see page 227 of [6] for a formal definition. We consider the scarcity problem in the context of vertex elimination. In other words, we aim to find the graph with the smallest number of edges by removing a subset of the intermediate vertices. Although vertex elimination greatly reduces the number of ways to combine edge weights and seems to be an “easier” problem than the original one, it is still combinatorial and is speculated to be NP-complete. The scarcity problem seems to be a problem at least as hard as the vertex-elimination problem.

In this paper, we propose to use integer programming (IP) to tackle the vertex elimination problem and the scarcity problem. The motivation of using IP is that it is a powerful optimization technique and has been successfully applied to solve many hard combinatorial optimization problems, for example, the traveling salesman problem [2]. Specifically, IP deals with problems of minimizing a function of many variables subject to (1) linear inequality and equality constraints and (2) integrality restrictions on the variables [11]. IP is usually stated as

$$\min\{c^T x : Ax \leq b, x \in Z_+^n\}, \quad (1)$$

where Z_+^n is the set of nonnegative integral n -dimensional vectors and $x = (x_1, \dots, x_n)$ are the *variables*. The generality of (1) allows it to model a wide variety of combinatorial optimization problems, for example, the traveling salesman problem [7], the minimum-weight spanning tree problem, and the set partitioning problem [11]. If one drops the integrality restriction on x , then the resulting problem becomes a *linear programming* (LP) relaxation of (1). Modern LP solvers can solve large-scale

LPs quickly and reliably. Although integer programming is NP-hard, the advanced technology of solving LPs allows one to solve (1) through a systematic branch-and-bound method in conjunction with solving LP relaxation (to obtain lower bounds) at each node of the branch-and-bound tree.

We develop integer programming formulations of the optimal vertex elimination problem and the scarcity problem. These IP formulations enable us to use existing IP solvers to find an optimal vertex elimination strategy or the scarcity of a computational graph. Our objective is not to replace the elimination heuristics used in AD tools, since finding the optimal elimination strategy for all basic blocks would be prohibitively expensive. Rather, we aim to use the optimization formulation to evaluate the effectiveness of heuristics and find an optimal strategy for certain key computational kernels. In particular, the optimal computational cost of the vertex elimination can be used to measure whether the heuristic solution is close enough to the optimal one; and knowing the optimal scarcity of a key computational kernels might reduce computational efforts significantly. For the vertex elimination problem, we additionally develop several computational techniques to enhance the basic IP formulation.

The paper is organized as follows. Section 2 present the IP formulations for the vertex elimination and the scarcity problem. In particular, Sect. 2.1 introduces vertex elimination; Sect. 2.2 models the vertex elimination problem as an integer program; Sect. 2.2.1 discusses several enhancements to the basic IP formulation of the vertex elimination problem; and Sect. 2.3 introduces the scarcity problem and presents one IP formulation of it. Section 3 presents computational results of solving the IP formulations of several small problems. Section 4 summarizes our work and briefly describes future areas for research.

2 Integer Programming Formulations

In this section, we first briefly introduce vertex elimination. Next, we describe how we model vertex elimination as an integer program. Then we show how to transform several known lower-bound results of vertex elimination into linear constraints that strengthen the IP formulation; in addition, we propose symmetry-breaking constraints that eliminate many equivalent solutions. Lastly, we present a basic IP formulation for the scarcity problem in the context of vertex elimination.

2.1 Introduction to vertex elimination

We assume the readers have a basic knowledge of AD and in which context the computational graph arises; the readers can refer to [6] and [9] for more background information. Consider the computational graph $G = (V, E)$ induced by a computer program that implements $F : \mathfrak{R}^n \rightarrow \mathfrak{R}^m$. The vertex set $V = \{1 - n, \dots, p + m\}$ and can

be partitioned into $X = \{1 - n, \dots, 0\}$, $Z = \{1, \dots, p\}$, and $Y = \{p + 1, \dots, p + m\}$, representing the set of n independent variables, p intermediate variables, and m dependent variables of the computer program F , respectively. E encodes the dependence relationship among the variables. In particular, if the j th variable is defined as a elementary function of the i th variable (and possibly other variables) in F , then $(i, j) \in E$. The partial derivative of the j th variable with respect to the i th variable is usually assigned as an edge weight for (i, j) , $\forall (i, j) \in E$.

It is well known that the Jacobian matrix $F' = F'(x) \in \mathfrak{R}^{m \times n}$ can be accumulated by some elimination techniques that transform G into a bipartite graph $G' = (\{X, \emptyset, Y\}, E')$. Vertex elimination is one such technique; see [5]. For a given intermediate vertex k , let P_k and S_k denote the set of predecessors and successors of k , respectively. Then eliminating vertex k involves (1) $\forall i \in P_k, j \in S_k$, multiply the edge weight of (i, k) with that of (k, j) ; (2) add a new edge (i, j) to the graph and assign the multiplied weight to (i, j) (if (i, j) already exists, add the multiplied edge weight to the original edge weight of (i, j)); and (3) remove vertex k and its incident edges from the graph. Eliminating vertex k thus requires $|P_k| \times |S_k|$ number of floating-point multiplications. After eliminating all intermediate vertices, the edge weights of the resulting bipartite graph are exactly the entries of F' . The total number of multiplications in vertex elimination is an approximation of the computational cost of accumulating the Jacobian matrix.

Naturally, one wishes to minimize the total number of multiplications required to eliminate all the intermediate vertices. Note that different elimination sequences might result in different numbers of multiplications. Finding the best elimination sequence among the $p!$ possible ones is a combinatorial problem.

2.2 IP formulation

Define $T = \{1, \dots, p\}$ to be a time set. For any $t \in T$, we use $G(t) = (V(t), E(t))$ to represent the computational graph after eliminating t intermediate vertices. Denote $G(0) = (V(0), E(0)) = G$. Note that $E(t)$ is undetermined unless a vertex elimination sequence is provided. We let $\tilde{E} = \cup_{t=0}^p E(t)$ denote the set of edges that could exist in the vertex elimination process. To model the fact that $G(t)$ is undetermined, we use variable c to represent the edges at time t :

$$c_{ijt} = \begin{cases} 1, & \text{if } (i, j) \in E(t) \\ 0, & \text{otherwise} \end{cases}. \quad (2)$$

Similarly, let variable d denote the edge deleted, and variable f denote the edge *generated*, where we say an edge (i, j) is generated if (i, j) is formed by combining an in edge and an out edge of an eliminated vertex; we use x to model the elimination sequence:

$$d_{ijt} = \begin{cases} 1, & \text{if } (i, j) \in E(t-1) \text{ and is deleted after removing a vertex} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$$f_{ijt} = \begin{cases} 1, & \text{if } (i, j) \in E(t) \text{ is generated after removing a vertex} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

$$x_{it} = \begin{cases} 1, & \text{if eliminate vertex } i \text{ at time } t \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

We represent the initial graph with the parameter

$$c_{ij0} = \begin{cases} 1, & \text{if } (i, j) \in E(0) \\ 0, & \text{otherwise} \end{cases}.$$

With these notation, the vertex elimination problem is formulated as (MinFlops).

$$\text{minimize } F = \sum_{t \in T} \sum_{(i,j) \in \tilde{E}} f_{ijt} \quad (\text{MinFlops})$$

$$\text{subject to } \sum_{i \in V} x_{it} = 1 \quad \forall t \in T \quad (6)$$

$$\sum_{t \in T} x_{it} = 1 \quad \forall i \in Z \quad (7)$$

$$x_{it} = 0 \quad \forall i \in X \cup Y, \forall t \in T \quad (8)$$

$$\left. \begin{aligned} d_{ijt} &\geq x_{it} + c_{ij(t-1)} - 1 \\ d_{ijt} &\geq x_{jt} + c_{ij(t-1)} - 1 \\ d_{ijt} &\leq x_{it} + x_{jt} \\ d_{ijt} &\leq c_{ij(t-1)} \end{aligned} \right\} \quad \forall (i, j) \in \tilde{E}, \forall t \in T \quad (9)$$

$$f_{ijt} \geq d_{ikt} + d_{kjt} - 1 \quad \forall (i, j) \in \tilde{E}, \forall k \in Z, \forall t \in T \quad (10)$$

$$\left. \begin{aligned} c_{ijt} &\geq f_{ijt} \\ c_{ijt} &\leq 1 - d_{ijt} \\ c_{ijt} &\leq c_{ij(t-1)} + (f_{ijt} + d_{ijt}) \\ c_{ijt} &\geq c_{ij(t-1)} - (f_{ijt} + d_{ijt}) \end{aligned} \right\} \quad \forall (i, j) \in \tilde{E}, \forall t \in T \quad (11)$$

$$x_{it} \in \{0, 1\} \quad \forall i \in V, \forall t \in T \quad (12)$$

$$c_{ijt}, d_{ijt}, f_{ijt} \in \{0, 1\} \quad \forall (i, j) \in \tilde{E}, \forall t \in T \quad (13)$$

The objective function of (MinFlops) is the sum of the number of edges generated in all time periods, which is equal to the total number of multiplications. Constraints (6) ensure that at any time period we eliminate exactly one vertex, and constraints (7) ensure that every intermediate vertex is eliminated at some time period. Constraints (8) enforce that independent or dependent vertices cannot be eliminated.

Constraints (9) define the edges to be eliminated at each time period. In particular, the first (second) inequality means if vertex i (j) is eliminated at time t and (i, j) exists at time $t-1$, then (i, j) is a deleted edge at time t . The third inequality ensures that the edge (i, j) can only be an deleted edge at time t if either vertex i or j is eliminated at time t , indicated by the values of x_{it} and x_{jt} . The fourth inequality

means that only edges existing in the previous time period can be eliminated. Constraints (10) ensure that if both edge (i, k) and (k, j) are eliminated edges, then (i, j) must be generated by combining (i, k) and (k, j) .

Constraints (11) enforce the proper relationship between $G(t-1)$ and $G(t)$. In particular, if the edge (i, j) is generated at time t , then $(i, j) \in E(t)$, which is enforced through the first inequality of (11). Similarly, if (i, j) is an deleted edge at time t , then $c_{ijt} \leq 1 - d_{ijt}$ forces $(i, j) \notin E(t)$. The last two inequalities of (11) ensure that all the other edges that are not incident to the eliminated vertex at time $t-1$ also exist in time t . Constraints (12) and (13) restrict all variables to be binary. Overall, constraints (6)–(13) model the vertex elimination process. Any optimal solution (x^*, d^*, f^*, c^*) to (MinFlops) specifies a vertex elimination sequence with the minimum multiplications required to make G bipartite.

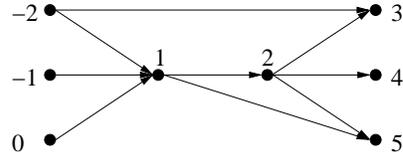


Fig. 1 Graph for example 1 and 2

EXAMPLE 1. We take Fig. 1 as an example: $X = \{-2, -1, 0\}$, $Y = \{3, 4, 5\}$, and $Z = T = \{1, 2\}$. The solution of the integer program (MinFlops) is as follows:

$$\begin{aligned}
 & x_{2,1} = x_{1,2} = 1 \text{ (remove 2 at } t = 1 \text{ and remove 1 at } t = 2) \\
 t = 1 : & d_{1,2,1} = d_{2,3,1} = d_{2,4,1} = d_{2,5,1} = 1 \\
 & f_{1,3,1} = f_{1,4,1} = f_{1,5,1} = 1 \\
 & c_{-2,1,1} = c_{-1,1,1} = c_{0,1,1} = c_{-2,3,1} = c_{1,3,1} = c_{1,4,1} = c_{1,5,1} = 1 \\
 t = 2 : & d_{-2,1,2} = d_{-1,1,2} = d_{0,1,2} = d_{1,3,2} = d_{1,4,2} = d_{1,5,2} = 1 \\
 & f_{-2,3,2} = f_{-2,4,2} = f_{-2,5,2} = f_{-1,3,2} = f_{-1,4,2} = f_{-1,5,2} = f_{0,3,2} = f_{0,4,2} = f_{0,5,2} = 1 \\
 & c_{-2,3,2} = c_{-2,4,2} = c_{-2,5,2} = c_{-1,3,2} = c_{-1,4,2} = c_{-1,5,2} = c_{0,3,2} = c_{0,4,2} = c_{0,5,2} = 1
 \end{aligned}$$

where all the other entries of (x, d, f, c) not listed above take the value of zero. The minimum number of multiplications required is thus $\sum_{(i,j) \in \tilde{E}, t=1} f_{ijt} + \sum_{(i,j) \in \tilde{E}, t=2} f_{ijt} = 3 + 9 = 12$.

2.2.1 Computational consideration for solving (MinFlops)

Define $q = n + m + p$. Since $G(t)$ is unknown for any $t \in T$, the cardinality of \tilde{E} is on the order of $\mathcal{O}(q^2)$. It then follows that the number of variables defined in (MinFlops) is on the order of $\mathcal{O}(q^2 p)$ and the number of constraints on the order of $\mathcal{O}(q^2 p^2)$. The size of the resulting integer program grows rapidly as the number of vertices and edges in the graph increase, making the integer program challenging to solve. In this subsection, we discuss these methods to help computationally solve

(MinFlops) with a standard IP solver: reducing the number of constraints, developing valid lower bounds, and developing symmetry-breaking constraints.

Reducing the number of constraints The large size of the IP is partly due to the fact that $G(t), \forall t \in T$ is unknown without fixing a vertex elimination sequence and thus $|\tilde{E}|$ is large. By definition, $\tilde{E} = \cup_{t=0}^p E(t)$ can be determined by enumerating all vertex elimination sequences, which is impractical.

Proposition 1. *Let $G^* = (V, E^*)$ be the transitive closure of G . Then $\tilde{E} \subseteq E^*$.*

Proof. For any $(i, j) \in \tilde{E}$, there must exist a path in G that connects vertex i and j , and thus $(i, j) \in E^*$.

Since the transitive closure of a graph is easy to compute, say, by Floyd-Warshall's algorithm, we use E^* instead of \tilde{E} when computation is involved. The number of constraints in (10) grows the fastest as the graph becomes bigger and is equal to the cardinality of the set

$$\mathcal{J} = \{(i, j, k, t) : (i, j) \in E^*, k \in Z, t \in T\}. \quad (14)$$

Here an implicit assumption is that any edge in E^* could exist in any time period. But this is not true. We can reduce the number of constraints in (10) by taking into account that some edges in E^* cannot exist in certain time periods, as shown in the following proposition.

Proposition 2. *Let l_{ij} denote the length of the shortest path that connects vertex i and j in G , $\forall (i, j) \in E^*$. If $l_{ij} \geq 2$, then $(i, j) \notin E(t), \forall t < l_{ij} - 1, t \in \{0\} \cup T$.*

Proof. Every time a vertex k is removed, the length of the shortest path between i and j can be shortened by 1 if k is on the shortest path (or one of the shortest paths) between i and j . Otherwise, the length of the shortest path does not change. Thus for any $(i, j) \in E^*$ and $l_{ij} \geq 2$, we need to remove at least $l_{ij} - 1$ vertices in order to form a direct edge between i and j , which proves the result.

This proposition implies that we can reduce the number of constraints by defining

$$\tilde{\mathcal{J}} = \{(i, j, k, t) : (i, j) \in E^*, t \geq l_{ij} - 1, t \in T, \forall k \in Z\},$$

and replacing (10) with

$$f_{ijt} \geq e_{ikt} + e_{kjt} - 1, \quad \forall (i, j, k, t) \in \tilde{\mathcal{J}}. \quad (15)$$

We comment that $|\tilde{\mathcal{J}}|$ could be much smaller than $|\mathcal{J}|$, which allows one to solve larger instances.

Developing valid lower bounds State-of-the-art IP solvers utilize a branch-and-bound (B&B) approach to solve integer programs. Tight lower bounds are crucial in reducing the solution time because the B&B approach relies on lower bounds (for minimization problem) obtained by solving the linear relaxations of an IP and

a global upper bound (GUB) to prune nodes in the B&B tree. If the lower bound of a node is greater than GUB, then the node contains no better solution than the current best one associated with the GUB and thus can be pruned. We consider two known results about the cost of vertex elimination and reformulate them as linear constraints in terms of the variables x and f . Let $[X \mapsto k]$ denote the set of paths connecting the independent vertices and k , and let $k \mapsto Y$ be the set of paths connecting k and the dependent vertices. The first known result is as follows.

Observation 1 *For any $k \in Z$, the cost of eliminating vertex k last, regardless of the vertex elimination ordering of all the other intermediate vertices, is $|X \mapsto k| \cdot |k \mapsto Y|$.*

From now on, we use γ_k to represent $|X \mapsto k| \cdot |k \mapsto Y|$. Although we do not know which vertex is eliminated last, x allows the flexibility of choosing any vertex k as the last one to eliminate, and $\sum_{k \in Z} \gamma_k x_{kp}$ represents the cost of eliminating the last vertex. The following valid inequality for vertex elimination can be added to (MinFlops) to strengthen the formulation:

$$F \geq \sum_{t \in T \setminus \{p\}} \sum_{(i,j) \in E^*} f_{ijt} + \sum_{k \in Z} \gamma_k x_{kp}, \quad (16)$$

where F is the variable representing the total cost of vertex elimination, and the first summation on the right-hand side is the cost of removing the first $p-1$ vertices. At first glance, the terms on both sides of the inequality seem to represent the same quantity. However, when computationally solving IP and the associated LP relaxations, all the integral restrictions on the variables are dropped, and so the right-hand side becomes a valid lower bound.

The second known result is established in [9]. Using the same notation as in [9], let $X-k$ be the minimum vertex cut between X and k , and let $k-Y$ be the minimum vertex cut between k and Y .

Observation 2 (Lemma 3.1 & 3.2 [9]) *The number of multiplications required to eliminate vertex k , among all possible vertex elimination sequences, is underestimated by $|X-k| \cdot |k-Y|$; the minimal number of multiplications required for the transformation $G \rightarrow G'$ is greater than or equal to $\sum_{k \in Z} |X-k| \cdot |k-Y|$.*

From now on, we use λ_k to denote $|X-k| \cdot |k-Y|$, $\forall k \in Z$. One immediate implication of Observation 2 is $F \geq \sum_{k \in Z} \lambda_k$. Although this inequality is valid, computationally it is not useful. The reason is that only one variable F is involved in this inequality, and this inequality does not cut off any fractional solutions of the LP relaxations, where we say a solution $(\bar{x}, \bar{c}, \bar{d}, \bar{f})$ is fractional if one or more of components of the solution have nonintegral values. Instead, we express the results in Observation 2 as

$$F \geq \sum_{t < s} \sum_{(i,j) \in E^*} f_{ijt} + \sum_{s \leq t' \leq p} \sum_{k \in Z} \lambda_k x_{kt'}, \quad \forall s \in T, \quad (17)$$

where the second term on the right-hand side is a lower bound on the cost of removing the last $p-s+1$ vertices. We comment that (17) define p constraints that have

the same meaning but might have different effects computationally because each constraint involves different components of x and f and so might cut off different fractional solutions of the LP relaxations.

Developing symmetry-breaking constraints It is easy to understand that some elimination sequences may result in the same number of multiplications. However, usually we need only one optimal sequence. The standard IP solvers cannot recognize the equivalence of two sequences and will waste considerable time exploring different branches of the B&B tree with the same optimal values. We consider one situation where two equivalent elimination sequences occur.

Observation 3 *If vertex i and j are not adjacent at time t , $\forall i, j \in Z, i < j, \forall t \in T$, then the following two elimination sequences give the same number of multiplications: (1) eliminate vertex i at time t and vertex j at time $t + 1$, and (2) eliminate vertex j at period t and vertex i at period $t + 1$; all the other vertices are eliminated in the same order.*

We call the two sequences in Observation 3 *symmetric*, and we develop symmetry-breaking constraints that permit only one of the two sequences.

Proposition 3. *The following constraints are violated by one of the two symmetric sequences in Observation 3:*

$$x_{jt} + x_{i(t+1)} - c_{ijt} \leq 1, \quad \forall i, j \in Z, i < j, \forall t \in T \setminus \{p\}. \quad (18)$$

Proof. If i and j are not adjacent at time t , then $c_{ijt} = 0$. Thus (18) becomes $x_{jt} + x_{i(t+1)} \leq 1$, which is invalid for the sequence that eliminates j at time t and then i at time $t + 1$, in the other words, $x_{jt} = x_{i(t+1)} = 1$. Obviously the sequence that eliminates i at time t and j at time $t + 1$, namely, $x_{it} = x_{j(t+1)} = 1$, is permitted.

We point out that we do not know beforehand whether two vertices i and j are adjacent at time t . Constraints (18) should also be valid in the case where i and j are adjacent at time t . If $(i, j) \in E(t)$, then c_{ijt} is expected to have value 1. Then (18) becomes

$$x_{jt} + x_{i(t+1)} - 1 \leq 1 \quad \text{or} \quad x_{jt} + x_{i(t+1)} \leq 2,$$

which permits all four possible combinations of x_{jt} and $x_{i(t+1)}$.

2.3 Scarcity

The scarcity problem arises when partially accumulating the Jacobian matrix gives overall lower cost than accumulating the full Jacobian matrix does; see Section 10.3 of [6] for more details on motivation and context. Here we make one simplification of the scarcity problem. In particular, we consider the scarcity problem in the context of vertex elimination; we do not consider the scarcity under edge elimination or any other accumulation techniques. Thus the same notation as introduced in

Sect. 2.1 and Sect. 2.2 is applicable. The problem reduces to finding the graph $G(t)$ in the transformation process $G(0) = G \rightarrow G' = G(p)$ with the minimum number of edges. And the minimum number of nonunitary edges can be found easily once we know $G(p)$.

Since the objective of the scarcity problem is different from that of the vertex elimination problem, we need additional variables besides those defined in (2)–(5). Define the absorbed edge at time t as

$$b_{ijt} = \begin{cases} 1, & \text{if } (i, j) \in E(t-1) \text{ and } (i, j) \text{ is generated at time } t \\ 0, & \text{otherwise} \end{cases} .$$

We formulate the scarcity problem as (Scarcity). The objective here is to maximize the reduction in edge count:

$$\sum_{t \in T} \sum_{(i,j) \in \tilde{E}} (d_{ijt} - f_{ijt} + b_{ijt}) = \sum_{t \in T} \left[\sum_{(i,j) \in \tilde{E}} d_{ijt} - \sum_{(i,j) \in \tilde{E}} (f_{ijt} - b_{ijt}) \right],$$

where the first and the second summations in the square brackets represent the number of edges eliminated at time t and the new edges added in time t , respectively. Note that through maximizing the reduction in edge count, we achieve the goal of minimizing the number of edges in the graph. The variable b is introduced here to model the absorbed edges, which does not change the edge count. Constraints (19) and (20) are similar to (6) and (7) but allow one to eliminate some, but not all, of the intermediate vertices. Constraint (21) enforces that the vertices have to be eliminated in consecutive time periods, mainly for easier interpretation. Constraints (22) and (23) properly define the generated edges and absorbed edges, respectively. Compared with (10), constraint (22) has two additional inequality to ensure that f_{ijt} takes value zero when (i, j) is not a generated edge at time t . We also enforce constraints (8), (9), (11), (12), and (13) of (MinFlops) here because they can be used without any change to model the vertex elimination process.

$$\begin{aligned}
& \text{maximize} && R = \sum_{t \in T} \sum_{(i,j) \in \tilde{E}} (d_{ijt} - f_{ijt} + b_{ijt}) && \text{(Scarcity)} \\
& \text{subject to} && (8), (9), (11), (12), (13) \\
& && \sum_{i \in V} x_{it} \leq 1 \quad \forall t \in T && (19) \\
& && \sum_{t \in T} x_{it} \leq 1 \quad \forall i \in Z && (20) \\
& && \sum_{i \in Z} x_{it} \geq \sum_{i \in Z} x_{i(t+1)} \quad \forall t \in T \setminus \{p\} && (21) \\
& && \left. \begin{aligned} f_{ijt} &\geq d_{ikt} + d_{kjt} - 1 \\ f_{ijt} &\leq \sum_{v \in V} d_{ivt} \\ f_{ijt} &\leq \sum_{v \in V} d_{vjt} \end{aligned} \right\} \quad \forall (i,j) \in \tilde{E}, \forall k \in Z, \forall t \in T && (22) \\
& && \left. \begin{aligned} b_{ijt} &\geq f_{ijt} + c_{ij(t-1)} - 1 \\ b_{ijt} &\leq f_{ijt} \\ b_{ijt} &\leq c_{ij(t-1)} \end{aligned} \right\} \quad \forall (i,j) \in \tilde{E}, \forall t \in T && (23) \\
& && b_{ijt} \in \{0, 1\}, \quad \forall (i,j) \in \tilde{E}, \forall t \in T. && (24)
\end{aligned}$$

EXAMPLE 2. Again take the computational graph in Fig. 1 for example. The solution of integer program (Scarcity) is

$$\begin{aligned}
x_{2,1} &= 1 \text{ (remove only vertex 2)} \\
d_{1,2,1} &= d_{2,3,1} = d_{2,4,1} = d_{2,5,1} = 1 \\
f_{1,3,1} &= f_{1,4,1} = f_{1,5,1} = 1 \\
b_{1,5,1} &= 1,
\end{aligned}$$

where we did not list c since its value can be inferred from (x, d, b, f) , and the other components of (x, e, b, f) not listed above take the value of zero. Thus $G(1)$ contains the minimum number of edges. The maximum reduction in edge count is

$$\sum_{t \in T} \left[\sum_{(i,j) \in \tilde{E}} d_{ijt} - \sum_{(i,j) \in \tilde{E}} (f_{ijt} - b_{ijt}) \right] = 4 - (3 - 1) = 2.$$

Note that $|E(0)| = 9$ minus the number of edge reduction 2 gives the edge count of $G(1)$, which is 7. Another way to recover the minimum edge count is to calculate $\sum_{(i,j) \in \tilde{E}} c_{ij1}$, where c is readily available from solving (Scarcity).

3 Computational experiments

In this section, we present computational results for solving (MinFlops) and (Scarcity). We collect several small problems from [4], where the optimal vertex elimination and scarcity can be verified by hand. Here the main goals are threefold: (1) to il-

illustrate how to apply the IP formulations we have developed to solve the vertex elimination and the scarcity problem, (2) to compare the optimal values found by the IP solver with heuristic solutions, and (3) to demonstrate the effects of the computational methods developed in Sect. 3. Both the IP models are formulated with GAMS [10] and solved with XPRESS 22.01 [1] on a standard computer under the Linux system.

3.1 Optimal vertex elimination

We compare four IP models for the optimal vertex elimination,

$$\min \{F : (6) - (13)\} \quad (M_0)$$

$$\min \{F : (6) - (9), (11) - (13), (15)\} \quad (M_1)$$

$$\min \{F : (6) - (9), (11) - (13), (15), (16), (17)\} \quad (M_2)$$

$$\min \{F : (6) - (9), (11) - (13), (15), (16), (17), (18)\}, \quad (M_3)$$

where (M_0) is the basic model (MinFlops), and (M_1) – (M_3) gradually incorporate the three methods proposed in Sect. 3 into the basic model.

Table 1 presents the computational times of these four models on five small problems, while Fig. 3.1 shows their computational graphs. We also include in the table the number of multiplications required by the forward mode, the backward mode, and minimum Markowitz degree for comparison. By comparing the CPU times for

Table 1 Comparison of CPU times (in seconds) of solving model (M_0) – (M_3) for five small problems. Instances (a)–(c) are from [6]. .

Problem	Vertices			No. of Multiplications				Times			
	X	Y	Z	Forward	Backward	Markowitz	F^*	M_0	M_1	M_2	M_3
fig10.4	4	3	3	22	18	22	18	0.04	0.04	0.02	0.11
ex10.8	4	3	5	28	24	26	22	0.66	0.26	0.37	0.77
fig10.1	4	2	5	20	18	16	15	0.81	0.71	0.35	0.29
butterfly	4	4	8	48	48	48	48	t ^a	t	t	337.97
revbound	1	1	10	10	19	10	10	t	t	42.33	31.42

^a t indicates failure to find a provably optimal solution within 600 sec.

(M_0) and (M_1) , we see that the amount of time required by (M_1) is less than that of (M_0) , as a result of the reduction in the number of constraints. The reduction is crucial for large graphs in that a standard IP solver might not be able to solve even the root node LP relaxations because of the out-of-memory issue with (M_0) on large instances. Comparing the results of (M_1) and (M_2) , we see that “revbound” could not be solved within the time limit if modeled by (M_1) but can be solved with the help of the lower bound constraints; these results demonstrate that the lower bound con-

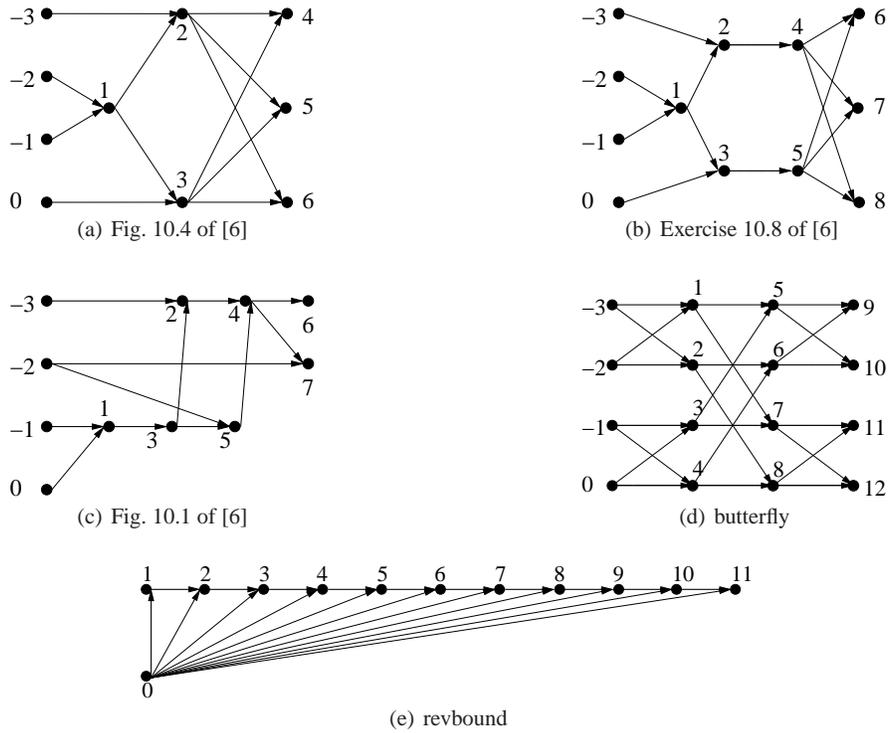


Fig. 2 Computational graphs of test problems in Table 1 and Table 2

straints we develop tighten the LP relaxations significantly. Observe that “butterfly” is a computational graph with lots of symmetry in its edge structure. Having or not having the symmetry-breaking constraint (18) makes a big difference on this graph. This clearly illustrates the benefits of adding the symmetry-breaking constraints to problems with many equivalent elimination sequences.

3.2 Scarcity

We tested (Scarcity) with the same set of instances as in Table 1. The results are presented in Table 2. In the column heads, R^* is the optimal value of (Scarcity), the maximum reduction in edge count, and $|E(t^*)|$ is the minimum number of edges that can be achieved with vertex elimination. We also list the set of vertices that need to be removed in order to achieve the scarcity.

Table 2 Results of solving (Scarcity).

Problem	$ X $	$ Y $	$ V $	$ E $	R^*	$E(r^*)$	Removed vertices	Time
fig10.4	4	3	10	12	1	11	2-3	0.05
ex10.8	4	3	12	14	3	11	2-4-3-5	5.29
fig10.1	4	2	11	12	5	7	2-3-5-1	2.92
butterfly	4	4	16	24	8	16	6-7-8-1-5-2-3-4	600 ^b
revbound	1	1	12	21	20	1	2-10-8-7-1-6-5-4-9-3	25.5

^b The corresponding IP was not solved to optimality within a time limit of 600 sec., and a feasible solution is returned instead.

4 Conclusion

We have developed integer programming models for two problems in AD: the vertex elimination problem for the optimal Jacobian accumulation, and the scarcity problem. These models allow us to use IP technology to solve these problems, so that one can evaluate the effectiveness of heuristics used in AD tools and find an optimal strategy for certain computational kernels. The IPs for these problems are nevertheless hard to solve. We have developed several techniques that strengthen the basic vertex elimination model. As demonstrated by Sect. 3, the IP approach to solve the two problems is promising.

Several directions remain for research. First, one might be able to derive alternative IP formulations for these problems with stronger LP relaxations, crucial to solving them with a standard IP solver. Also, it is possible to solve these problems more efficiently with specialized IP techniques, such as Benders's decomposition [3]. Second, as indicated by the computational results in Table 1 and Table 2, (Scarcity) is harder to optimize than (MinFlops), and it is desirable to find tight lower bounds and other computational enhancements techniques for the basic model. Third, we hope to create a benchmark with a wide variety of small to medium-sized instances with known optimal vertex elimination sequence and scarcity. Conceptually it is not hard to see that the integer programming approach can also be extended to edge-elimination, and how to computationally solve the corresponding IP models is both interesting and challenging.

Acknowledgements We sincerely thank Robert Luce for helpful discussion that led to an earlier IP formulation of the vertex elimination problem. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.

References

1. Xpress-Optimizer Reference Manual (2009). URL <http://fico.com/xpress>

2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton (2007)
3. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* **4**, 238–252 (1962). URL <http://dx.doi.org/10.1007/BF01386316>. DOI 10.1007/BF01386316
4. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. No. 19 in *Frontiers in Appl. Math.* SIAM, Philadelphia, PA (2000)
5. Griewank, A., Reese, S.: On the calculation of Jacobian matrices by the Markowitz rule. In: A. Griewank, G.F. Corliss (eds.) *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pp. 126–135. SIAM, Philadelphia, PA (1991)
6. Griewank, A., Walther, A.: *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd edn. No. 105 in *Other Titles in Applied Mathematics*. SIAM, Philadelphia, PA (2008). URL <http://www.ec-securehost.com/SIAM/OT105.html>
7. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. *J. ACM* **7**, 326–329 (1960). DOI <http://doi.acm.org/10.1145/321043.321046>. URL <http://doi.acm.org/10.1145/321043.321046>
8. Naumann, U.: Optimal Jacobian accumulation is NP-complete. *Math. Prog.* **112**, 427–441 (2006). DOI 10.1007/s10107-006-0042-z
9. Naumann, U., Hu, Y.: Optimal vertex elimination in single-expression-use graphs. *ACM Transactions on Mathematical Software* **35**(1), 1–20 (2008). DOI 10.1145/1377603.1377605
10. Rosenthal, R.E.: *GAMS – A User’s Guide* (2011)
11. Wolsey, L.A., Nemhauser, G.L.: *Integer and Combinatorial Optimization*. Wiley-Interscience (1999)

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (“Argonne”) under Contract DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.