

Transformation of Linearized Computational Graphs Driven by the Associativity of the Chain Rule

Von der Fakultät Mathematik, Informatik und Naturwissenschaften der RWTH Aachen
University zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Mathematiker
Viktor Mosenkis
aus Lwow, Ukraine

Berichter : Prof. Dr. Uwe Naumann, RWTH Aachen University
 Prof. Trond Steihaug, Ph.D., Bergen University

Tag der mündlichen Prüfung : 16.10.2017

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek verfügbar.

Contents

Contents	3
1 Introduction	7
2 Foundations	19
2.1 Problem description and state of the art in elimination methods	19
2.2 Generic branch and bound algorithm	25
2.3 Further definitions and notation	27
3 Properties of linearized directed acyclic graphs	31
3.1 Subgraph monotonicity	31
3.1.1 The proof	32
3.1.2 Consequences for the minimal edge count problem	38
3.1.3 Consequences for the optimal edge and vertex elimination problems	39
3.2 Commutativity of edge and vertex eliminations	41
3.2.1 Commutativity of two vertex eliminations	41
3.2.2 Commutativity of a vertex and edge elimination	42
3.2.3 Commutativity of two edge eliminations	45
3.2.4 Consequences for the minimal edge count problem	48
3.2.5 Consequences for the optimal edge and vertex elimination problems	49
3.3 Summary	53
4 Absorption-free l-DAGs	55
4.1 Properties of absorption-free l-DAGs	56
4.2 Minimal edge count problem	63
4.3 Optimal vertex and edge elimination problem	67
4.3.1 Vertex elimination solves the edge elimination problem on absorption-free l-DAGs	67
4.3.2 Optimality preserving eliminations and critical degree	84
4.3.3 Lower bounds	98
5 Minimal edge count problem on arbitrary l-DAGs	111
5.1 Branch and bound algorithm	111
5.2 Exploit commutativity of edge and vertex eliminations	113
5.3 Optimality preserving eliminations	116
5.3.1 Vertex elimination	116
5.3.2 Edge elimination	119
5.4 Lower bounds	122

6	Vertex and edge elimination problems on arbitrary l-DAGs	127
6.1	Branch and bound algorithm	128
6.2	Exploiting commutativity of eliminations	129
6.2.1	Exploiting commutativity of vertex eliminations	129
6.2.2	Exploiting commutativity of edge eliminations	133
6.3	Lower bounds	135
6.3.1	Simple bounds	135
6.3.2	Lower bounds based on reduction to absorption-free subgraphs	139
6.4	Numerical results	144
6.4.1	Vertex elimination	144
6.4.2	Edge elimination	147
6.4.3	Conclusion	150
6.5	Optimality preserving elimination	150
7	Outlook	157
7.1	Minimal edge count problem	157
7.2	Optimal Jacobian accumulation	158
7.2.1	Absorption-free l-DAGs	158
7.2.2	Arbitrary l-DAGs	160
7.3	Solving MEC problem with minimal number of operations	166
8	Conclusion	169
A	Appendix	171
A.1	Code	171
A.2	Running the branch and bound algorithm	171
A.3	EliAD-format	172
A.4	Output file	173
	Bibliography	175

Acknowledgement

I would first like to thank my thesis advisor Prof. Dr. Uwe Naumann. The door of his office was always open whenever I ran into a trouble, spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction.

I take this opportunity to express gratitude to all (former and current) members of LuFG Informatik 12: Software and Tools for Computational Engineering for friendship, support and valuable discussion during this long time.

I would also like to thank The Numerical Algorithm Group for giving me the opportunity to finish this work while being their employee. I am also grateful to my NAG colleague David Sayers for his valuable feedback on my English style.

Finally, I must express my very profound gratitude to my parents and to my partner, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Chapter 1

Introduction

Motivation The efficient computation of derivatives of mathematical functions which are implemented as computer programs has become more important in various branches of industry and academia. Derivatives are used in different ways. In some cases they are used directly. For example in financial applications derivatives, where they are also referred to as Greeks, are used to investigate the volatility of a given product. Derivatives are also requested by many optimization algorithms like steepest decent and (Quasi-) Newton method. While derivative information is highly desired, its efficient and reliable computation remains a big challenge. Writing derivative code by hand is a very time consuming and error prone job as the underlying code base is typically very large. Even if one succeeds in writing efficient derivative code by hand, one has to maintain two source codes, that of the original function and that for derivative computation. For these reasons more "automatic" ways of computing derivatives are required. In the past finite differences was a very popular approach to approximate derivatives. This approach fails on more and more real world problems for two reasons. The first one is that for many functions approximating tangents by secants does not provide satisfactory results. The second problem of using finite differences is that the computational costs for computing the full dense Jacobian grows linearly with the number of inputs of the function. So this method is too expensive (in terms of computational time) for functions with many inputs. Therefore Algorithmic (or Automatic) Differentiation (AD) becomes the method of choice for computing derivatives.

AD is a collection of techniques allowing computation of accurate derivative information of a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that is implemented as a computer program in some high level programming language like C/C++ or Fortran. AD couples rule-based differentiation of language intrinsics with accumulation of derivatives according to the chain rule. Exploitation of associativity of the chain rule leads to an exponential number of ways for combining local partial derivatives of the intrinsics. Using AD we automatically get accurate derivative information. How fast we get this information depends on the strategy chosen to combine the local partial derivatives. Unfortunately finding an optimal strategy to accumulate the Jacobian is an NP-hard problem [Nau08]. Thus all AD tools employ some sort of heuristic strategies. The most popular heuristics used by AD tools are forward and reverse modes [GW08]. In forward mode the products of the Jacobian matrix (or Jacobian)

$$F' = F'(x) = \begin{pmatrix} \frac{\partial F_1(x)}{\partial x_1} & \cdots & \frac{\partial F_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial F_m(x)}{\partial x_1} & \cdots & \frac{\partial F_m(x)}{\partial x_n} \end{pmatrix}$$

with a tangent $\dot{x}$ according to $\dot{y} = F' \dot{x}$ are computed, while reverse mode results in products of an adjoint $\bar{y}$ with Jacobian $\bar{x} = \bar{y}^T F'$. Both forward and reverse mode can be executed in vector modes computing

products not only with one tangent or adjoint but rather with a set of tangents or adjoints. With these two modes we can compute the derivative of the function F at costs that are independent of the number of outputs (forward mode) or of the number of inputs (reverse mode). The reverse mode is the method of choice in many numerical applications as the number of inputs often exceeds the number of outputs. Although these modes provide satisfactory results for most problems, choosing a better strategy can save a lot of computational time as demonstrated in [Nau99, Nau00]. As already mentioned, the problem of finding the best strategy is an NP-hard problem. So the gain one can get from finding an optimal solution for the whole program often does not justify the computational time invested in optimizing the derivative computation. Due to associativity of the chain rule, it is possible to preaccumulate derivative information for certain parts of the code and then use this information to propagate the derivatives in forward or reverse mode. Some of the tools like (OPENAD/F [UNF⁺08] and dco/c++ [LLN11]) use this fact to preaccumulate derivative information for basic blocks and statements. Sometimes it might be rewarding to put additional effort into optimizing this preaccumulation code. As basic blocks and statements do not contain control flow (loops and branches), the optimization can be done once during the compilation of the code and then reused multiple times during the code execution. For example let us discuss the computation of the Jacobian matrix of the function f from Listing 1.1. This function takes two n -dimensional vectors a and b as inputs and stores the result in two n -dimensional output vectors y and z . One possible way of computing the Jacobian of this function is presented in Listing 1.2. The Jacobian is accumulated by combining the derivatives of each statement in the order they occur in the program. This code is slightly optimized as we ensure that we do not recompute the partial derivatives of statements. Instead they are stored in `tmp1`, `tmp2` and `tmp3`. Still this derivative code can be optimized as shown in Listing 1.3. The idea behind the optimization is to precompute the derivative of `v3` with respect to `v1` in Line 12 rather than multiply the local partial derivatives in the same order they occur in the original code. This small tweak saves 3 multiplications and 1 addition inside the loops body and allows to compute the Jacobian approximately 10% faster compared to the implementation in Listing 1.2.

Listing 1.1: Example Jacobian computation: original function

```

1 void f(int n, const double *a, const double *b, double *y, double *z
    ) {
2     double v1, v2, v3;
3
4     for(int i = 0; i<n; i++) {
5         v1=a[i]*b[i];
6         v2=sin(v1);
7         v3=v2/v1;
8         y[i]=v3/5.0;
9         z[i]=v3*v3;
10    }
11 };

```

Listing 1.2: Example Jacobian computation: derivative code without optimization

```

1 void df(int n, const double *a, const double *b, double **jac) {
2     double v1, v2, v3, tv1, tv2, tv3, tv4, tmp1, tmp2, tmp3;
3
4     for(int i = 0; i<n; i++) {
5         v1=a[i]*b[i];
6         v2=sin(v1);
7         v3=v2/v1;
8         tmp1 = cos(v1);

```

```

9      tmp2 = 1/v1;
10     tmp3 = -v2/(v1*v1)
11
12     tv1 = tmp1*a[i];
13     tv2 = tmp2*tv1 + tmp3*a[i];
14     tv3 = tmp1*b[i];
15     tv4 = tmp2*tv3 + tmp3*b[i];
16     // dy/da
17     jac[i][i]      = tv2/5.0;
18     // dz/da
19     jac[n+i][i]    = 2*v3*tv2;
20     // dy/db
21     jac[i][n+i]    = tv4/5.0;
22     // dz/db
23     jac[n+i][n+i]  = 2*v3*tv4;
24 }
25 };

```

Listing 1.3: Example Jacobian computation: derivative code with optimization

```

1 void df_opt(int n, const double *a, const double *b, double **jac) {
2     double v1, v2, v3, tv1, tv2, tv3, tmp1, tmp2, tmp3, tmp;
3
4     for(int i = 0; i<2; i++) {
5         v1=a[i]*b[i];
6         v2=sin(v1);
7         v3=v2/v1;
8         tmp1 = cos(v1);
9         tmp2 = 1/v1;
10        tmp3 = -v2/(v1*v1)
11
12        tmp = tmp1*tmp2 + tmp3;
13        tv1= b[i]*tmp;
14        tv2= a[i]*tmp;
15        // dy/da
16        jac[i][i]      = tv1/5.0;
17        // dz/da
18        jac[n+i][i]    = 2*v3*tv1;
19        // dy/db
20        jac[i][n+i]    = tv2/5.0;
21        // dz/db
22        jac[n+i][n+i]  = 2*v3*tv2;
23    }
24 };

```

We do not always need to compute the full Jacobian matrix, sometimes we are interested in computing several Jacobian-vector or vector-Jacobian products. This is for example the case when solving some optimization problem with matrix-free Newton-Krylov methods. Trying to optimize the derivative code by preaccumulating the Jacobian (of basic blocks) can be counter productive in such situations compared to simple propagation of tangents or adjoints. For example the function f in Listing 1.4 takes two inputs a

$\mathbf{b}$ and stores the result in a 3 dimensional vector $\mathbf{y}$. Application of the forward mode of AD ($\dot{\mathbf{y}} = \mathbf{F}'\dot{\mathbf{x}}$) to each line of function $\mathbf{f}$ in Listing 1.4 yields function $\mathbf{f_t}$ as shown in Listing 1.5. According to forward mode of AD $\mathbf{f_t}$ computes Jacobian-vector products of the function $\mathbf{f}$. Computing one tangent with the help of function $\mathbf{f_t}$ involves 6 multiplications and 1 addition. If we try to optimize this computation by precomputing the derivative of the function $\mathbf{f}$ before the propagation of tangents, we get a code as in Listing 1.6. Even if we ignore the cost of operations used to precompute the Jacobian ($\mathbf{Jac}$), computation of one Jacobian-vector or vector-Jacobian with this code requires 7 multiplications and 3 additions.

Listing 1.4: Example Jacobian-vector product: original function

```
1 void f(int n, const double a, const double b, double *y) {
2     double v = a*b;
3     y[1] = exp(v);
4     y[2] = sin(v);
5     y[3] = cos(v);
6 };
```

Listing 1.5: Example Jacobian-vector product: forward mode

```
1 void f_t(int n, const double a, const double ta, const double b,
2         const double tb, double *ty) {
3     double v = a*b;
4     double tv = a*tb+b*ta;
5     ty[1] = exp(v)*tv;
6     ty[2] = sin(v)*tv;
7     ty[3] = cos(v)*tv;
8 };
```

Listing 1.6: Example Jacobian-vector product: preaccumulated Jacobian

```
1 void f_t(int n, const double a, const double ta, const double b,
2         const double tb, double *ty) {
3     double Jac[3][3];
4     double v = a*b;
5     Jac[0][0] = exp(v)*b;
6     Jac[1][0] = cos(v)*b;
7     Jac[2][0] = -sin(v)*b;
8
9     Jac[0][1] = exp(v)*a;
10    Jac[1][1] = cos(v)*a;
11    Jac[2][1] = -sin(v)*a;
12
13    for (int i=0; i<3; i++)
14        ty[i] = Jac[i][0]*ta + Jac[i][1]*tb;
15 };
```

So depending on what derivative information is required, we have to address different objectives while optimizing the preaccumulation code. If we want to preaccumulate the full Jacobian matrix, we are interested in generating a code that minimizes the number of operations needed to accumulate the Jacobian. This problem is referred to as the *optimal Jacobian accumulation* problem. If Jacobian-vector or vector-Jacobian multiplications are needed, we have to solve the minimal edge count problem. The

minimal edge count problem tries to find a graph representation of the Jacobian with minimal number of edges (see below for a formal definition of this problem). Such minimal graph representation allows us to minimize the number of operations needed to propagate tangents or adjoints and reduces the costs for storing the preaccumulated Jacobian.

This thesis deals with solving the OPTIMAL JACOBIAN ACCUMULATION and MINIMAL EDGE COUNT problems. The approach taken in this thesis is to consider both problems as elimination problems on graphs. For this purpose the parts of the code that should be preaccumulated are represented by a directed acyclic graph (DAG) with elemental partial derivatives as edge label, yielding linearized DAGs (l-DAGs). It is assumed that the code can be represented by a single DAG as for example for basic blocks.

Problem description First of all we need a formal definition of l-DAGs.

Definition 1.1. We consider implementations of multivariate vector functions $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ as computer programs mapping a vector of inputs $\mathbf{x} \in \mathbb{R}^n$ onto a vector of outputs $\mathbf{y} \in \mathbb{R}^m$ as $\mathbf{y} = F(\mathbf{x})$. The given implementation of F is assumed to decompose into a single assignment code (SAC) with total of $p + m$ assignments of the values of scalar intrinsic (or elemental) functions φ_j to intermediate variables v_j as follows:

$$\begin{aligned} \text{for } j = 1, \dots, p + m \\ v_j = \varphi_j(v_i)_{i \prec j} \end{aligned}$$

where following the notation in [GW08] $v_{k-n} = x_k$ for $k = 1, \dots, n$, $v_{l+p} = y_l$ for $l = 1, \dots, m$ and $i \prec j$ denotes a direct dependence of v_j on v_i . A directed acyclic graph (DAG) $G = G(F) = (V, E)$ with integer vertices $V = V(G) = \{1 - n, \dots, 0, 1, \dots, p + m\}$ and edges $E = E(G) = \{(i, j) \mid i, j \in V, i \prec j\}$ is induced.

The $\varphi_j, j = 1, \dots, p + m$, are assumed to be continuously differentiable in a neighborhood of their respective arguments. The DAG G is linearized by attaching the values of local partial derivatives

$$c_{jk} \equiv \frac{\partial \varphi_j}{\partial v_k}(v_i)_{i \prec j}$$

to all edges $(k, j) \in E(G)$ yielding the linearized DAG or l-DAG.

For example the function $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ with the mapping

$$f(x_1, x_2) = \begin{pmatrix} y_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} (\sin(x_1 \cdot x_2) + x_1 \cdot x_2)^2 \\ \exp(\sin(x_1 \cdot x_2) + x_1 \cdot x_2) \end{pmatrix}$$

can be implemented by the SAC

$$v_1 = v_{-1} \cdot v_0; v_2 = \sin(v_1); v_3 = v_1 + v_2; v_4 = v_3^2; v_5 = \exp(v_3) \quad ,$$

and yields the l-DAG shown in Figure 1.1(a) (Note that $x_1 = v_{-1}$, $x_2 = v_0$, $y_1 = v_4$ and $y_2 = v_5$). The local partial derivatives

$$\begin{aligned} c_{1-1} = v_0; c_{10} = v_{-1}; c_{21} = \cos(v_1); \\ c_{31} = c_{32} = 1; c_{43} = 2 \cdot v_3; c_{53} = \exp(v_3) = v_5 \end{aligned}$$

are attached to the respective edges. A graph-based interpretation of the chain rule of differential calculus first presented in [Bau74] yields the following method for accumulating all entries of the Jacobian matrix

$A = F'(\mathbf{x}) \in \mathbb{R}^{m \times n}$ of F at some point $\mathbf{x} \in \mathbb{R}^n$. Let $(i, \dots, j)$ denote a path connecting a vertex i with a vertex j in G and let $A = (a_{j,i})_{i=-n+1, \dots, 0}^{j=p+1, \dots, p+m}$. Then

$$a_{j,i} = \sum_{(i, \dots, j)} \prod_{(k,l) \in (i, \dots, j)} c_{lk} \quad (1.1)$$

for all sources $i \in \{0, \dots, -n+1\}$ and sinks $j \in \{p+1, \dots, p+m\}$.

The chain rule of differential calculus is associative. This fact allows numerous ways of combining local partial derivatives. Elimination methods on l-DAGs build an important subset. These methods are based on local application of Equation (1.1) to parts of the l-DAG. For example *elimination of a vertex* $v \in V(G)$ is defined through the application of Equation (1.1) to the subgraph induced by v and all its adjacent vertices (compare Definition 2.4 and Example 2.2). Another elimination method is *elimination of an edge*. This method is a generalization of vertex elimination. There exist two ways of eliminating an edge: front and back elimination. *Front elimination of an edge* $(i, j) \in E(G)$ is defined through the application of Equation (1.1) to the subgraph induced by the vertices i, j and all successors of j (compare Definition 2.2 and Example 2.1). *Back elimination* of (i, j) is defined analogously on the subgraph induced by the vertices i, j and all predecessors of i (see Definition 2.3 and Example 2.1). Both vertex and edge elimination methods can also be used to compute (generalized) derivative information of other computations, as long as the underlying differential calculus is associative. For example to propagate subgradients of McCormick relaxations [CMB⁺11, BMMN11, CN11, BMN12] or uncertainties using the method of moments as shown in [BN10, BN11].

Normally people associate AD with forward and reverse (vector) modes. Both forward and reverse (vector) modes can be represented through vertex elimination on l-DAGs. Conceptually, the computation of the product of the Jacobian with some matrix S (e.g. set of tangents) can be performed by augmenting the l-DAG with q new sources $1 - q - n, \dots, -n$ and edges (i, j) for $i = 1 - q - n, \dots, -n$ and $j = 1 - n, \dots, 0$, such that $c_{ji} = s_{n+j, i+q+n}$, where $S = (s_{l,k})_{k=1, \dots, q}^{l=1, \dots, n}$, see Figure 1.1(b) for an example. Application of Equation (1.1) to this augmented l-DAG yields the matrix product $A \cdot S$. To compute this product in forward (vector) mode we need to eliminate all vertices from $-n+1$ to p in increasing order. Analogously, application of Equation (1.1) to the l-DAG augmented with q new sinks $p+m+1, \dots, p+m+q$ and edges (i, j) for $i = p+1, \dots, p+m$ and $j = p+m+1, \dots, p+m+q$ such that $c_{ji} = w_{i-p, j-p-m}$ where $(w_{l,k})_{k=1, \dots, q}^{l=1, \dots, m} = W \in \mathbb{R}^{m \times q}$ allows us to compute the product of the matrix $W \in \mathbb{R}^{m \times q}$ (e.g. set of adjoints) with the Jacobian matrix A . To compute this product in reverse (vector) mode we have to eliminate all vertices from $p+m$ to 1 in decreasing order. The only operations used by elimination methods are multiplication and addition. Moreover additions are only performed subsequent to a previous multiplication, if we get two parallel edges (edges having same source and sink). Modern CPU architectures such as Intel Haswell can perform *fused multiply-add* (fma) operations in the same time as single scalar multiplication operations [JP98]. Hence, on such architectures we can compute all needed additions for "free". Therefore in this thesis we assume number of fma's as a measure for the costs. In established literature different measure for complexity are used, in [Nau00, Nau04] we count the number of fma's in [CHMU12] and [GW08] (for large l-DAGs) the number of multiplications. As additions are only performed subsequent to a previous multiplication these two measures for complexity are the same.

The number of arithmetic operations performed by forward and reverse modes grows with the number of edges in the original l-DAG $G = (V, E)$, yielding a computational cost of order $O(|E(G)|)$. So we are looking for ways to reduce the number of edges in the l-DAG while preserving the same linear mapping induced by the original Jacobian. For example the SAC of a function $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ with $p = 3$ is augmented with a $K_{2,2}$ biclique to represent the matrix product $A \cdot S$ where $A, S \in \mathbb{R}^{2 \times 2}$. The cost induced by the setup in Figure 1.1(b) is $2 \cdot |E| = 14$ fma's. We can reduce the size of G in Figure 1.1(b) through back elimination of the edge $(2, 3)$ followed by back elimination of $(1, 3)$ yielding a graph G' as

on Figure 1.1(c) with induced costs of $2 \cdot |E(G')| = 8$ fma's. The total cost adds up to $3 + 8 = 11$ fma's where 3 is the cost of preaccumulation step.

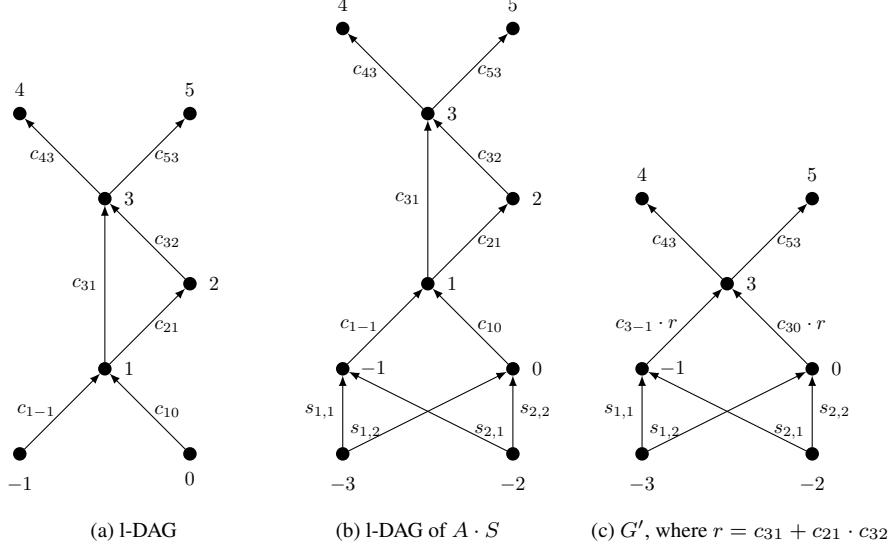


Figure 1.1: Example

The *minimum edge count problem by edge elimination* (MEC) (or simply minimum edge count problem) is to determine a sequence of front and back edge eliminations such that the resulting l-DAG contains the lowest possible number of edges. A simpler version of this problem is *minimum edge count problem by vertex elimination* (MECV) where we need to determine a sequence of vertex eliminations such that resulting l-DAG contains lowest possible number of edges.

The computation of the whole Jacobian matrix can be achieved through transformation of the original l-DAG into a bipartite graph by eliminating all intermediate vertices (vertices from 1, to p) through edge or vertex eliminations. According to Equation (1.1), the edge labels of the resulting bipartite graph correspond to the non-zero entries of the Jacobian. The computational costs can differ depending on the order of eliminations performed. For example the Jacobian induced by the l-DAG in Figure 1.1(a) can be obtained by eliminating the vertex 1 (at cost of 4 fma's) followed by elimination of vertices 2 (2 fma's) and 3 (4 fma's) yielding a total cost of 10 fma's. A better strategy to compute this Jacobian is to eliminate the vertex 2 (1 fma) first followed by elimination of vertices 1 (2 fma's) and 3 (4 fma's). This strategy yields a total cost of 7.

The problem of transforming the l-DAG into a bipartite graph by front and/or back elimination of its edges with minimal overall cost is referred to as the *optimal Jacobian accumulation problem by edge elimination* or *optimal edge elimination* (EE) problem. Similarly, the problem of transforming the l-DAG into a bipartite graph by elimination of its vertices with minimal overall costs is referred to as the *optimal Jacobian accumulation problem by vertex elimination* or *optimal vertex elimination* (VE) problem. As mentioned above in this thesis we take the number of performed fma's as the measure of complexity. Hence both VE and EE problems aim to minimize the number of fma's.

Both forward and reverse modes can be used to obtain the full Jacobian. In forward mode this can be achieved by setting the $\dot{X} = I_n$, where I_n denotes the identity of $\mathbb{R}^n$. In reverse mode the Jacobian can be computed by setting $\bar{Y} = I_m$, where I_m denotes the identity of $\mathbb{R}^m$. As we just learned, the minimum edge count problem aims to minimize the number of operations needed to propagate the derivative infor-

mation in forward and reverse mode. So one can be tempted to believe that the solution of the MEC (MECV) problem can be used as solution of the EE (VE) problem. Unfortunately this is not true in general case. First of all the minimum edge count problem does not require that the transformation of the l-DAG into one with minimal number of edges is optimal in the sense of needed number of fma's. Let us for example assume that elimination of all intermediate vertices (vertices from 1 to p) solves the MEC problem. So the solution of the MEC problem will tell us that we have transform the l-DAG into a bipartite graph. Still this fact does not help us to solve the EE or VE problem at all, as we still need to find a sequence of edge or vertex eliminations transforming the l-DAG into a bipartite graph with minimal cost.

Objective and approach This thesis deals with solving the optimal Jacobian accumulation and minimum edge count problems by edge and vertex eliminations. Neither vertex nor edge elimination exploit the fact that some of the edge labels are not unique. Thus we assume that all local partial derivatives are unique. Similar assumptions were also made in other publication [GR91, GN03, Nau02, Nau04]. The assignments within a SAC can be executed in different orders without changing the implemented function. Thus, in general, there exists more than one version of SAC for a given function implementation. Nevertheless the l-DAGs induced by the different versions of SAC are isomorphic. Vertex and edge elimination are not influenced by the underlying SAC but only the the structure of the l-DAG. Therefore we will not discuss different versions of SAC in this thesis. For the same reason throughout this thesis we treat EE, VE, MEC and MECV problems as pure combinatorial optimization problems on l-DAGs and not discuss the programs inducing the l-DAGs or explicitly compute the corresponding Jacobians.

Both optimal Jacobian accumulation and the minimum edge count problems by vertex and edge eliminations are conjectured to be NP-hard. Therefore in practice the solution of these problems is approximated by greedy-like heuristics [AGN03, Nau00] or computed by branch and bound based methods as in [CHMU12] if the exact solution is of interest. The potential of edge elimination is almost never used in practice due to the dramatically bigger search space compared to vertex elimination. The aim of this thesis is to support further development of algorithms for solving the VE, EE, MECV and MEC problems. Especially we want to show that the potential of edge elimination should not be neglected. These goals are achieved through

- giving a better understanding of the problems structure,
- providing building blocks for development of new algorithms such as
 - new approach for computing lower bounds
 - optimality preserving eliminations
- showing how to make use of edge eliminations in practice.

The flow through the main chapters of this thesis and the connections between the different parts and section are shown in Figure 1.2. The picture describes only chapters 3 to 6, not touching the introduction, foundation, outlook and conclusion.

To provide a better understanding about the structure of the (VE, EE, MEC and MECV) problems several properties of the l-DAG and the eliminations methods are proved in Chapter 3. One important property is "commutativity" of elimination methods. It is obvious that in some cases the order of edge (vertex) eliminations does not change the resulting l-DAG. In Section 3.2 this observation is formalized and proved for both elimination methods. It turns out the order of vertex eliminations does not change the resulting l-DAG. This property is referred to as "commutativity" of vertex eliminations. It allows to reduce the search space for the VE and MECV problems significantly. A surprising outcome of this result is, that the search space of the VE problem is larger than that of the MECV problem. So far it was assumed that the MECV problem is at least as hard as the VE problem [CHMU12]. This property is also extensively used

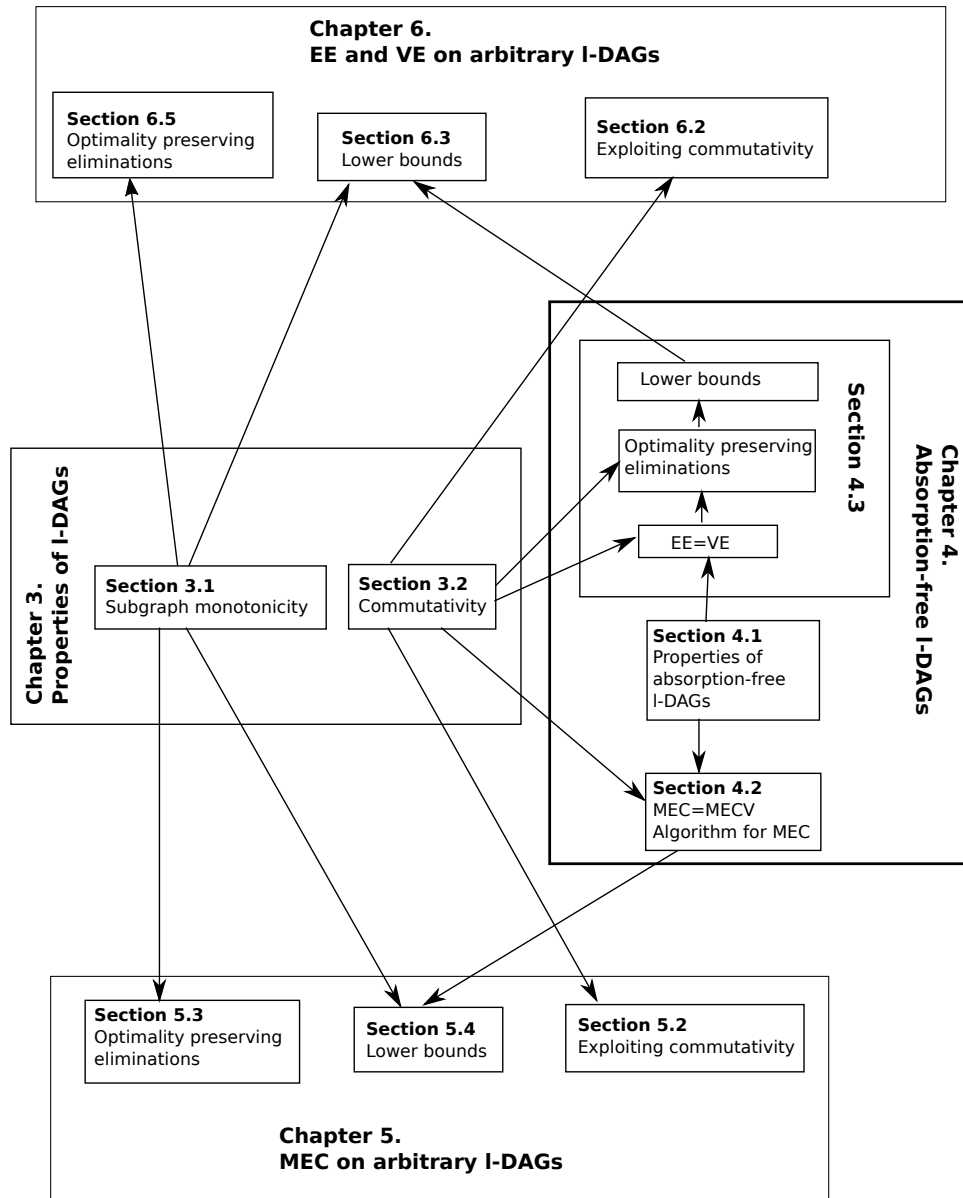


Figure 1.2: Connections and dependencies between chapters and sections

in various proofs in Chapter 4. Unfortunately the order of edge eliminations can change the resulting l-DAG in some cases. Therefore this property is used only to reduce the search space of the EE and MEC problems within a branch and bound algorithm. Another important property of l-DAGs is referred to as "subgraph monotonicity" (see Section 3.1). Generally speaking "subgraph monotonicity" means that if an l-DAG G' is a subgraph (see Definition 3.1) of an l-DAG G then G' is still a subgraph of G after the application of the same edge (vertex) elimination to both l-DAGs. This property is used to prove various optimality preserving eliminations¹ but most importantly it allows us to reuse results or solutions from a subgraph of the l-DAG either as part of the optimal solution or as a lower bound² for it. Thanks to "subgraph monotonicity" first lower bounds for the MEC problem could be proved.

To develop useful building blocks for future algorithms we first need to understand what kind of algorithms are most likely to be used. As already mentioned both optimal Jacobian accumulation and the minimum edge count problems by vertex and edge eliminations are conjectured to be NP-hard. These kind of problems are typically solved either by branch and bound based methods or the optimal solution is approximated by greedy-like algorithms. For efficient application of such algorithms, good lower bounds are essential. In the context of branch and bound methods better lower bounds allow earlier cuts to be performed. Good lower bounds can also help to understand the quality of the computed solution. This is for example useful for analysing the results of greedy heuristics. A big part of this thesis deals with developing a new approach for computing lower bounds. The approach taken in this thesis to compute the lower bounds is based on reduction to absorption-free graphs exploiting "subgraph monotonicity". Absorption-free l-DAGs are directed graphs where two vertices are connected by at most one directed path. We compute the lower bound on absorption-free subgraph of the original l-DAG G and then use the computed result as the lower bound for G . This approach can be used to compute lower bounds for all discussed problems. Therefore a big part of research during the course of this thesis was spent on investigating this subclass of graphs. The most interesting result for absorption-free graphs is the proof that $EE = VE$. An analogous result for the minimum edge count problem ($MEC = MECV$) is also provided. While a linear time algorithm for solving the MEC problem on absorption-free graphs could be developed, the EE problem on absorption-free graphs turned out to be more challenging. So only a solution to the EE problem for absorption-free graphs with maximal path length 3 is proved. For l-DAGs with greater path length, new algorithms for lower bounds and optimality preserving eliminations are developed. The superiority of the new lower bounds over the existing one [NH08] (sum of Markowitz degrees, compare Lemma 4.24) is proved.

Another useful technique for branch and bound and greedy like algorithms are optimality preserving eliminations [GW08, NU05]. They allow the search space for the branch and bound algorithm to be reduced and can also be used to improve the greedy heuristics. In this thesis several optimality preserving elimination are proved. The optimality preserving eliminations for edge eliminations are generalized versions of the results for vertex eliminations. Thus mixing of edge and vertex elimination in algorithms is suggested to improve the results for approximative algorithm and to reduce the corresponding search space if the exact solution is required, demonstrating that edge eliminations should not be neglected in praxis.

Related work The problems discussed in this thesis are only a small part of the AD research field. The most recent advances and activities in this field can be found in the proceeding [CFG⁺02, BCH⁺06, BBH⁺08, FHP⁺12]. The *optimal Jacobian accumulation* problem is the task of finding a way to compute F' using minimal number of fma's. This problem was proven to be NP-hard in [Nau08]. The proof exploits potential algebraic dependencies among the local partial derivatives. Prior to this publication all

¹Vertex or edge eliminations that can be safely performed without losing the optimality

²A lower bound for the EE (VE) problem indicates how many fma's are at least needed to transform the l-DAG into a bipartite graph through edge (vertex) eliminations, while a lower bound for the MEC (MECV) problem indicates the minimal number of edges contained in the solution of the MEC (MECV) problems

previous work on this subject has always made the assumption that the local partial derivatives are algebraically independent [GR91, GN03, Nau02, Nau04]. The optimal Jacobian accumulation problem under the assumption that all local partial derivatives are algebraically independent is referred to as *structural optimal Jacobian accumulation problem* (SOJA). The SOJA problem is conjectured to be NP-hard but the formal proof is still outstanding. In most publications a simplified version of the SOJA problem is examined where the accumulation of the Jacobian is considered as a transformation of the l-DAG with elimination methods, such as vertex and edge eliminations. Historically, the first developed elimination method was vertex elimination [GR91]. Originally vertex elimination was used in to calculate the fill-in generated by LU/Cholesky decomposition in a sparse matrix [Ros70, Ros72, RT78]. Vertex elimination was the only known elimination method until in [Nau99] a more generalized technique, edge elimination was introduced. In the same publication a counter example was presented showing that $VE \neq EE$ implying that the SOJA problem cannot be solved by vertex elimination. In [Nau04] it is shown that the solution of the EE problem does not solve the SOJA problem in general. In the same paper a new elimination technique *face elimination* together with the corresponding optimal face elimination (FE) problem was introduced. The big difference to the vertex and edge elimination is, that face elimination works on the dual graph. To compute the Jacobian one has to transform the dual graph into a tripartite graph. For more details please consult [Nau04]. FE is the most general of the three (FE, EE, VE) problems. However, it remains unclear whether $FE = SOJA$. For a special sub-class of graphs, the absorption-free l-DAGs (see Definition 4.1) Griewank proved that $SOJA = FE$ in [GW08]. VE and EE problems are conjectured to be NP-hard thus several publications deal with approximating the optimal solution by greedy-like heuristics based on Markowitz degree as in [Nau00] or by simulating annealing as in [Nau02, NG01]. An integer programming based algorithm for solving the VE problem was presented in [CHMU12]. Discussion on optimization of derivative code by vertex and edge elimination within a source transformation tools OPENAD/F and EliAD can be found in [UN12, TFPR02, TBPF05].

The *minimum edge count* (MEC) problem is a special case of the scarcity exploitation problem, discussed in [GV04, LU08]. The notion of scarcity exploits the fact that the Jacobian may have less degrees of freedom than the number of its entries. In [LU08] an automatic approach for detecting and using scarcity in computation graphs was introduced. Similar to the MEC problem the goal is to find a graph representation of the Jacobian with minimal number of edges. The only difference is that in addition to elimination techniques reroutings and normalizations are allowed. MEC problem is conjectured to be computationally intractable. Supporting evidence is provided by corresponding proofs for similar problems from the solution of sparse linear systems [Gil80, RT78]. A formal proof is still outstanding. Apart from the previously cited work there are not many publications on this problem. I am aware of only four further publications dealing with the MEC problem. In [LU08] various greedy heuristics to approximate the solution of the MEC problem are evaluated, while in [CHMU12] an integer programming formulation of the MECV problem is derived. Griewank and Walther devoted a section in their textbook [GW08] to this problem. Practical application of the MEC problem as part of the OPENAD/F is described in [UN12]. Analogously to the SOJA problem vertex elimination does not solve the MEC problem in general as shown in [MN12]. No attempts have been made to apply face elimination for solving the minimum edge count problem so far.

Another approach aiming to minimize the computational effort needed to accumulate the Jacobian, is based on exploiting sparsity of the Jacobian matrix. It is broadly discussed in AD community but not covered in this thesis. The idea behind this approach is to identify those columns (rows) of the Jacobian matrix that can be computed within one tangent (adjoint) pass, as their non-zero entries do not intersect. This problem is covered in various publications such as [HS05b, HS05a, GMP05, Var11, VN09]. Sparsity is a pure structural property and is implicitly exploited by all elimination methods discussed in this thesis, as edges in the l-DAG identify dependencies between underlying elemental functions. There are also approaches aiming reduce the number of tangent (adjoint) passes needed to accumulate the Jacobian, that are not based on pure structural properties of the l-DAG. For example exploiting the numerical rank of

the Jacobian [AKHL⁺08] or using the knowledge about its constant entries [VN09, Var11].

Structure of the thesis This thesis is organized as follows. In Chapter 2 the fundamentals of the EE, VE, MEC and MECV problems as well as definitions and notations used are shown. Afterwards the properties of l-DAGs, "subgraph monotonicity" and "commutativity of edge and vertex eliminations", are proved in Chapter 3. Chapter 4 is devoted to the EE, VE, MEC and MECV problems on absorption-free graphs. This is the most important chapter of the thesis. In Chapter 5 the results for the MEC and MECV problems on arbitrary l-DAG are presented. This is followed by discussions of the EE and VE problems on arbitrary l-DAGs in Chapter 6. An outlook is given in Chapter 7. The thesis is closed with conclusions in Chapter 8.

Chapter 2

Foundations

Following the motivation and brief informal definitions of the elimination techniques in the previous chapter, this chapter establishes the theoretical basis of this thesis. For that reason this chapter contains the definitions of the l-DAGs and the elimination methods as well as rigorous formulations of the examined problems. We also present the proofs that vertex elimination does not solve the EE problem (proved by Naumann in [Nau04]) and the MEC problem ([MN12]). For an expert reader in the field of AD the first section (Section 2.1) will probably not contain any new information. Nevertheless, this section introduces the notations used in connection with elimination methods, and therefore may be valuable to the reader. In Section 2.2 a generic branch and bound algorithm (Algorithm 1) is presented. The algorithm can be specialized to solve all four problems discussed in this thesis. It is used to demonstrate the practical application of the results developed in later chapters. Further basic notation which is not directly connected to elimination methods are introduced in the last section of this chapter. In this thesis we deal only with structural problems and will not touch the general OJA problem. Therefore throughout this thesis we will refer to structural optimal Jacobian accumulation problem as optimal Jacobian accumulation problem. Analogously, we also will write OJA instead of SOJA.

2.1 Problem description and state of the art in elimination methods

In this section we provide formal definitions of the problems that are the subject of research in this thesis and introduce the corresponding notations. We also discuss how vertex and edge elimination relate to each other. A rigorous definition of the l-DAG was presented in the previous chapter (see Definition 1.1) thus we omit redefining it here. To increase the readability, throughout this thesis we address the vertices of the l-DAG as follows

- for $i = 1 - n, \dots, 0$ as x_{i+n} (corresponding to the inputs of the function),
- for $i = 1, \dots, p$ as v_i and
- for $i = p + 1, \dots, p + m$ as y_{i-p} (corresponding to the outputs of the function).

In figures we mark inputs/outputs of the function with upward/downward facing triangles. Figure 2.1(a) demonstrates the l-DAG from Figure 1.1(a) with the new notation. Moreover we will omit drawing the edge labels on illustrations as we deal with a pure graph problem and leave the connections to the induced computation of derivatives aside.

Further following assumptions to the l-DAGs are made in this thesis:

- the outputs of the function do not have any successors in the l-DAG,

- the vertices v_i for $i = 1, \dots, p$ always have predecessors and successors.

These assumptions are not hard restrictions. If the output of the function has a successor, it means that it is used in the computation of some other output. So we could easily rewrite the program to match the first assumption by copying the result in to an auxiliary variable and using it inside the computation instead of the output itself. With the help of this assumption we know, that we do not have to eliminate any of the outputs to compute the Jacobian. This fact helps makes it easier to formulate claims and algorithms. The second assumption also does not restricts the code in any way, as vertex v_i without successors and/or predecessors can be safely eliminated as described in [GW08]. So existence of such vertices does not harden the problems discussed here.

Before we can define edge and vertex eliminations we need to introduce the notation for the neighbourhood of a vertex (edge).

Definition 2.1. Consider an l-DAG $G = (V, E)$, $v \in V(G)$ and $(i, j) \in E(G)$. We set

$$\begin{aligned} \text{src}(i, j) &= i & \text{tgt}(i, j) &= j, \\ S_G(v) &= \{\text{tgt}((v, i)) \mid (v, i) \in E(G)\}, & P_G(v) &= \{\text{src}((i, v)) \mid (i, v) \in E(G)\} \end{aligned}$$

Now we can define the front elimination of an edge and the corresponding costs.

Definition 2.2. Consider an l-DAG G . The edge $(i, j) \in E(G)$ is front eliminatable on G , if $S_G(j) \neq \emptyset$. The front elimination of (i, j) is performed by introducing new edges (i, j') for all $j' \in S_G(j)$ and $j' \notin S_G(i)$. The new edge labels are set to $c_{j'i} = c_{ji}c_{j'j}$. For all $(i, j') \in E(G)$ the existing labels are updated according to $c_{j'i} = c_{j'i} + c_{ji}c_{j'j}$. The front elimination of edge (i, j) on l-DAG G is denoted by $G - (i, j)^+$.

We call front edge elimination of (i, j) a pure edge elimination if the vertex j is not removed. This means that $V(G - (i, j)^+) = V(G)$.

The cost of front elimination of an edge (i, j) is equal to the number of fma's performed during this elimination. This number is equal to the number of the successors of j . Hence the cost of front elimination of (i, j) is equal to $|S_G(j)|$ and we denote it with

$$\sigma_G((i, j)^+) = |S_G(j)| \quad ,$$

where $|\cdot|$ denotes the cardinality (number of elements) of the set.

Back elimination of edge is defined in a similar way.

Definition 2.3. Consider an l-DAG G . The edge $(i, j) \in E(G)$ is back eliminatable on G , if $P_G(i) \neq \emptyset$. The back elimination of $(i, j) \in E(G)$ is performed by introducing new edges (i', j) for all $i' \in P_G(i)$ and $i' \notin P_G(j)$. The new edge labels are set to $c_{ji'} = c_{ji}c_{i'i}$. For all $(i', j) \in E(G)$ the existing labels are updated according to $c_{ji'} = c_{ji'} + c_{ji}c_{i'i}$. The back elimination of edge (i, j) on l-DAG G is denoted by $G - (i, j)^-$.

We call the back elimination of (i, j) a pure edge elimination if the vertex i is not removed. This means that $V(G - (i, j)^-) = V(G)$.

The cost of the back elimination of the edge (i, j) is equal to the number of fma's that has to be performed during this elimination. This number is equal to the number of the predecessors of i . Hence the cost of front elimination of (i, j) is equal to $|P_G(i)|$ and we denote it with

$$\sigma_G((i, j)^-) = |P_G(i)| \quad ,$$

where $|\cdot|$ denotes the cardinality (number of elements) of the set.

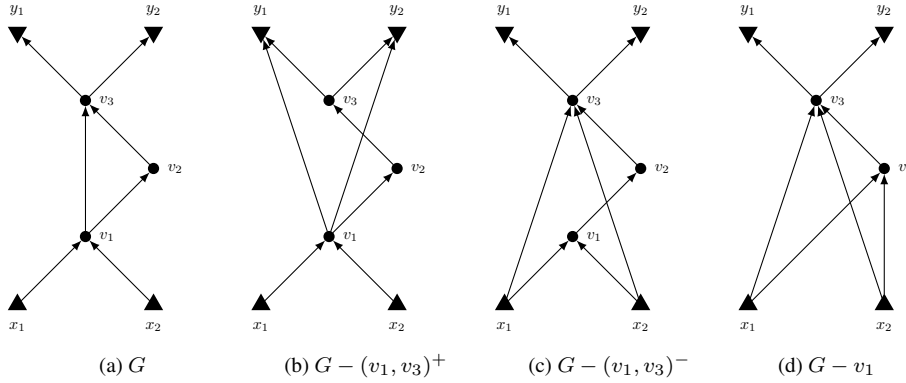


Figure 2.1: Example for vertex and edge eliminations

Let us discuss the front and back edge elimination on the following example.

Example 2.1. Consider an l-DAG G as in Figure 2.1(a). The result of front elimination of the edge (v_1, v_3) is shown in Figure 2.1(b), while the result of the back elimination of the same edge is presented in Figure 2.1(c).

Vertex elimination is defined as a sequence of edge eliminations.

Definition 2.4. Consider an l-DAG G . A vertex $v \in V(G)$ is *eliminatable* on G , if $P_G(v) \neq \emptyset$ and $S_G(v) \neq \emptyset$. A vertex v is *eliminated* from G by front elimination of all its in-edges (edges connecting v with its predecessors) or by back elimination of all its out-edges (edges connecting v with its successors).

The elimination of vertex v from l-DAG G is denoted by $G - v$.

We denote the cost of the elimination of vertex v in l-DAG G with $\sigma_G(v)$. The costs of a vertex elimination are equal to the number of fma's performed during this elimination. Hence

$$\sigma_G(v) = |P_G(v)| \cdot |S_G(v)| \quad .$$

Next we discuss vertex elimination on the following example.

Example 2.2. Consider an l-DAG G as in Figure 2.1(a). The result of eliminating vertex v_1 from G is shown in Figure 2.1(d).

For rigorous definition of the VE and EE and MEC problems we need to set the concept of elimination sequences.

Definition 2.5. Consider an l-DAG G . Subsequent elimination of eliminatable vertices (edges) is called vertex (edge) elimination sequence. We denote the vertex (edge) elimination sequence with the help square brackets. The application of the vertex (edge) elimination sequence $[w_1, \dots, w_k]$ to l-DAG G is recursively defined as

$$G - [w_1, \dots, w_k] = \left(G - [w_1, \dots, w_{k-1}] \right) - w_k \quad ,$$

where w_i for $1 \leq i \leq k$ is a vertex (edge) elimination of an eliminatable vertex (edge) on $G - [w_1, \dots, w_{i-1}]$. Similarly the cost of this vertex (edge) elimination sequence are also recursively defined as

$$\sigma_G([w_1, \dots, w_k]) = \sigma_G([w_1, \dots, w_{k-1}]) + \sigma_{G - [w_1, \dots, w_{k-1}]}(w_k) \quad .$$

If the sequence of eliminations contains both vertex and edge eliminations we refer to it simply as elimination sequence or if we want to emphasize this fact as transformation sequence.

An elimination sequence that transforms an l-DAG into a bipartite graph is called a complete elimination sequence.

For the sake of shorter notation we omit writing the referred l-DAG G in notations if l-DAG is G (e.g. $P_G(v) = P(v)$ or $\sigma_G(v) = \sigma(v)$) or the choice of this l-DAG is clear from context.

Now we can define the optimal vertex (edge) and the minimum edge count problems.

Definition 2.6. The problem of finding a complete vertex elimination sequence with minimal costs for a given l-DAG G is called vertex elimination (VE) problem. We refer to a vertex elimination sequence that solves the VE problem as optimal vertex elimination sequence. $VE(G)$ denotes the cost of an optimal vertex elimination sequence.

Definition 2.7. The problem of finding a complete edge elimination sequence with minimal costs for a given l-DAG G is called edge elimination (EE) problem. We refer to a edge elimination sequence that solves the EE problem as optimal edge elimination sequence. $EE(G)$ denotes the cost an optimal edge elimination sequence.

Definition 2.8. The minimum edge count MEC problem is to determine a sequence of front and back edge eliminations such that the resulting l-DAG contains the lowest possible number of edges. $MEC(G)$ denotes the number of edges in the resulting l-DAG.

Definition 2.9. The minimum edge count by vertex elimination MECV problem is to determine a sequence of vertex eliminations such that the resulting l-DAG contains the lowest possible number of edges. $MECV(G)$ denotes the number of edges in the resulting l-DAG.

It should be noted that due to commutativity of vertex and edge eliminations (see Section 3.2) the solution of VE, EE, MEC and MECV problems is in general not unique.

Next we formally define optimality preserving eliminations.

Definition 2.10. Consider l-DAG G . We say that a particular vertex (edge) elimination preserves optimality in the context of VE (MECV, EE, MEC) problem if there exists a vertex (edge) elimination sequence solving VE (MECV, EE, MEC) problem that begins with this vertex (edge) elimination. If the context problem is clear we avoid mentioning it.

As we can see from the Definition 2.4, edge elimination is a more general elimination method than vertex elimination. Hence it is clear that $EE(G) \leq VE(G)$. This result was already proved by Naumann in [Nau04].

Lemma 2.1. For any l-DAG G holds

$$EE(G) \leq VE(G) \quad .$$

Proof. The proof follows immediately from the definition of vertex elimination as a sequence of edge eliminations. $\square$

In [Nau04] Naumann presented an example where vertex elimination do not solve the EE problem.

Lemma 2.2. A solution of VE does not solve EE on arbitrary l-DAGs.

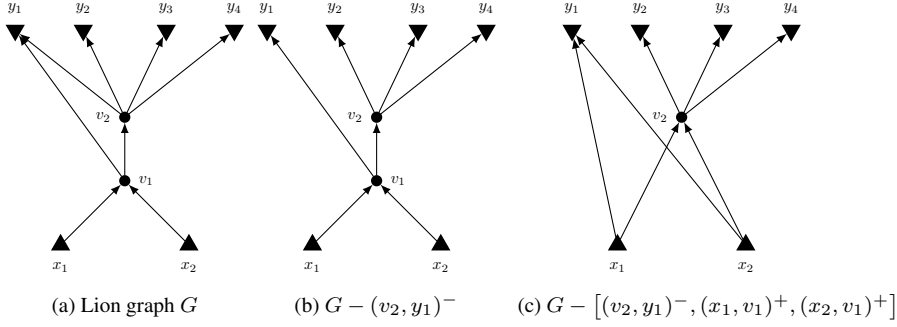


Figure 2.2: Proof of Lemma 2.2

Proof. This proof was presented by Naumann in [Nau99]. Consider the l-DAG G as on Figure 2.2(a). This graph is also called lion graph due to its shape. Both possible vertex elimination sequences $[v_1, v_2]$ and $[v_2, v_1]$ yield the same costs of 12 fmas. Using edge elimination we can transform the l-DAG G to bipartite graph with smaller costs. Therefore we first back eliminate the edge (v_2, y_1) at the costs of 1 fma. On the resulting l-DAG $G - (v_2, y_1)^-$ (compare Figure 2.2(b)), we then front eliminate the edges (x_1, v_1) and (x_2, v_1) to remove the vertex v_1 . These two front edge eliminations require elimination 4 fmas. The resulting l-DAG is shown on Figure 2.2(c). The front elimination of the edges (x_1, v_2) and (x_2, v_2) on the resulting l-DAG transforms the l-DAG into a bipartite graph, the costs of these eliminations are 6 fmas. Hence the overall cost of this edge elimination sequence adds up to 11 fmas. So vertex elimination does not solve the EE problem on the lion graph. In [Nau04] Naumann proved that 11 is the solution of the EE problem on this graph. In Example 6.23 we will show a shorter proof of this fact, exploiting the results of this thesis. $\square$

The relationship between edge and vertex elimination in the context of the MEC problem is similar to that in the context of OJA problem. The solution of the MEC problem is at least as good as that of MECV problem.

Lemma 2.3. *For any l-DAG G holds*

$$\text{MEC}(G) \leq \text{MECV}(G)$$

Proof. Follows immediately from the definition of vertex elimination as a sequence of edge eliminations. $\square$

Vertex elimination does not solve the MEC problem in general case. The corresponding proof was presented in [MN12].

Lemma 2.4. *A solution of the MECV problem does not solve the MEC problem on arbitrary l-DAGs.*

Proof. An example for which the statement is true is presented in Figure 2.3(a). All four possible vertex elimination sequences $[v_1]$, $[v_2]$, $[v_1, v_2]$ and $[v_2, v_1]$ increase the number of edges. Hence $\text{MECV}(G) = 9$. Back elimination of edge (v_2, y_1) (compare Figure 2.3(b)) followed by front elimination of edge (x_3, v_1) results in a l-DAG that contains only 8 edges (compare Figure 2.3(c)) completing the proof. $\square$

Although edge elimination is a more general elimination method, the investigation of vertex elimination is still of interest as the search space is much smaller than that induced by edge elimination. The search space can be reduced even further using the result from Lemma 3.7 showing that permutations

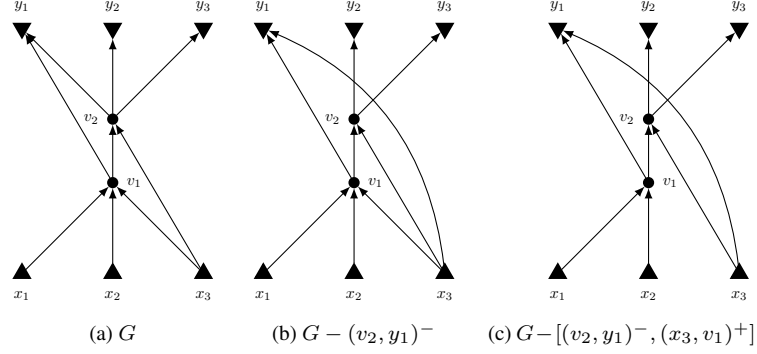


Figure 2.3: Proof of Lemma 2.4

of vertex eliminations does not change the resulting graph (for more details see Section 3.2.4 and Section 3.2.5). With the proofs that vertex elimination solves the MEC and EE problem on absorption-free l-DAGs (see Theorem 4.3 and Theorem 4.4) vertex elimination becomes even more useful (see Chapter 4). Also as outlined in Chapter 7, vertex elimination is the method of choice when it comes to developing fast algorithm that compute near optimal solutions for MEC or EE problems.

As already mentioned in the introduction, edge elimination does not solve the structural OJA problem in general. The l-DAG that was used as a counter example in this proof is shown on Figure 2.4. Due to its shape this l-DAG is also known as the bat graph. In the proof it was shown that optimal edge elimination

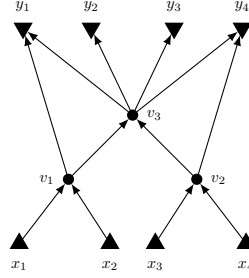


Figure 2.4: Bat graph

on the bat graph requires at least 23 fma's, while the Jacobian could be obtained at cost of 22 fma's. In this paper a new elimination method based on eliminations on the dual graph was introduced. This elimination technique is called *face elimination*, the corresponding elimination problem is referred to as *optimal face elimination* problem (FE). It was also shown that with face eliminations the Jacobian of the bat graph could be obtained at the costs of 22 fmas. In [GW08] Griewank proved that $FE = OJA$ on absorption-free l-DAGs. For the general case the proof is still outstanding. We will not pursue face eliminations any further as it goes beyond scope of this thesis. The main idea of this thesis is to improve existing algorithms for solving the VE and MECV problems by providing better lower bounds, proving various optimality preserving eliminations and showing how to obtain better and faster solutions by mixing vertex and edge eliminations. As face elimination works on the dual representation of the l-DAG it is not clear how face elimination can contribute to this objective. Despite the big flexibility of face elimination, solving the OJA or MEC problem with this elimination method directly does not seem the way to go. The main reason for this is the fact that the search space of face elimination is much bigger compared to that

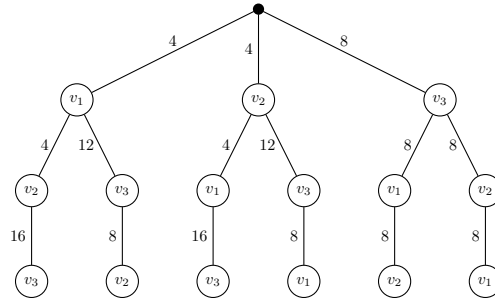


Figure 2.5: Depth first search for the VE problem on the bat graph (compare Figure 2.4). Edge labels represent the costs of the corresponding vertex eliminations.

of vertex or edge elimination. The potential advantage of using face elimination seems not to justify the additional effort involved by using face elimination. Supporting evidence is provided in [Pei09]. In this bachelor thesis a branch and bound algorithm was applied to solve the VE, EE and FE problems for various I-DAGs. Solving the FE problem for the lion graph with this approach took more than 3 hours, while EE problem for the same graph could be solved in less than 5 seconds. Face elimination could not improve the solution delivered by the edge elimination. Both required 11 fma's to accumulate the Jacobian. Face elimination also seems to be not the right choice for solving the MEC problem. One aim of solving the MEC problem is to find a memory efficient representation of the Jacobian matrix. Face elimination works on dual graphs and it is not always possible to transform the dual graph back to the I-DAG after performing several face eliminations. So when using face elimination one should be prepared to store the dual graph representation of the Jacobian matrix, which requires more memory compared to storing the original I-DAG. It is very unlikely that a potentially better solution provided by face elimination can compensate for this drawback. Still face elimination is a very interesting technique from theoretical point of view and should be part of research. Possible entry points for such investigation and connections to this thesis are given in Chapter 7.

2.2 Generic branch and bound algorithm

All four problems (VE, EE, MEC and MECV) discussed in this thesis are combinatorial optimizations problems. Moreover these problems are conjectured to be NP-hard. One widely used algorithm to solve such kind of problems is the branch and bound algorithm ([LMSK63, LW66]). In the following we develop a generic branch and bound algorithm (Algorithm 1) that can be used to solve these problems. The reason why we develop a generic algorithm, is that we will use it to solve both the minimal edge count problem by (vertex) edge elimination and the optimal (vertex) edge elimination problems. The algorithm presented here is used to demonstrate how the theoretical results developed in this thesis can be applied to solve the particular problem. We also use this algorithm to demonstrate the impact of these result on the size of the search space. Especially we will show how the commutativity of edge and vertex elimination can be used within such an algorithm to reduce the search space.

The branch and bound algorithm presented here is based on depth first search. The search space induced by performing the depth first search for the VE problem on the bat graph (compare Figure 2.4) is shown in Figure 2.5. The algorithm is implemented as a recursive algorithm. It takes the I-DAG G and the upper bound ub as inputs and computes the elimination sequence that solves the problem and the corresponding costs. The resulting elimination sequence is stored in `solution` which is a queue that handles the elements in first-in, first-out (FIFO) order. It provides the functions `solution.put` and `solution.pull` to store and retrieve the items. The algorithm mainly consists of the function `COM-`

PUTESOLUTION($G, G_p, \text{elim}, \text{ub}, \text{solution}$) that is called recursively. This function takes $G, G_p, \text{elim}, \text{ub}$ as inputs and stores its result in solution . G is hereby the l-DAG that has to be solved. ub is the upper bound for the costs of the solution (solution). The remaining two arguments G_p and elim are used only to exploit commutativity of edge and vertex eliminations. To be able to exploit the commutativity we need to know the elimination elim that transformed G_p into G . The second function that is used by the algorithm is COSTOFSOLUTION. It computes the costs of solution on G . In case of minimal edge count problem the cost is the number of edges in the l-DAG after all eliminations stored in solution are applied. In the case of optimal vertex and edge elimination problems the cost is the number of fmas performed by the elimination sequence contained in solution .

Algorithm 1 Generic branch and bound algorithm

Require: l-DAG G, ub // ub – upper bound for optimal solution can be ∞
Ensure: $\text{solution}, \text{costs}$ // Optimal elimination sequence and the costs

```

1: COMPUTESOLUTION( $G, \emptyset, \emptyset, \text{ub}, \text{solution}$ )
2:  $\text{costs} \leftarrow \text{COSTOFSOLUTION}(G, \text{solution})$ 
3:
4: function COMPUTESOLUTION( $G, G_p, \text{elim}, \text{ub}, \text{solution}$ ) // Computes optimal solution for  $G$ 
5:    $\text{found} \leftarrow \text{true}$ 
6:    $\text{lb} \leftarrow \text{COMPUTELOWERBOUND}(G)$ 
7:   if  $\text{lb} > \text{ub}$  then
8:      $\text{found} \leftarrow \text{false}$ 
9:   else
10:     $M \leftarrow \text{GETELIMINATIONS}(G)$  // Store possible eliminations in  $M$ 
11:     $\text{REMOVEPERMUTABLEELIMINATIONS}(G_p, \text{elim}, M)$  // Exploit commutativity
12:    while  $M \neq \emptyset$  do
13:      choose  $e1 \in M$  //  $e1$  can be vertex or front (back) edge elimination
14:       $G' \leftarrow \text{PERFORMELIMINATION}(G, e1)$ 
15:       $\text{ub\_tmp} \leftarrow \text{UPDATEUPPERBOUND}(G, e1, \text{ub})$ 
16:       $\text{found} \leftarrow \text{COMPUTESOLUTION}(G', G, e1, \text{ub\_tmp}, \text{solution\_tmp})$  // Recursion
17:      if  $\text{found}$  then // New best solution found
18:         $\text{solution} \leftarrow \text{solution\_tmp.put}(e1)$  // Store best elimination sequence
19:         $\text{costs} \leftarrow \text{COSTOFSOLUTION}(G, \text{solution\_tmp})$ 
20:         $\text{ub} \leftarrow \text{costs} - 1$ 
21:        if  $\text{costs} == \text{lb}$  then // Optimal solution for  $G$  was found
22:          exit
23:        end if
24:      end if
25:    end while
26:  end if
27:  return  $\text{found}$ 
28: end function
29:
30: function PERFORMELIMINATION( $G, e1$ ) // Applies elimination  $e1$  to  $G$  and returns the result
31:   return  $G - e1$  //  $e1$  can be vertex or front (back) edge elimination
32: end function

```

The function COMPUTESOLUTION works as follows. At the beginning we set the variable found to **true**. This variable identifies whether the COMPUTESOLUTION function could find a valid solution for G or not. Valid solution means that the costs of this solution are not higher than the provided upper

bound ub . Afterwards we compute the lower bound for the solution of l-DAG G . If the lower bound is greater than the upper bound provided in ub , we exit the function as we cannot solve the problem with the requested costs. In the global view this means that we do not need to investigate the current branch any further as this branch will not lead us to the optimal solution. If the lower bound for G is not higher than the provided upper bound we determine all possible eliminations calling the `GETELIMINATIONS` function and store them in M . The subsequent call to `REMOVEPERMUTABLEELIMINATIONS` removes the not needed eliminations from M exploiting the commutativity of edge or vertex eliminations. For each elimination $e1$ in M we first perform the elimination of $e1$ on G using the function `PERFORMELIMINATION` and store the resulting l-DAG in G' (Note that $e1$ can be back or front edge elimination, or even a vertex elimination). Then we try to find an optimal solution for G' . Therefore first of all we have to update the upper bound appropriately using the `UPDATEUPPERBOUND` function and then apply `COMPUTESOLUTION` recursively to G' with the updated upper bound. If `COMPUTESOLUTION` could find a valid solution for G' we need to update our current best solution in `solution` and compute its costs. We then set our current upper bound to the costs of this solution minus one. We subtract one to ensure that the next solution we find has smaller costs compared to the current one. If the costs of the current solution are equal to the lower bound we determined for G , we know that the optimal solution for G was found and therefore we do not have to investigate this branch any further.

2.3 Further definitions and notation

In this section all further definitions, assumptions and notations that are used throughout the whole thesis are gathered. Only some section specific definitions will be defined at the place where they are used.

Extending the definitions in Definition 2.1 to sets of vertices (edges) yields.

Definition 2.11. Consider an l-DAG $G = (V, E)$, $T \subseteq V$ and $S \subseteq E$. We set,

$$\begin{aligned} \text{src}(S) &= \{\text{src}((i, j)) \mid (i, j) \in S\}, & \text{tgt}(S) &= \{\text{tgt}((i, j)) \mid (i, j) \in S\}, \\ S(T) &= S_G(T) = \bigcup_{i \in T} S(i) & P(T) &= P_G(T) = \bigcup_{i \in T} P(i) \quad . \end{aligned}$$

The definition of l-DAG already distinguishes between the three different subsets of vertices: inputs outputs and intermediate vertices. The next definition defines the mathematical notation for these sets which will be used in the various proof.

Definition 2.12. Consider l-DAG $G = (V, E)$. We set

$$\begin{aligned} X &= X(G) = \{v \in V \mid P_G(v) = \emptyset\}, & Y &= Y(G) = \{v \in V \mid S_G(v) = \emptyset\} \quad , \\ Z &= Z(G) = V \setminus (X \cup Y) \quad . \end{aligned}$$

We refer to vertices from X as inputs, Y as outputs and Z as intermediate vertices.

Only the elements of $Z(G)$ can be eliminated during the edge or vertex elimination. Therefore we also refer to them as eliminatable vertices. Similarly we can define a set of eliminatable edges.

Definition 2.13. Consider l-DAG G . The set

$$E^+ = E^+(G) = \{(i, j) \mid j \in Z(G)\}$$

is called the set of front eliminatable edges. The set

$$E^- = E^-(G) = \{(i, j) \mid i \in Z(G)\}$$

is called the set of back eliminatable edges, while we refer to $E^*(G) = E^+ \cup E^-$ as the set of eliminatable edges.

An eliminatable edge $(i, j) \in E^*(G)$ is called intermediate edge if $i \in Z(G)$ and $j \in Z(G)$. Otherwise we refer to such an edge as non-intermediate.

To keep the illustrations cleaner and to improve the legibility we avoid drawing and mentioning edges connecting inputs with the outputs, in the context of EE and VE problems. Those edges cannot be eliminated and they also have no influence on the costs of a vertex or edge elimination. In the context of MEC and MECV problems we do keep these edges as they have influence on the solution.

Definition 2.14. Consider l-DAG G , $v \in Z(G)$. Following the definition in [GR91] the Markowitz degree of the vertex v is number of multiplications needed to eliminate the vertex v and is equal to

$$\text{mark}_G(v) = |P_G(v)| \cdot |S_G(v)| \quad .$$

Next we define the notation for the subset of inputs or outputs that are connected to a given vertex or edge. This notation is used in Chapter 4 to formalize claims and algorithms.

Definition 2.15. Consider l-DAG G , $v \in V(G)$ and $(i, j) \in E(G)$. We set

$$X_G(v) = \{w \in X(G) \mid \text{there exists a directed path from } v \text{ to } w\}$$

and for the outputs

$$Y_G(v) = \{w \in Y(G) \mid \text{there exists a directed path from } v \text{ to } w\}$$

We refer to $X_G(v)$ as inputs from which v is reachable and to $Y_G(v)$ as outputs that are reachable from v on G . Analogously for the edges we set

$$X_G((i, j)) = X_G(i)$$

and for the outputs

$$Y_G((i, j)) = Y_G(j)$$

We refer to $X_G((i, j))$ as inputs from which (i, j) can be reached and to $Y_G((i, j))$ as outputs that are reachable from (i, j) on G .

Let us discuss this definition on the following example.

Example 2.3. Consider the l-DAG G as in Figure 2.2(b).

$$X_G(v_2) = \{x_1, x_2\} \quad , \quad Y_G(v_2) = \{y_2, y_3, y_4\} \quad ,$$

and

$$X_G((v_1, v_2)) = \{x_1, x_2\} \quad , \quad Y_G((v_1, v_2)) = \{y_2, y_3, y_4\} \quad .$$

The next definition is used to define and describe subgraphs in Chapter 3 and Chapter 4.

Definition 2.16. Consider l -DAG G , and $S \subseteq E(G)$. We set $T = (V', S)$, where $V' = (\text{src}(S) \cup \text{tgt}(S))$. Then we recursively remove all vertices $v \in Z(G)$ with $|P_T(v)| = 0$ or $|S_T(v)| = 0$ from T . We refer to the resulting graph as the l -DAG induced by the set S of edges and denote it with $H[S]$.

Example 2.4. Consider the lion graph G (see Figure 2.2(a)). The l -DAG induced by the set

$$S_1 = \{(x_1, v_1), (v_1, v_2), (v_2, y_2), (v_2, y_3)\}$$

is presented in Figure 2.6(a).

Consider the bat graph G (see Figure 2.4). The l -DAG induced by the set

$$S_2 = \{(x_1, v_1), (x_2, v_1), (v_1, y_1), (v_1, v_3)\}$$

is presented in Figure 2.6(b). Note that $v_3 \notin V(H[S_2])$ although $(v_1, v_3) \in S_2$, as v_3 has no outgoing edge in S_2 .

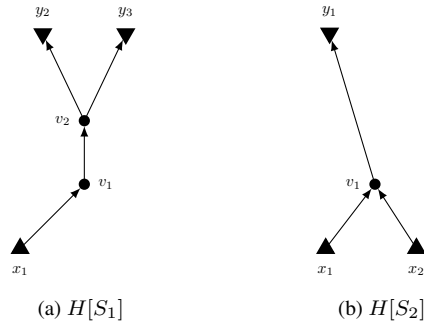


Figure 2.6: Example 2.4 l -DAG induced by a set of edges

Definition 2.17. The set of intermediate vertices with exactly one in-edge or one out-edge is denoted by $\mathcal{M}(G) = \{v \in Z(G) \mid |S_G(v)| = 1 \vee |P_G(v)| = 1\}$.

In some algorithms (e.g. Algorithm 16, Algorithm 17) we need a total ordering on vertices and edges thus we define the " $<$ " operator on these sets.

Definition 2.18. Consider l -DAG G . For $a, b \in V(G)$

$$a < b \Leftrightarrow \pi(a) < \pi(b) \quad ,$$

where $\pi : X(G) \cup Z(G) \cup Y(G) \rightarrow \{1-n, \dots, 0, 1, \dots, p+m\}$ maps a vertex on to its ordering number in the SAC.

For $e_1, e_2 \in E(G)$

$$e_1 < e_2 \Leftrightarrow (\text{src}(e_1) < \text{src}(e_2)) \vee ((\text{src}(e_1) = \text{src}(e_2)) \wedge (\text{tgt}(e_1) < \text{tgt}(e_2))) \quad .$$

Chapter 3

Properties of linearized directed acyclic graphs

This chapter contains proofs of basic properties of l-DAGs. These properties build the basis for almost all results of this thesis. With the exception of Section 3.2.3 and Section 3.2.5 these results have already been published in [MN12].

This chapter is split into two sections. In the first section (Section 3.1) we show that the fact that one l-DAG is a subgraph of another l-DAG is invariant under edge (vertex) eliminations and discuss its consequences. We refer to this property of l-DAGs as "subgraph monotonicity". The latter is one of the most important results of this thesis. It allows us to reduce the problem (EE, VE, MEC or MECV) to a subgraph of the original l-DAG. This result can be seen as a glue between the results for absorption-free (Chapter 4) and arbitrary l-DAGs (Chapter 5, Chapter 6), as it permits the reuse of results from absorption-free l-DAGs on arbitrary l-DAGs. New algorithms that compute lower bounds for the MEC, MECV, VE and EE problems, presented in this thesis, are based on this property. Reduction to subgraphs is also used for generic optimality preserving techniques for MEC (Theorem 5.2, Theorem 5.3), MECV (Theorem 5.1), EE and VE (Lemma 4.18) problems.

In the last section of this chapter (Section 3.2) we show in which cases the order of edge and vertex eliminations does not change the resulting l-DAG. The proofs of the most important findings of this thesis namely Theorem 4.2 ($EE(G) = VE(G)$ on absorption-free l-DAGs) and Theorem 4.4 ($MEC(G) = MECV(G)$ on absorption-free l-DAGs) are based on these results. Using these results we show that solving the minimal edge count problem by vertex eliminations we do not need to check all possible permutations for a given subset of intermediate vertices as they do not have any influence on the number edges in the resulting l-DAG (Section 3.2.4). Thus we can concentrate on finding a proper subset of intermediate vertices that has to be eliminated. These results can also be used to reduce the search space when solving the EE, VE, MEC or MECV problem with branch and bound algorithm (compare Section 5.2 and Section 6.2).

Apart from the properties of an l-DAG we also introduce a new technique. We refer to this technique as bi-coloring of an l-DAG (Definition 3.2). This technique allows us to see the influence of particular edges on the number of multiplications for a given elimination sequence. It is also used to prove the "subgraph monotonicity" property.

3.1 Subgraph monotonicity

In this section we discuss "subgraph monotonicity" and show its consequences for the EE, VE and MEC problems. While "subgraph monotonicity" and its consequences for the minimal edge count problem

were already published in [MN12], the implications for the EE and VE problems have not been presented to the public so far.

In Section 3.1.1 we prove the "subgraph monotonicity" for l-DAGs. The "subgraph monotonicity" property means that after the application of any edge (vertex) elimination to both G and subgraph G' the resulting l-DAG of G' is still a subgraph of the resulting l-DAG of G . This property is one of the central results of this thesis. In Section 3.1.2 and Section 3.1.3 we describe the consequences of this property for VE, EE and MEC problems. The results from this section build the basis for efficient lower bounds and various optimality preserving eliminations developed later in this thesis (see Chapter 5 and Chapter 6).

In the commonly used definitions of edge (vertex) elimination the actual elimination of an edge (vertex) is merged with potential subsequent absorption of parallel edges (edges with same source and sink). It is also the case in the definitions of edge and vertex eliminations presented in the previous chapter (Definition 2.3, Definition 2.2 and Definition 2.4). This approach is based on the assumption that immediate absorption preserves optimality. So far this assumption has not been proved in the literature for any of the problems. A first step was done in [MN12] where "subgraph monotonicity" was used to prove this assumption for the MEC problem. Here we will also prove the corresponding result for the VE and EE problems. Therefore we first split the edge elimination on l-DAGs in front (back) elimination of an edge and absorption of parallel edges. Once we have proved that immediate absorption preserve optimality in the context of all three problems VE, EE and MEC, we will switch back to the commonly used definitions of edge and vertex eliminations.

3.1.1 Subgraph monotonicity: The proof

The results of this section are part of [MN12]. In this section we prove that for two l-DAGs G' and G with $G' \subseteq G$ any elimination sequence applied to both l-DAGs does not destroy the *subset property* (compare in Theorem 3.1). To prove this theorem we first show that application of single elimination maintains the subset property in Lemma 3.1, Lemma 3.2 and Lemma 3.3.

First of all let us define subgraphs.

Definition 3.1. Consider l-DAGs G and G' . G' is a subgraph of G denoted as $G' \subseteq G$ if and only if $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$, $X(G') \subseteq X(G)$ and $Y(G') \subseteq Y(G)$. We also refer to this as subgraph monotonicity.

Next we define bi-coloring of the l-DAG.

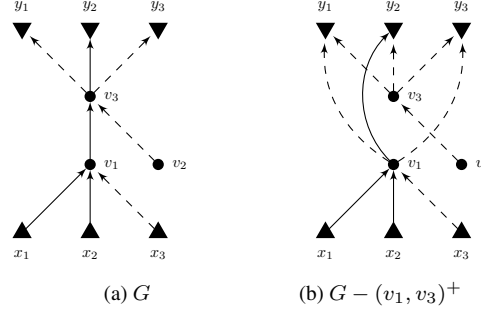
Definition 3.2. Consider l-DAG $G = (V(G), E(G))$. A Boolean mapping $f_G : E(G) \rightarrow \{0, 1\}$ is called a bi-coloring of G .

In this section we apply the same eliminations (elimination sequences) to both G and its subgraph G' . To visualize the differences between the resulting l-DAGs we use bi-coloring. Therefore we color the l-DAG G as follows: the edges of G that are part of G' are colored with 1, the other are colored with 0. Hence we can distinguish between edges of G that belong to G' and those which do not based on their color. To keep this coloring after the elimination we need to update the coloring in a proper way. For this we require further notation.

Definition 3.3. Consider a bi-colored l-DAG G , $v \in V(G)$ and $S \subseteq E(G)$. We set

$$\begin{aligned} S_G^1(v) &= S_G(v) \cap \text{tgt}(f_G^{-1}(1)), & P_G^1(v) &= P_G(v) \cap \text{src}(f_G^{-1}(1)), \\ S_G^0(v) &= S_G(v) \cap \text{tgt}(f_G^{-1}(0)), & P_G^0(v) &= P_G(v) \cap \text{src}(f_G^{-1}(0)), \end{aligned}$$

where $f_G^{-1}(1)$ and $f_G^{-1}(0)$ denote the sets of edges in G mapped onto 1 and 0 by f_G , respectively.

Figure 3.1: Example of front elimination of edge (v_1, v_3) on bi-colored l-DAG

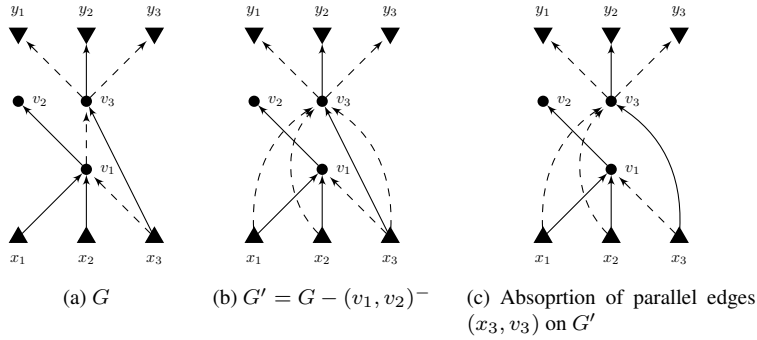
Now we extend the previously introduced transformations with a coloring function on the resulting l-DAG.

Definition 3.4. Consider a bi-colored l-DAG G . The front elimination of an edge (i, j) is performed by introducing new edges (i, j') for all j' with $(j, j') \in E(G)$. The new edge labels are set to $c_{(i, j')} = c_{(i, j)} \cdot c_{(j, j')}$. Edge (i, j) is deleted. If vertex j has only one in-edge in G , then we delete it together with all its incident edges.

The resulting l-DAG is $G' = (V(G'), E(G'))$. For each new edge (i, j') we define its color in G' as $f_{G'}((i, j')) = f_G((i, j)) \wedge f_G((j, j'))$. We set $f_{G'}((j, j')) = 0$ for all j' with $j \prec j'$ if $P_G^1(j) = \{i\}$. We write $G - (i, j)^+$ for the front elimination of (i, j) in G .

See Figure 3.1 for an example. Dashed lines represent 0-colored edges. The front elimination of (v_1, v_3) introduces two new 0-colored edges (v_1, y_1) and (v_1, y_3) and a new 1-colored edge (v_1, y_2) . Edge (v_1, v_3) is removed leaving (v_2, v_3) as the only in-edge of v_3 . The colors of all edges emanating from v_3 become 0 as v_3 does not have 1-colored incoming edge.

The definition of back elimination in bi-colored l-DAGs is similar.

Figure 3.2: Example of back elimination of edge (v_1, v_3) and absorption on bi-colored l-DAG

Definition 3.5. Consider a bi-colored l-DAG G . The back elimination of edge (i, j) is performed by introducing new edges (i', j) for all i' with $i' \prec i$. The new edge labels are set to $c_{(i', j)} = c_{(i, j)} \cdot c_{(i', i)}$, where $(i', i) \in E(G)$. Edge (i, j) is deleted. If vertex i has only one out-edge in G , then we delete it together with all its incident edges.

The resulting l -DAG is $G' = (V(G'), E(G'))$. For each new edge (i', j) we define its color in G' as $f_{G'}((i', j)) = f_G((i, j)) \wedge f_G((i', i))$. We set $f_{G'}((i', i)) = 0$ for all i' with $i' \prec i$ if $S_G^1(i) = \{j\}$. We write $G - (i, j)^-$ for the back elimination of (i, j) in G .

Refer to Figure 3.2(a) and Figure 3.2(b) for an example. The 0-colored edge (v_1, v_3) is back eliminated. New 0-colored edges (x_1, v_3) , (x_2, v_3) , and (x_3, v_3) are introduced. The colors of the in-edges of v_1 are left unchanged as the remaining out-edge (v_1, v_2) carries color 1.

Definition 3.6. Consider a bi-colored l -DAG G , $k_1, k_2 \in E(G)$. If $\text{src}(k_1) = \text{src}(k_2)$ and $\text{tgt}(k_1) = \text{tgt}(k_2)$, the absorption of two edges k_1, k_2 is performed by removing edge k_2 . The label of the edge k_1 is set to $c_{k_1} = c_{k_1} + c_{k_2}$ and $f_{G'}(k_1) = f_G(k_1) \vee f_G(k_2)$. Otherwise G remains unchanged.

We write $G - k_1 || k_2$ for the merging (absorption) of k_1 and k_2 in G .

See Figure 3.2(b) and Figure 3.2(c) for an example of absorption. Vertices x_3 and v_3 are connected by two parallel edges, 0- and 1-colored, respectively. The resulting edge carries color 1. According to the following lemma the subset property is preserved when eliminating an arbitrary 1-colored edge.

Lemma 3.1. Consider a bi-colored l -DAG $G = (V, E)$, where $G' \subseteq G$, with $G' = G[f_G^{-1}(1)]$. The following holds for every edge $(i, j) \in E$, with and $f_G((i, j)) = 1$:

1. $G' - (i, j)^+ \subseteq G - (i, j)^+$ and $E(G' - (i, j)^+) = f_{G - (i, j)^+}^{-1}(1)$,
2. $G' - (i, j)^- \subseteq G - (i, j)^-$ and $E(G' - (i, j)^-) = f_{G - (i, j)^-}^{-1}(1)$.

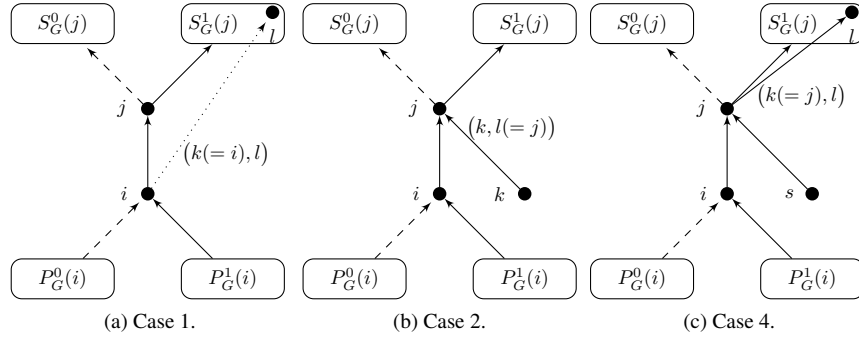


Figure 3.3: Illustration of cases in proof of Lemma 3.1

Proof. Ad 1: Let us first ensure that $V(G') \subseteq V(G)$, $X(G') \subseteq X(G)$ and $Y(G') \subseteq Y(G)$. The statement is obviously true if $V(G)$ is not changed by the elimination. According to the definition of the front elimination of an edge j is the only vertex which can be removed by front elimination of the edge (i, j) . If $j \notin V(G - (i, j)^+)$ then i is the only predecessor of j in G . Hence $j \notin V(G' - (i, j)^+)$ as i is the only predecessor j in G' too because $E(G') \subseteq E(G)$. Therefore $V(G') \subseteq V(G)$, $X(G') \subseteq X(G)$ and $Y(G') \subseteq Y(G)$. It remains to be shown that

$$E(G' - (i, j)^+) = f_{G - (i, j)^+}^{-1}(1) \quad , \quad (3.1)$$

which implies $E(G' - (i, j)^+) \subseteq E(G - (i, j)^+)$ and hence $G' - (i, j)^+ \subseteq G - (i, j)^+$. Equation (3.1) is proved by showing that if $(k, l) \in E(G')$ then $(k, l) \in E(G)$ and vice versa. Hence we consider the following four cases:

- $(k, l) \notin E(D)$,
- $(k, l) \in E(D)$, $k \neq j$ and $l = j$,
- $(k, l) \in E(D)$, $k \neq j$ and $l \neq j$,
- $(k, l) \in E(D)$, $k = j$,

where $D = G'$ or $D = G$ respectively.

” $\subseteq$ ” Consider $(k, l) \in E(G' - (i, j)^+)$.

Case 1. $(k, l) \notin E(G')$. In this case (k, l) is a fill edge that is added to G' due to the elimination of (i, j) . Therefore $k = i$ and $l \in S_{G'}(j)$, which implies that $l \in S_G^1(j)$ (see Figure 3.3(a)). Hence $(k, l) \in E(G - (i, j)^+)$ and $f_{G-(i,j)^+}((k, l)) = f_G((i, j)) \wedge f_G((j, l)) = 1$ yielding $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$.

Case 2. $(k, l) \in E(G')$, $k \neq j$ and $l = j$. From $(k, l) \in E(G')$ follows that $(k, l) \in E(G)$ and $f_G((k, l)) = 1$. The vertex j has at least two in-edges in G as $l = j$, and therefore j is not removed from G (see Figure 3.3(b)). This implies that $(k, l) \in E(G - (i, j)^+)$. The color of (k, l) remains unchanged according to the definition of edge elimination as $k \neq j$. Hence $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$.

Case 3. $(k, l) \in E(G')$, $k \neq j$ and $l \neq j$. From $(k, l) \in E(G')$ follows that $(k, l) \in E(G)$ and $f_G((k, l)) = 1$. Moreover $(k, l) \in E(G - (i, j)^+)$ as (k, l) is not incident with j . According to the definition of edge elimination the color of (k, l) remains unchanged and hence $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$.

Case 4. $(k, l) \in E(G')$ and $k = j$. From $(k, l) \in E(G' - (i, j)^+)$ and $k = j$ follows that j has at least one in-edge apart from (i, j) in G' and therefore j has at least one 1-colored in-edge in G apart from (i, j) (see Figure 3.3(c)). Hence $(k, l) \in E(G - (i, j)^+)$ and $f_{G-(i,j)^+}((k, l)) = f_G((k, l)) = 1$ as $(k, l) \in E(G')$ and $|P_G^1(j)| > 1$.

” $\supseteq$ ” Consider $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$.

Case 1. $(k, l) \notin E(G)$. In this case (k, l) is a fill edge that is added to G due to the elimination of edge (i, j) . Therefore $k = i$ and $l \in S_G(j)$ (see Figure 3.3(a)). From $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$ follows that $f_G((j, l)) = 1$ as $1 = f_{G-(i,j)^+}((k, l)) = f_G((i, j)) \wedge f_G((j, l))$ and therefore $l \in S_G^1(j)$. This observation implies that $l \in S_{G'}(j)$. Hence $(k, l) \in E(G' - (i, j)^+)$.

Case 2. $(k, l) \in E(G)$, $k \neq j$ and $l = j$. From $k \neq j$ and $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$ follows that $f_G((k, l)) = f_{G-(i,j)^+}((k, l)) = 1$, and hence $(k, l) \in E(G')$ (see Figure 3.3(b)). Edge (k, l) is not deleted in G' during the front elimination of (i, j) as (k, l) is another in-edge of j in G' apart from (i, j) . Therefore $(k, l) \in E(G' - (i, j)^+)$.

Case 3. $(k, l) \in E(G)$, $k \neq j$ and $l \neq j$. From $k \neq j$ and $(k, l) \in f_{G-(i,j)^+}^{-1}(1)$ follows that $f_G((k, l)) = f_{G-(i,j)^+}((k, l)) = 1$, and hence $(k, l) \in E(G')$. Edge (k, l) is not deleted in G' during the front elimination of (i, j) as it is not incident with j . Therefore $(k, l) \in E(G' - (i, j)^+)$.

Case 4. $(k, l) \in E(G)$ and $k = j$. From $f_{G-(i,j)^+}((k, l)) = 1$ and $k = j$ follows that j has at least one 1-colored in-edge in G apart from (i, j) . Therefore j has also at least one in-edge apart from (i, j) in G' (see Figure 3.3(c)), which implies that j is not deleted in G' during the front elimination of (i, j) . Additionally according to the definition of front elimination of an edge in bi-colored 1-DAGs

$$f_G((k, l)) = f_{G-(i,j)^+}((k, l)) = 1 \quad .$$

Therefore $(k, l) \in E(G')$, and as j is not deleted, $(k, l) \in E(G' - (i, j)^+)$.

Ad 2: The proof is analogous to the proof of I due to symmetry. $\square$

The following Lemma states that elimination of a 0-colored edge (i, j) from a bi-colored l-DAG G yields that the induced subgraph G' is still a subgraph of the l-DAG obtained by elimination of (i, j) from G . Proving this Lemma turns out to be straight-forward.

Lemma 3.2. *Consider a bi-colored l-DAG $G = (V, E)$ where $G' \subseteq G$, with $G' = G[f_G^{-1}(1)]$. For every edge $(i, j) \in E$, with $f_G((i, j)) = 0$ holds*

1. $G' \subseteq G - (i, j)^+$ and $E(G') = f_{G-(i,j)^+}^{-1}(1)$,
2. $G' \subseteq G - (i, j)^-$ and $E(G') = f_{G-(i,j)^-}^{-1}(1)$.

Proof. Ad 1: Let us first ensure that $V(G') \subseteq V(G)$, $X(G') \subseteq X(G)$ and $Y(G') \subseteq Y(G)$. The statement is obviously true if $V(G)$ is not changed by the elimination. According to the definition of front elimination of an edge, j is the only vertex which can be removed through front elimination of (i, j) . If $j \notin V(G - (i, j)^+)$ then i is the only predecessor of j in G . Hence $j \notin V(G')$ and therefore $V(G') \subseteq V(G)$, $X(G') \subseteq X(G)$ and $Y(G') \subseteq Y(G)$.

By Definition 3.4 the front elimination of a 0-edge preserves all 1-edges. If 1-edges emanate from j , then there are other 1-edges leading into j . Hence the colors of all edges of G remain unchanged in $G - (i, j)^+$.

Ad 2: This case is analogous to I due to symmetry. $\square$

It remains to be shown that the subset property also holds for l-DAGs resulting from absorption.

Lemma 3.3. *Consider a bi-colored l-DAG $G = (V, E)$ where $G' \subseteq G$, with $G' = G[f_G^{-1}(1)]$. For every pair of parallel edges $e_1, e_2 \in E$ holds*

$$G'_1 \subseteq G' \text{ and } E(G'_1) = f_{G'}^{-1}(1) \quad ,$$

where $G'_1 = G' - e_1 \parallel e_2$ and $G' = G - e_1 \parallel e_2$.

Proof. Absorption does not change the vertex sets, therefore $V(G'_1) \subseteq V(G')$, $X(G'_1) \subseteq X(G')$ and $Y(G'_1) \subseteq Y(G')$. Hence we only have to show that

$$E(G'_1) = f_{G'}^{-1}(1) \quad .$$

As $E(G') = f_G^{-1}(1)$ and absorption effects only the edges e_1 and e_2 it is sufficient to show that

$$e_i \in E(G'_1) \quad \Leftrightarrow \quad e_i \in f_{G'}^{-1}(1), \quad 1 \leq i \leq 2 \quad .$$

Case 1. $f_G(e_1) = f_G(e_2) = 1$. Prior to absorption both e_1 and e_2 are contained in G' and G respectively. According to the definition of absorption e_2 has to be removed from both l-DAGs while e_1 remains. Moreover $f_{G'}(e_1) = f_G(e_1) \vee f_G(e_2) = 1$ and therefore $e_1 \in f_{G'}^{-1}(1)$.

Case 2. $f_G(e_1) = f_G(e_2) = 0$. Obviously, $e_1, e_2 \notin E(G')$ and therefore $e_1, e_2 \notin E(G'_1)$. According to the definition of absorption e_2 is removed from G . Moreover $f_{G'}(e_1) = f_G(e_1) \vee f_G(e_2) = 0$ and therefore $e_1 \notin f_{G'}^{-1}(1)$.

Case 3. $f_G(e_1) = 0$ and $f_G(e_2) = 1$. Obviously, $e_2 \in E(G')$ and $e_1 \notin E(G')$. Therefore $e_2 \in E(G'_1)$ and $e_1 \notin E(G'_1)$. According to the definition of absorption e_2 is removed from G . Moreover $f_{G'}(e_1) = f_G(e_1) \vee f_G(e_2) = 1$ and therefore $e_1 \in f_{G'}^{-1}(1)$. Hence $E(G'_1)$ is isomorphic to $f_{G'}^{-1}(1)$ as e_1 and e_2 have the same target and source.

Case 4. $f_G(e_1) = 1$ and $f_G(e_2) = 0$. The proof of this case is analogous to the proof of **Case 3** due to symmetry.

□

The following example shows that the requirements $X(G') \subseteq X(G)$ and $Y(G') \subseteq Y(G)$ in Definition 3.1 are essential to ensure that the subset property is not destroyed by edge elimination.

Example 3.1. Consider the bi-colored l-DAG G in Figure 3.4(a). Obviously $X(G') \not\subseteq X(G)$ where $G' = G[f_G^{-1}(1)]$. Front elimination of edge (x_1, v_1) yields an l-DAG that does not contain vertex v_1 (compare Figure 3.4(b)). Although G' still contains this vertex. Hence in this case the subset property is broken by edge elimination.

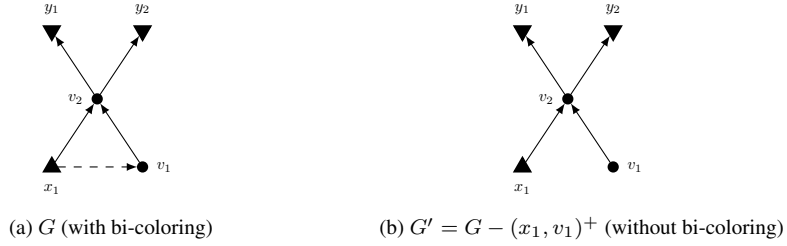


Figure 3.4: Illustration of Example 3.1

Next we extend the results to sequences of edge eliminations and absorptions. For rigorous formulation of such sequences we need the following notation.

Definition 3.7. Consider an l-DAG $G = (V, E)$. The transformation operator $T_{k_1, k_2} = T_{k_1, k_2}(G)$ is defined as

$$T_{k_1, k_2}(G) = \begin{cases} G - k_1^+ & \text{if } k_1 \in E^+, k_2 = \emptyset, \\ G - k_2^- & \text{if } k_2 \in E^-, k_1 = \emptyset, \\ G - k_1 \parallel k_2 & \text{if } k_1, k_2 \in E, \text{ and } \text{src}(k_1) = \text{src}(k_2) \wedge \text{tgt}(k_1) = \text{tgt}(k_2), \\ G & \text{otherwise} \end{cases}.$$

we use the composition to denote the subsequent execution of the transformation operator

$$T_{k_3, k_4} \circ T_{k_1, k_2}(G) = (T_{k_3, k_4}(T_{k_1, k_2}(G))) \quad .$$

So the transformation operator performs front (back) elimination of an edge if we supply only one edge. In the case that front (back) elimination is not applicable, for example if we try to eliminate an edge that connects the inputs with the outputs, the l-DAG is left unchanged. If we provide two edges the transformation operator performs absorption of these two edges. The absorption is performed only in the case that the supplied edges are parallel (have the same sink and source). Otherwise the l-DAG is not changed. The following theorem is the main result of this section. It shows that the subset property is not destroyed by any sequence of edge eliminations and absorptions as an immediate consequence from Lemma 3.1, Lemma 3.2, and Lemma 3.3.

Theorem 3.1. Consider a bi-colored l-DAG $G = (V, E)$ with $G' \subseteq G$, where $G' = (G[f_G^{-1}(1)])$. The following holds for every sequence of eliminations $T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}$, with $k_i, l_i \in E(G_{i-1}) \cup \{\emptyset\}$ for all $1 \leq i \leq n$ and $G_i = (T_{k_i, l_i} \circ \dots \circ T_{k_1, l_1}(G))$:

$$T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G') \subseteq T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G)$$

and

$$E(T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G')) = f_{T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G')}^{-1}(1) \quad .$$

Proof. We perform an induction over n . The case $n = 1$ follows directly from Lemma 3.1, Lemma 3.2, and Lemma 3.3, because in this case we perform only one edge elimination or absorption.
 $n \mapsto n + 1$: From Lemma 3.1, Lemma 3.2, and Lemma 3.3 follows that

$$X(G[f_G^{-1}(1)]) \subseteq X(G_n) \quad \text{and} \quad Y(G[f_G^{-1}(1)]) \subseteq Y(G_n) \quad .$$

Hence G_n satisfies the assumptions of Lemma 3.1, Lemma 3.2, and Lemma 3.3. Application of each of these lemmas (depending on which transformation is performed on G_n) proves the claim. $\square$

Now we can finally prove the "subgraph monotonicity".

Theorem 3.2. Consider l-DAGs G' and G , where $G' \subseteq G$. The following holds for every sequence of eliminations $T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}$, with $k_i, l_i \in E(G_{i-1}) \cup \{\emptyset\}$ for all $1 \leq i \leq n$ and $G_i = (T_{k_i, l_i} \circ \dots \circ T_{k_1, l_1}(G))$:

$$T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G') \subseteq T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G) \quad .$$

Proof. First we color G such that l-DAG G' is the l-DAG induced by the 1-colored edges:

$$f_G(e) = \begin{cases} 1 & \text{if } e \in E(G'), \\ 0 & \text{if } e \notin E(G') \end{cases} \quad .$$

Hence $G' = G[f^{-1}(1)]$. From Theorem 3.1 follows that for an arbitrary elimination sequence $T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}$

$$T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G') \subseteq T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G) \quad .$$

$\square$

3.1.2 Subgraph monotonicity: Consequences for the minimal edge count problem

In this section we use "subgraph monotonicity" (Theorem 3.2) to prove that immediate absorption preserves optimality in the context of the MEC problem (Theorem 3.5). This result was already published in [MN12].

Further we prove that the solution of the MEC problem on a subgraph of G is a lower bound for the solution of the MEC problem on G (Theorem 3.3). This result builds the base for developing lower bounds and optimality preserving eliminations for the minimal edge count problem presented in Section 5.3 and Section 5.4.

Theorem 3.3. Consider l-DAGs G' and G , where $G' \subseteq G$.

$$\text{MEC}(G') \leq \text{MEC}(G) \quad .$$

Proof. From Theorem 3.2 follows that for an arbitrary elimination sequence $T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}$

$$T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G') \subseteq T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}(G)$$

and therefore $\text{MEC}(G') \leq \text{MEC}(G)$. $\square$

An immediate conclusion is the fact that keeping parallel edges increases the search space while not improving the solution of the minimal edge count problem.

Theorem 3.4. *Consider a l-DAG G and two parallel edges $k_1, k_2 \in E(G)$ and let $G' = (G - k_1) \parallel k_2$. Then $\text{MEC}(G') = \text{MEC}(G)$.*

Proof. With $G' \subseteq G$ and Theorem 3.3 follows that $\text{MEC}(G') \leq \text{MEC}(G)$. From $G' = G - k_1 \parallel k_2$ follows that $\text{MEC}(G') \geq \text{MEC}(G)$ and hence $\text{MEC}(G') = \text{MEC}(G)$. $\square$

Theorem 3.4 leads us to the conclusion that immediate absorption preserves optimality in the context of the MEC problem.

Theorem 3.5. *Immediate absorption preserves optimality in the context of the MEC problem.*

Proof. The theorem follows trivially from Theorem 3.4. $\square$

3.1.3 Subgraph monotonicity: Consequences for the optimal edge and vertex elimination problems

In this section we show that similar results we proved for the MEC problem in Section 3.1.2 also hold for the EE and VE problems.

First we set the theoretical basis for commonly used elimination techniques (Theorem 3.7 immediate absorption preserves optimality). Further we prove that we can use the solution of the VE or EE problems on a subgraph of G as a lower bound for the solution of the problem on G itself (Theorem 3.8 and Theorem 3.9). These results build the base for new lower bounds, based on reduction to absorption-free l-DAGs, presented in Section 6.3.2. Also optimality preserving eliminations discussed in Section 6.5 are build thereon.

Before we can proceed we need to define the costs for edge elimination and absorption. We decided to use fma as the measure for complexity, hence the costs of front [back] eliminating an edge without subsequent absorption are the same as in Definition 2.2 and Definition 2.3 since the number of multiplications has not changed. For absorption of two edges according to Definition 3.6 it becomes more complicated. To absorb two edges we have to add their edge labels. If we perform the addition together with the multiplication we can get the sum operation for free (fma) if not, the absorption causes additional computational overhead. For simplification we assume that the costs of all absorptions are 0. This simplification is possible because we intend to show that performing absorption separately from the elimination of the edge cannot provide better results for solving the EE or VE problems, compared to the commonly used definitions of edge and vertex eliminations. So if we can prove this fact under assumption that the costs of all absorptions are 0, we automatically get the proof for the cases where the additions cause extra costs.

First we show that immediate absorption preserves optimality for the EE problem. The following Lemma is the first step towards this result.

Lemma 3.4. *Consider a l-DAG G with two parallel edges $k_1, k_2 \in E(G)$. Let $G' = G - k_1 \parallel k_2$ then for every sequence of eliminations $T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}$, with $k_i, l_i \in E(G_i) \cup \{\emptyset\}$ for all $1 \leq i \leq n$ and $G_i = T_{k_i, l_i} \circ \dots \circ T_{k_1, l_1}(G)$ follows*

$$\sigma_G(T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1}) \geq \sigma_{G'}(T_{k_n, l_n} \circ \dots \circ T_{k_1, l_1})$$

Proof. From Theorem 3.2 follows that $G'_i \subseteq G_i$, where $G'_i = T_{k_i, l_i} \circ \dots \circ T_{k_1, l_1}(G)$. Hence $|P_{G'_i}(v)| \leq |P_{G_i}(v)|$ and $|S_{G'_i}(v)| \leq |S_{G_i}(v)|$ for all $v \in V(G)$. Hence

$$\sigma_{G'_i-1}(T_{k_i, l_i}) \leq \sigma_{G_i-1}(T_{k_i, l_i})$$

for all $1 \leq i \leq n$ and therefore the claim holds. $\square$

Theorem 3.6. *Consider a l-DAG G and two parallel edges $k_1, k_2 \in E(G)$. Let $G' = G - k_1 \parallel k_2$ then $EE(G') = EE(G)$.*

Proof. With $G' \subseteq G$ and Lemma 3.4 follows that $EE(G') \leq EE(G)$. From $G' = G - k_1 \parallel k_2$ follows that

$$EE(G) \leq EE(G - k_1 \parallel k_2) + \sigma_G(k_1 \parallel k_2) = EE(G') + 0 = EE(G') \quad .$$

Hence $EE(G') = EE(G)$. $\square$

This leads us to the result that immediate absorption preserves optimality in the context of the EE problem.

Theorem 3.7. *Immediate absorption preserves optimality in the context of the EE problem.*

Proof. The theorem follows trivially from Theorem 3.6. $\square$

We have proven that immediate absorption preserves optimality in the context of the EE problem. So now we have proven that splitting edge elimination and absorption cannot reduce the costs of the optimal edge elimination solution. Therefore from now on we will use the definition of front and back edge elimination as in Definition 2.2 and Definition 2.3.

In the following two lemmas we show that the costs of eliminating an edge or a vertex on a subgraph of G are never greater than the costs performing this elimination on G itself.

Lemma 3.5. *Consider an l-DAG $G' \subseteq G$. The following holds for any $(i, j) \in E^*(G)$*

$$\sigma_G((i, j)^+) \geq \sigma_{G'}((i, j)^+) \quad \text{and} \quad \sigma_G((i, j)^-) \geq \sigma_{G'}((i, j)^-) \quad .$$

Proof. It is sufficient to show the inequality for front elimination of an edge. The proof for back elimination follows due to symmetry.

Case 1. $(i, j) \notin E^*(G')$. In this case we do not have to front eliminate the edge (i, j) on G' , yielding $\sigma_{G'}((i, j)^+) = 0$. Hence the claim is trivially true in this case.

Case 2. $(i, j) \in E^*(G')$. From $G' \subseteq G$ follows that $|S_{G'}(j)| \leq |S_G(j)|$ and therefore

$$\sigma_G((i, j)^+) = |S_G(j)| \geq |S_{G'}(j)| = \sigma_{G'}((i, j)^+) \quad .$$

Hence the claim holds. $\square$

We can easily extend this result to elimination of a vertex.

Lemma 3.6. *Consider an l-DAG $G' \subseteq G$. The following holds for any $v \in Z(G)$,*

$$\sigma_G(v) \geq \sigma_{G'}(v) \quad .$$

Proof. The proof follows immediately from Lemma 3.5. $\square$

Now we have all results we need to extend Theorem 3.3 to the EE and VE problems.

Theorem 3.8. Consider l-DAGs G' and G , where $G' \subseteq G$.

$$EE(G') \leq EE(G) \quad .$$

Proof. This result follows immediately from Lemma 3.5 and from "subgraph monotonicity" (Theorem 3.2). $\square$

Theorem 3.9. Consider l-DAGs G' and G , where $G' \subseteq G$.

$$VE(G') \leq VE(G) \quad .$$

Proof. This result follows immediately from Lemma 3.6 and from "subgraph monotonicity" (Theorem 3.2). $\square$

3.2 Commutativity of edge and vertex eliminations

In this section we investigate in which cases the order of two subsequent eliminations does not influence the resulting l-DAG. We also refer to this property as *commutativity* or *interchangeability* of eliminations.

First research attempts on this topic were made in the bachelor thesis of Elmar Peise ([Pei09]) supervised as part of this thesis. In that bachelor thesis it was proved that the order of elimination of not adjacent vertices does not change the resulting l-DAG. We have also shown the criteria under which two subsequent edge eliminations can be permuted without changing the resulting l-DAG. Similar results were proved for face eliminations. These results were discussed in the context of reducing the search space for a branch and bound algorithm that was used to solve the VE, EE and FE problems respectively. The interchangeability of adjacent vertices and its effects on the branch and bound algorithm for VE problem was investigated in the master thesis of Tarek El Habbaj ([EH12]) that was also supervised during this thesis. A first discussion of these results in the context of MECV problem was published in [MN12].

This section is structured as follows. In Section 3.2.1 we prove that the order of vertex eliminations does not change the resulting graph. From all results introduced in this section this is the most used and therefore the most important one. In Section 3.2.2 the interchangeability of a vertex and an edge elimination is discussed. This result is important for the proofs of Theorem 4.1 and Theorem 4.4. The commutativity of two edge eliminations is investigated in Section 3.2.3. The commutativity of two edge eliminations is not used in any proof presented here. Instead it is only used to reduce the search space for the branch and bound algorithm. The exploitation of these results in the context of the MEC, MECV, EE and VE problems is described at the end of this section in Section 3.2.4 and Section 3.2.5.

3.2.1 Commutativity of two vertex eliminations

In this section we prove that the order of vertex eliminations does not have an impact on the resulting l-DAG no matter whether the vertices are adjacent or not (compare Lemma 3.7).

Lemma 3.7. Consider an l-DAG G . The following holds for all $a, b \in Z(G)$ and $a \neq b$

$$G_1 = G - [a, b] = G_2 = G - [b, a] \quad .$$

Proof. Both G_1 and G_2 contain all vertices of G except of a and b . Hence $V(G_1) = V(G_2) = V \setminus \{a, b\}$. It remains to show that $E(G_1) = E(G_2)$.

" $\subseteq$ " We are going to show: $(i, j) \in E(G_1) \Rightarrow (i, j) \in E(G_2)$. Therefore we examine two cases. In the first case G already contains the edge (i, j) , in the second case the edge is introduced by the eliminations.

Case 1. $(i, j) \in E(G)$. From the fact that $(i, j) \in E(G_1)$ follows that

$$i \neq a \wedge i \neq b \wedge j \neq a \wedge j \neq b \quad .$$

Hence neither a nor b are incident to (i, j) . Therefore $(i, j) \in E(G_2)$.

Case 2. $(i, j) \notin E(G)$. This case is divided in two subcases:

- a and b are not adjacent
- a and b are adjacent.

Case 2.1. a and b are not adjacent. From $(i, j) \in E(G_1)$ and $(i, j) \notin E(G)$ follows that,

$$\text{either } (i, j) \in E(G - a) \quad \text{or} \quad i \in P_{G-a}(b) \wedge j \in S_{G-a}(b) \quad .$$

If $(i, j) \in E(G - a)$, then we know that $i \in P_G(a)$ and $j \in S_G(a)$. Both i and j are not equal to b , hence $i \in P_{G-b}(a)$ and $j \in S_{G-b}(a)$. Therefore $(i, j) \in E(G_2)$.

Consider the case $i \in P_{G-a}(b) \wedge j \in S_{G-a}(b)$. As a and b are not adjacent, $i \in P_G(b)$ and $j \in S_G(b)$. Therefore $(i, j) \in E(G - b)$. Due to the fact that both i and j are not equal to a , $(i, j) \in E(G_2)$.

Case 2.2. a and b are adjacent. From $(i, j) \in E(G_1)$ and $(i, j) \notin E(G)$ follows that,

$$\text{either } (i, j) \in E(G - a) \quad \text{or} \quad i \in P_{G-a}(b) \wedge j \in S_{G-a}(b) \quad .$$

If $(i, j) \in E(G - a)$, then we know that $i \in P_G(a)$ and $j \in S_G(a)$. Both i and j are not equal to b , hence $i \in P_{G-b}(a)$ and $j \in S_{G-b}(a)$. Therefore $(i, j) \in E(G_2)$.

Now let's take a look at the case that $i \in P_{G-a}(b) \wedge j \in S_{G-a}(b)$. If $i \in P_G(b)$ and $j \in S_G(b)$, then $(i, j) \in E(G - b)$. As both i and j are not equal to a , $(i, j) \in E(G_2)$.

In case that either $i \notin P_G(b)$ or $j \notin S_G(b)$, we first assume that $(a, b) \in E(G)$. Therefore $S_{G-a}(b) = S_G(b)$. Hence $j \in S_G(b)$ and therefore $i \notin P_G(b)$ due to the assumption. As $i \in P_{G-a}(b)$ and $i \notin P_G(b)$, we know that $i \in P_G(a)$ and therefore $i \in P_{G-b}(a)$. Additionally from $j \in S_G(b)$ and $(a, b) \in E(G)$ follows that $j \in S_{G-b}(a)$. Hence $(i, j) \in E(G_2)$.

Now we assume that $(b, a) \in E(G)$. Therefore $P_{G-a}(b) = P_G(b)$. Hence $i \in P_G(b)$ and therefore $j \notin S_G(b)$ due to the assumption. As $j \in S_{G-a}(b)$ and $j \notin S_G(b)$, we know that $j \in S_G(a)$. Additionally from $i \in P_G(b)$ and $(b, a) \in E(G)$ follows that $i \in P_{G-b}(a)$. Hence $(i, j) \in E(G_2)$.

" $\supseteq$ " The proof is analogous due to symmetry. □

Let us discuss this result on the following example.

Example 3.2. Consider the l-DAG G in Figure 3.5(a). Elimination of vertex v_1 yields the graph $G_1 = G - v_1$ presented in Figure 3.5(b). Removing the vertex v_2 from G_1 results in the l-DAG presented on Figure 3.5(c). Performing the eliminations in reverse order yield the same result. For comparison the result of eliminating vertex v_2 from G can be found in Figure 3.5(d).

3.2.2 Commutativity of a vertex and edge elimination

In this section we discuss under which constraints the order of a vertex and an edge elimination can be switched without changing the resulting l-DAG. We use the results of this section only in proofs (e.g. Theorem 4.4 and Theorem 4.1). None of the algorithms presented later makes use of them. Unlike the interchangeability of two vertex eliminations, there are cases when the order of a vertex and edge elimination does change the resulting graph. This can happen for example if the eliminated vertex is

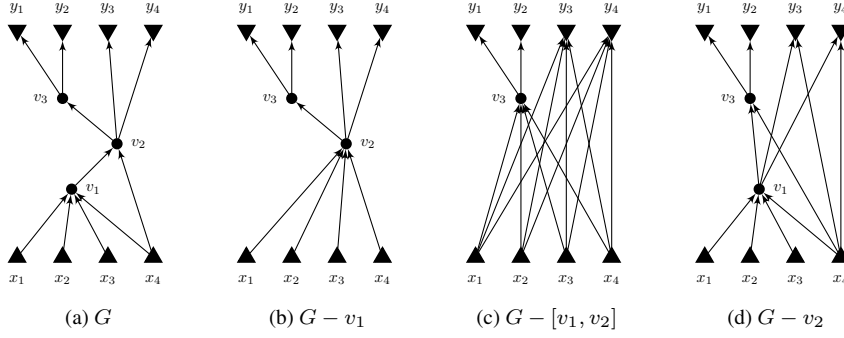


Figure 3.5: Example of commutativity of two vertex eliminations

source or target of the edge. In this case we cannot perform the edge elimination after the vertex has been removed as the edge does not exist in the resulting graph. The resulting l-DAG also changes, if the eliminated vertex is a successor of the source and a predecessor of the target of the eliminated edge. This situation is described in Example 3.4. The next two lemmas prove that in all other cases edge and vertex elimination can be interchanged without changing the resulting l-DAG. First we prove this result for front elimination of an edge.

Lemma 3.8. *Consider an l-DAG G . The following holds for all $(v_1, v_2) \in E(G)$, $a \in Z(G)$, where $v_1 \neq a$, $v_2 \neq a$ and $a \notin S_G(v_1) \wedge a \notin P_G(v_2)$:*

$$G - [(v_1, v_2)^+, a] = G - [a, (v_1, v_2)^+] \quad .$$

Proof. The case $|P_G(v_2)| = 1$ is covered by the Lemma 3.7. Hence we assume that $|P_G(v_2)| \geq 2$. For the sake of notational brevity we set

$$G_1 = G - [(v_1, v_2)^+, a] \text{ and } G_2 = G - [a, (v_1, v_2)^+] \quad .$$

To show the equality of G_1 and G_2 , we first show that their vertex sets are equal. $V(G - (v_1, v_2)^+) = V(G)$, as $|P_G(v_2)| \geq 2$ and hence according to the definition of vertex elimination $V(G_1) = V(G) \setminus \{a\}$. Moreover $|P_{G-a}(v_2)| \geq 2$ as $v_1 \neq a$, $v_2 \neq a$ and therefore $V(G_2) = V(G - a) = V(G) \setminus \{a\}$. Hence $V(G_1) = V(G_2)$.

Next we show that $E(G_1) = E(G_2)$.

" $\subseteq$ " We show $(i, j) \in E(G_1) \Rightarrow (i, j) \in E(G_2)$ by considering three cases. The first case is where G already contains the edge (i, j) , in the second case (i, j) is introduced by the edge elimination and in the third case by vertex elimination.

Case 1. $(i, j) \in E(G)$. $(i, j) \neq (v_1, v_2)$ as otherwise $v_1 \in P_G(a)$ and $v_2 \in S_G(a)$ which contradicts the assumption. Therefore $(i, j) \in E(G - (v_1, v_2)^+)$ yielding $i \neq a$ and $j \neq a$ as otherwise $(i, j) \notin E(G_1)$. It follows that $(i, j) \in E(G - a)$. Furthermore $(i, j) \neq (v_1, v_2)$ and therefore $(i, j) \in E(G_2)$.

Case 2. $(i, j) \notin E(G)$ and $(i, j) \in E(G - (v_1, v_2)^+)$. From the fact that (i, j) is introduced by the front elimination of the edge (v_1, v_2) follows that $i = v_1 \wedge j \in S_G(v_2)$. Furthermore (i, j) is not incident with a as otherwise $(i, j) \notin E(G_1)$ and hence $j \in S_{G-a}(v_2)$ implying that $(i, j) \in E(G_2)$.

Case 3. $(i, j) \notin E(G)$ and $(i, j) \notin E(G - (v_1, v_2)^+)$ meaning that $i \in P_{G-(v_1, v_2)^+}(a)$ and $j \in S_{G-(v_1, v_2)^+}(a)$. According to the definition of front edge elimination the only vertex whose successor set is changed during the front elimination of (v_1, v_2) is v_1 . Therefore $j \in S_G(a)$ as $v_1 \neq a$.

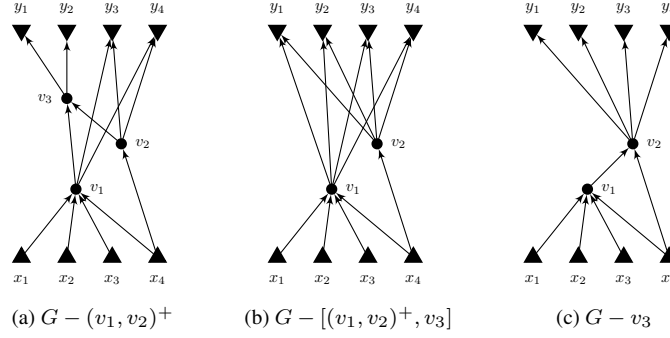


Figure 3.6: Example of commutativity of vertex and edge elimination

Case 3.1. If $i \in P_G(a)$, then $(i, j) \in E(G - a)$ and therefore $(i, j) \in E(G_2)$ as $(i, j) \neq (v_1, v_2)$ due to the assumption that $(i, j) \notin E(G)$.

Case 3.2. If $i \notin P_G(a)$, then $i = v_1$ and $a \in S_G(v_2)$ as otherwise $i \notin P_{G-(v_1, v_2)^+}(a)$. Hence $j \in S_{G-a}(v_2)$ implying that $(i, j) \in E(G_2)$.

” $\supseteq$ ” We show $(i, j) \in E(G_2) \Rightarrow (i, j) \in E(G_1)$ by considering again three cases.

Case 1. $(i, j) \in E(G)$. From $(i, j) \in E(G_2)$ follows that (i, j) is not incident with a . Therefore $(i, j) \in E(G - a)$ and hence $(i, j) \neq (v_1, v_2)$ as otherwise $(i, j) \notin E(G_2)$. Hence $(i, j) \in E(G - (v_1, v_2)^+)$. Moreover $(i, j) \in E(G_1)$ as (i, j) is not incident with a .

Case 2. $(i, j) \notin E(G)$ and $(i, j) \in E(G - a)$. Hence $i \in P_G(a) \wedge j \in S_G(a)$. According to the definition of front elimination $S_{G-(v_1, v_2)^+}(a) = S_G(a)$ as $a \neq v_1$. From the assumption that $a \neq v_2$ follows that $(i, a) \neq (v_1, v_2)$ implying $i \in P_{G-(v_1, v_2)^+}(a)$ and hence $(i, j) \in E(G_1)$.

Case 3. $(i, j) \notin E(G)$ and $(i, j) \notin E(G - a)$. Hence $i = v_1$ and $j \in S_{G-a}(v_2)$.

Case 3.1. $j \in S_G(v_2)$. In this case $(i, j) \in E(G - (v_1, v_2)^+)$. Vertex a is not incident with (i, j) because otherwise $(i, j) \notin E(G_2)$ and therefore $(i, j) \in E(G_1)$.

Case 3.2. $j \notin S_G(v_2)$. (v_2, j) is introduced by the vertex elimination of a in G . Hence $v_2 \in P_G(a)$ and $j \in S_G(a)$. Therefore, $i = v_1 \in P_{G-(v_1, v_2)^+}(a)$ implying that $(i, j) \in E(G_1)$.

□

A corresponding result for back edge elimination can be proved similarly.

Lemma 3.9. Consider an l-DAG G . The following holds for all $(v_1, v_2) \in E(G)$, $a \in Z(G)$, where $v_1 \neq a$, $v_2 \neq a$ and $a \notin S_G(v_1) \wedge a \notin P_G(v_2)$:

$$G - [(v_1, v_2)^-, a] = G - [a, (v_1, v_2)^-] \quad .$$

Proof. The proof is analogous to that of Lemma 3.8. □

Let us now discuss the results of Lemma 3.8 on the following example.

Example 3.3. Consider an l-DAG G as in Figure 3.5(a). It is easy to see that $G - [(v_1, v_2)^+, v_3] = G - [v_3, (v_1, v_2)^+]$ (compare Figure 3.6).

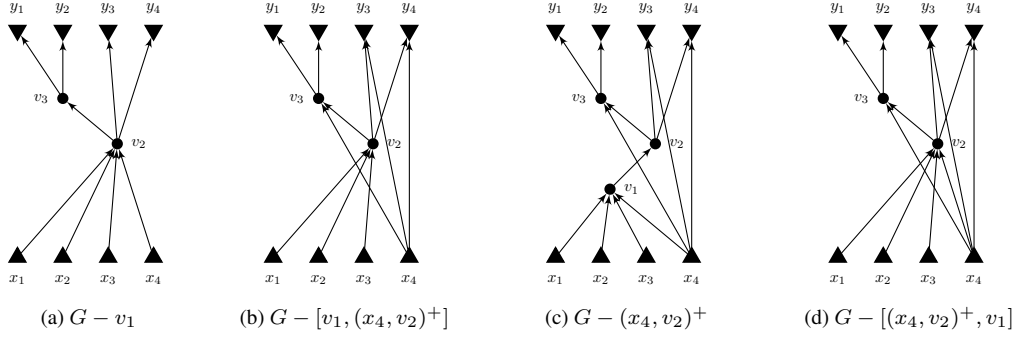


Figure 3.7: Requirements for commutativity of vertex and edge elimination (illustrating Example 3.4)

As discussed at the beginning of this section, the prerequisites of Lemma 3.8 [Lemma 3.9] are essential. If $a = v_1$ or $a = v_2$ we simply cannot perform the front [back] elimination of (v_1, v_2) after the elimination of vertex a as $(v_1, v_2) \notin E(G - a)$. The reason for the second prerequisite is not that obvious. The following example demonstrates its necessity.

Example 3.4. Consider an l-DAG G as in Figure 3.5(a). After elimination of vertex v_1 followed by front elimination of edge (x_4, v_2) the resulting graph $G - [v_1, (x_4, v_2)^+]$ does not contain the edge (x_4, v_2) , while as $(x_4, v_2) \in E(G - [(x_4, v_2)^+, v_1])$ (compare Figure 3.7).

3.2.3 Commutativity of two edge eliminations

The constraints under which the order of two edge eliminations can be switched without changing the resulting l-DAG are even more complicated than those from the Section 3.2.2. The reason for this is the fact that a vertex elimination can be expressed by a sequence of edge eliminations. Therefore Example 3.4 also illustrates that switching the order of edge eliminations can change the resulting l-DAG (elimination of vertex v_1 can also be performed by back elimination of (v_1, v_2)). This fact makes it hard to exploit the commutativity of edge elimination in proofs. Still we could achieve good results using this property to reduce the search space within the branch and bound algorithm presented in Section 6.2.2.

The next three lemmas prove in which cases two subsequent edge eliminations yield the same remainder graph. The proofs are not very complicated in the sense that you do not need to find any specific tricks. You simply compare the corresponding edge and vertex sets. Cross product as defined in standard graph theory is used in the proofs.

Definition 3.8. Consider l-DAG G . For two subsets of vertices $L, M \subseteq V(G)$ we set

$$L \times M = \{(i, j) \mid i \in L \text{ and } j \in M\} \quad .$$

Lemma 3.10. The following holds for every l-DAG G , $(a, b) \neq (c, d) \in E(G)$, where $c \not\prec^* a$ and $b \neq c$

$$(c, d) \in E(G - (a, b)^+) \text{ and } (a, b) \in E(G - (c, d)^+) \quad .$$

If additionally $a \neq c$ or $(b, d), (d, b) \notin E(G)$, then

$$G - [(a, b)^+, (c, d)^+] = G - [(c, d)^+, (a, b)^+] \quad .$$

Proof. We set $G_1 = G - (a, b)^+$, $G_2 = G - (c, d)^+$, $G'_1 = G_1 - (c, d)^+$ and $G'_2 = G_2 - (a, b)^+$. First we proof the first part of the claim. Assume that $(c, d) \notin E(G_1)$. Hence $(c, d) \in E(G)$ is incident with b and b is removed from G during the front elimination of (a, b) . The fact that b is removed from G implies that a is the only predecessor of b . Therefore there exist two cases where the conclusion that b is incident with (c, d) is satisfied, $b = c$ or $(a, b) = (c, d)$. Both cases contradict the assumptions of the lemma. The proof that $(a, b) \in E(G - (c, d)^+)$ is analogous.

Let us now proof the second part of the lemma. First we show the equality of the vertex sets $V(G'_1) = V(G'_2)$. According to the definition of front elimination it is sufficient to show that

$$d \notin V(G'_1) \Leftrightarrow d \notin V(G'_2) \quad \text{and} \quad b \notin V(G'_1) \Leftrightarrow b \notin V(G'_2) \quad .$$

Case 1. $b = d$. In this case d is removed from G if and only if $P_G(d) = \{a, c\}$, and therefore the elimination order does not make any difference.

Case 2. $b \neq d$. We show $d \notin V(G'_1) \Leftrightarrow d \notin V(G'_2)$ for this case. Assume that $d \notin V(G'_1)$ implying that $P_{G_1}(d) = \{c\}$ and therefore from $b \neq d$ follows that $P_G(d) = \{c\}$. Hence $d \notin V(G_2)$ implying $d \notin V(G'_2)$. Assume that $d \notin V(G'_2)$. Therefore $d \notin V(G_2)$ as otherwise $b = d$. Hence $P_G(d) = \{c\}$ implying $d \notin V(G'_1)$. The proof that $b \notin V(G'_1) \Leftrightarrow b \notin V(G'_2)$ is analogous.

Now we show that the corresponding edge sets are equal. We assume that none of the vertices are removed during the eliminations. First we need some intermediate results. From the assumption $a \neq c \vee (b, d), (d, b) \notin E(G)$ follows that

$$(c, d) \notin \{a\} \times S_G(b) \quad (3.2)$$

and

$$(a, b) \notin \{c\} \times S_G(d) \quad (3.3)$$

$a \neq d$ as otherwise $(c, a) \in E(G)$ leading to a contradiction to $c \not\prec^* a$. Hence the successor set of d in G is not changed by the front elimination of (a, b) yielding

$$S_G(d) = S_{G_1}(d) \quad (3.4)$$

Analogously as $b \neq c$

$$S_G(b) = S_{G_2}(b) \quad (3.5)$$

Hence the following equality holds

$$\begin{aligned} E(G'_1) &= (E(G_1) \setminus \{(c, d)\}) \cup (\{c\} \times S_{G_1}(d)) \\ &= \left[\left((E(G) \setminus \{(a, b)\}) \cup (\{a\} \times S_G(b)) \right) \setminus \{(c, d)\} \right] \cup (\{c\} \times S_{G_1}(d)) \\ &\stackrel{\text{Eq(3.2), Eq(3.4)}}{=} \left(E(G) \setminus \{(a, b), (c, d)\} \right) \cup \left((\{a\} \times S_G(b)) \cup (\{c\} \times S_G(d)) \right) \\ &\stackrel{\text{Eq(3.3), Eq(3.5)}}{=} \left[\left((E(G) \setminus \{(c, d)\}) \cup (\{c\} \times S_G(d)) \right) \setminus \{(a, b)\} \right] \cup (\{a\} \times S_{G_2}(b)) \\ &= (E(G_2) \setminus \{(a, b)\}) \cup (\{a\} \times S_{G_2}(b)) \\ &= E(G'_2) \quad . \end{aligned}$$

Let us now look at the case where the edge eliminations cause a removal of a vertex. We have shown that the vertex sets of G'_1 and G'_2 are equal. Hence due to the fact that the removal of a vertex yields the deletion of all incident edges the resulting l-DAGs are still equal after the removal of the vertices. $\square$

Lemma 3.11. *The following holds for every l-DAG G , $(a, b) \neq (c, d) \in E(G)$, where $c \not\prec^* a$ and $a \neq c$:*

$$(c, d) \in E(G - (a, b)^-) \text{ and } (a, b) \in E(G - (c, d)^-) \quad .$$

If additionally $a \neq c$ or $(b, d), (d, b) \notin E(G)$, then

$$G - [(a, b)^-, (c, d)^-] = G - [(c, d)^-, (a, b)^-] \quad .$$

Proof. The proof is analogous to Lemma 3.10 due to symmetry. □

Lemma 3.12. *The following holds for every l-DAG G , $(a, b) \neq (c, d) \in E(G)$, where $b \neq c$:*

$$(c, d) \in E(G - (a, b)^+) \text{ and } (a, b) \in E(G - (c, d)^-)$$

If additionally $(a \neq c \vee (b, d) \notin E(G)) \wedge (b \neq d \vee (a, c) \notin E(G))$, then

$$G - [(a, b)^+, (c, d)^-] = G - [(c, d)^-, (a, b)^+] \quad .$$

Proof. The proof is similar to Lemma 3.10 □

In the following example we demonstrate the application of Lemma 3.10.

Example 3.5. *Consider l-DAG G as on Figure 3.5(a). Front elimination of (x_4, v_1) followed by front elimination of (x_3, v_1) yields the same l-DAG as if we first front eliminated the edge (x_3, v_1) followed by front elimination of (x_4, v_1) (compare Figure 3.8).*

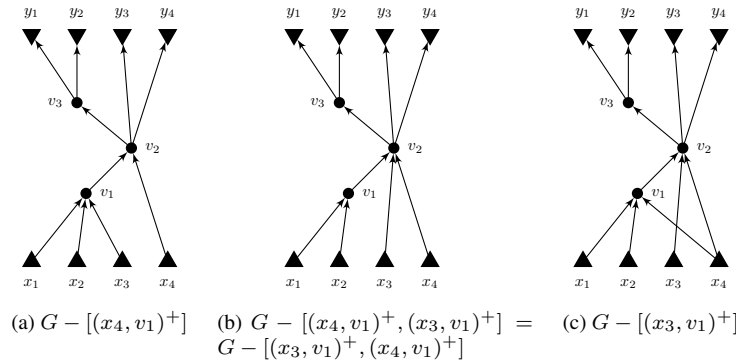


Figure 3.8: Commutativity of edge eliminations according to Lemma 3.10
(illustrating Example 3.5)

We avoid to present an example for Lemma 3.11 as this result is symmetrical to Lemma 3.10. The example for commutativity of edge eliminations as in Lemma 3.12 is presented in Example 3.10.

Example 3.6. *Consider the l-DAG G in Figure 3.5(a). Front elimination of (x_4, v_1) followed by back elimination of (v_3, y_2) yields the same l-DAG as if we first back eliminated the edge (v_3, y_2) followed by front elimination of (x_4, v_1) (compare Figure 3.9).*

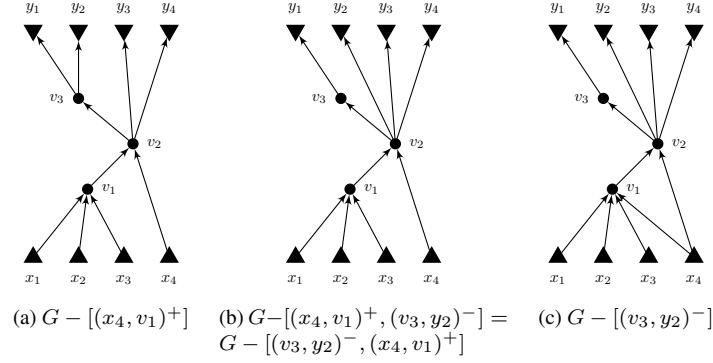


Figure 3.9: Commutativity of edge eliminations according to Lemma 3.12 (illustrating Example 3.6)

3.2.4 Consequences for the minimal edge count problem

In this section the direct implications for the MEC problem from the results of the Section 3.2.1 and Section 3.2.3 are discussed. The results from these sections can be directly applied to the MEC problem.

Vertex elimination

While solving the MEC problem by vertex elimination for a given l-DAG G there are

$$\sum_{k=1}^{|Z(G)|} k! \binom{|Z(G)|}{k}$$

different vertex elimination sequences to be compared. By default we need to check all permutations of each subset of $Z(G)$. Fortunately from Lemma 3.7 we know that permutations of vertex eliminations do not change the resulting graph. Hence we only have to compare all subsets of $Z(G)$ implying that the search space is reduced to

$$\sum_{k=0}^{|Z(G)|} \binom{|Z(G)|}{k} = 2^{|Z(G)|}$$

different vertex elimination sequences. Interestingly, without Lemma 3.7 one would expect that the search space of the MECV problem to be bigger than that of the VE problem. As in VE we need to check all permutations of $Z(G)$. Hence there are $|Z(G)|!$ different vertex elimination sequences that have to be compared. Lemma 3.7 tells us that this is not the case.

Edge elimination

As already mentioned, the result that two edge eliminations are commutable, is used inside a branch and bound algorithm (Algorithm 1) to reduce the search space of the MEC problem. The actual application of this result inside the branch and bound is shown in Algorithm 17. Due to the fact that new edges are created during the edge elimination sequence, it is more difficult to quantify the number of different edge elimination sequences that build the search space for the MEC problem. We currently do not have any theoretical results for this. A significant problem is the fact that the order of two edge eliminations cannot always be permuted without changing the resulting l-DAG. Therefore it is hard to predict, how big

the gain from exploiting the commutativity of edge eliminations is, as it depends on the actual l-DAG. On smaller l-DAGs, where a lot of edges are adjacent, we can avoid checking only small number of edge elimination sequences. For example in Example 5.2 we see that we avoid checking only 3 edge elimination sequences from an overall number of 36 different edge elimination sequences. The bigger the l-DAG is, the more advantage we draw from exploiting the commutativity of edge eliminations. For example exploiting the commutativity of edge eliminations on the lion graph when solving the EE problem can reduce the search space by a factor of 6 (Table 6.2). Although for this problem we also have to ensure that the costs after switching the edge eliminations are equal. So the search space reduction should be even bigger for solving the MEC problem. On substantially big l-DAGs we should see a similar behaviour as from exploiting the commutativity of vertex eliminations.

3.2.5 Consequences for the optimal edge and vertex elimination problems

In this section the implications from the results of the Section 3.2.1 and Section 3.2.3 to the EE and VE problems are discussed. In contrast to the MEC problem the results cannot be directly applied to the EE and VE problems. First we need additional results about how the permutation of the vertex (edge) eliminations effects the costs of the elimination sequence.

Vertex elimination

Although the resulting l-DAG is the same, no matter in which order the vertex eliminations are performed, the corresponding costs can be different as shown in the following example.

Example 3.7. Consider the l-DAG G in Figure 3.5(a). The costs of eliminating v_1 followed elimination of v_2 are

$$\sigma_G([v_1, v_2]) = 4 + 12 = 16 \quad ,$$

while eliminating v_2 followed by v_1 yields

$$\sigma_G([v_2, v_1]) = 6 + 12 = 18 \quad .$$

Obviously if the vertices are not adjacent, the order of the vertex eliminations does not change the costs of the elimination sequence. Let us prove this result.

Lemma 3.13. Consider an l-DAG G . The following holds for any two not adjacent vertices $a, b \in Z(G)$:

$$\sigma_G([a, b]) = \sigma_G([b, a]) \quad .$$

Proof. a and b are not adjacent. Therefore the predecessor and successor sets of a are not changed by the elimination of b . Similarly the elimination of b has no effect on the predecessor and successor sets of a . Hence $\sigma_{G-a}(b) = \sigma_G(b)$ and $\sigma_{G-b}(a) = \sigma_G(a)$. Yielding

$$\sigma_G([a, b]) = \sigma_G(a) + \sigma_{G-a}(b) = \sigma_G(a) + \sigma_G(b) = \sigma_G(b) + \sigma_{G-b}(a) = \sigma_G([b, a]) \quad .$$

□

The following examples demonstrate this result.

Example 3.8. Consider the l-DAG G in Figure 3.5(a). The costs of eliminating v_1 followed by elimination of v_3 are the same as performing the elimination in reverse order.

$$\sigma_G([v_1, v_3]) = 4 + 2 = 6 = 2 + 4 = \sigma_G([v_3, v_1]) \quad .$$

Although the proof of Lemma 3.13 is almost obvious, this result is crucial for the proofs that vertex elimination solves the MEC and EE problems on absorption-free l-DAGs (see Chapter 4). It also allows to reduce the search space for solving the VE problem with branch and bound significantly (see Section 6.2 for more details). To solve the VE problem for a given l-DAG G we have to check all possible complete vertex elimination sequences on this graph. This number is equal to the number of all permutations of the intermediate vertices of G . Hence the search space for the VE problem on G consists of $|Z(G)|!$ different complete vertex elimination sequences. Exploiting the commutativity of vertex eliminations and the result from Lemma 3.13, we do not have to check all possibilities. We can rather stop exploring continuations of some elimination sequences. For example in Example 6.1 we do not test the elimination sequence $[v_3, v_1]$, as we know that the costs will be the same as for elimination sequences starting with $[v_1, v_3]$. As the costs remain the same only in case that the vertices are not adjacent, we cannot predict how big the benefit will be. Hence on small l-DAGs, where a significant amount of vertices are adjacent, the benefit of exploiting commutativity will be smaller than on bigger l-DAGs, where the percentage of adjacent vertices is small. For example in Example 6.1 we see that on an l-DAG with only 3 vertices we can avoid checking only one elimination sequence from an overall number of 6 different vertex elimination sequences. On bigger l-DAGs we can expect greater savings in terms of search space reduction. For example on an l-DAG with 10 intermediate vertices we can reduce the number of checked vertex elimination sequences by a factor of 230 (compare Table 6.1, Figure 6.7(a)).

The next lemma allows us to compute the cost of a vertex elimination sequence, where the order of two subsequent vertex eliminations with adjacent vertices was changed. This result allows us to reduce the search space for the branch and bound algorithm even further.

Lemma 3.14. *Consider an l-DAG G . For any vertex elimination sequence $[v_1, \dots, v_m]$ on G holds:*

$$\sigma_G([v_1, \dots, v_m]) + \sigma_{G_{i-1}}([v_{i+1}, v_i]) = \sigma_G([v_1, \dots, v_{i-1}, v_{i+1}, v_i, \dots, v_m]) + \sigma_{G_{i-1}}([v_i, v_{i+1}]) \quad ,$$

where $G_i = G - [v_1, \dots, v_i]$ and $1 \leq i \leq m - 1$.

Proof. We set $G' = G_{i-1} - [v_{i+1}, v_i]$. From Definition 2.4 follows

$$\sigma_G([v_1, \dots, v_m]) = \sigma_G([v_1, \dots, v_{i-1}]) + \sigma_{G_{i-1}}([v_i, v_{i+1}]) + \sigma_{G_{i+1}}([v_{i+2}, \dots, v_m]) \quad (3.6)$$

and

$$\begin{aligned} \sigma([v_1, \dots, v_{i-1}, v_{i+1}, v_i, \dots, v_m]) &= \sigma([v_1, \dots, v_{i-1}]) \\ &\quad + \sigma_{G_{i-1}}([v_{i+1}, v_i]) + \sigma_{G'}([v_{i+2}, \dots, v_m]) \quad . \end{aligned} \quad (3.7)$$

From Lemma 3.7 follows that $G_{i+1} = G'$. Hence subtraction of Equation (3.6) from Equation (3.7) yields

$$\sigma_G([v_1, \dots, v_m]) - \sigma_G([v_1, \dots, v_{i-1}, v_{i+1}, v_i, \dots, v_m]) = \sigma_{G_{i-1}}([v_i, v_{i+1}]) - \sigma_{G_{i-1}}([v_{i+1}, v_i])$$

finishing the proof. $\square$

Let us discuss how this result can be used to reduce the search space on an example.

Example 3.9. *Consider the l-DAG G in Figure 3.5(a). We will show that the vertex elimination sequence $[v_2, v_1, v_3]$ cannot be optimal without actually performing the whole vertex elimination sequence. From Example 3.7 we know that*

$$\sigma_G([v_1, v_2]) = 4 + 12 = 16 \quad \text{and} \quad \sigma_G([v_2, v_1]) = 6 + 12 = 18 \quad .$$

With Lemma 3.14 we get

$$\begin{aligned}\sigma_G([v_2, v_1, v_3]) &= \sigma_G([v_1, v_2, v_3]) + \sigma_G([v_2, v_1]) - \sigma_G([v_1, v_2]) \\ &= \sigma_G([v_1, v_2, v_3]) - 16 + 18 \\ &= \sigma_G([v_1, v_2, v_3]) + 2 \quad .\end{aligned}$$

Hence the vertex elimination sequence $[v_2, v_1, v_3]$ needs 2 fmas more than $[v_1, v_2, v_3]$ and therefore it cannot be an optimal elimination sequence.

As already mentioned this result allows further reduction of the search space of the VE problem. For example in Example 6.2 we do only check 4 vertex elimination sequences instead of originally 6. Moreover only two of these elimination sequences are complete, yielding even bigger savings in computational time. On l-DAGs with more intermediate vertices the savings are even more significant (see Table 6.1). Please see Section 6.2 for more details on the practical application of this result.

Edge elimination

Similar to vertex elimination, the order of edge eliminations can have influence on the cost of the edge elimination sequence, even though the resulting l-DAG is the same. The following three lemmas proof under which constraints the costs remain the same. These results are used in Section 6.2 to reduce the search space for the branch and bound algorithm when applied to the EE problem.

Example 3.10. Let G be the lion graph. Front elimination of (x_1, v_1) followed by back elimination of (v_2, y_1) yields costs of 4 fmas (compare Figure 3.10(a) and Figure 3.10(b)). Performing the same eliminations in reverse order (back elimination of (v_2, y_1) followed by front elimination of (x_1, v_1)) yields costs of 3 fmas (compare Figure 3.10(c) and Figure 3.10(b)) although the resulting l-DAG is the same.

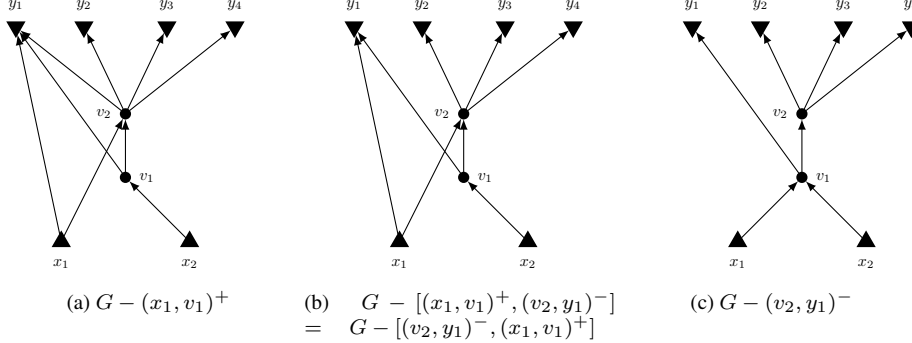


Figure 3.10: The order of edge eliminations can change the costs although the resulting l-DAG is the same (illustrating Example 3.10)

First we examine the difference in costs between two front edge eliminations.

Lemma 3.15. Consider l-DAG G , $(a, b), (c, d) \in E^*(G)$. If $b \neq c$ and $a \neq d$, then

$$\sigma_G([(a, b)^+, (c, d)^+]) = \sigma_G([(c, d)^+, (a, b)^+]) \quad .$$

Proof. We set $G_1 = G - (a, b)^+$ and $G_2 = G - (c, d)^+$. According to the definition of front edge elimination

$$\sigma_G((a, b)^+) = |S_G(b)| \quad \text{and} \quad \sigma_G((c, d)^+) = |S_G(d)| \quad .$$

From $b \neq c$ follows that $c, d \in V(G_1)$. As $a \neq d$ the front elimination of (a, b) does not change the successor set of d and hence $S_{G_1}(d) = S_G(d)$. Similarly from $d \neq a$ follows that $a, b \in V(G_2)$ and from $b \neq c$ follows $S_{G_2}(b) = S_G(b)$. Hence

$$\begin{aligned}\sigma_G([(a, b)^+, (c, d)^+]) &= |S_G(b)| + |S_{G_1}(d)| = |S_G(b)| + |S_G(d)| \\ &= |S_G(d)| + |S_{G_2}(b)| = \sigma_G([(c, d)^+, (a, b)^+]) \quad .\end{aligned}$$

□

The proof for two back edge eliminations is symmetrical.

Lemma 3.16. Consider l-DAG G , $(a, b), (c, d) \in E^*(G)$. If $b \neq c$ and $a \neq d$ then

$$\sigma_G([(a, b)^-, (c, d)^-]) = \sigma_G([(c, d)^-, (a, b)^-]) \quad .$$

Proof. The proof is analogous to the proof of Lemma 3.15 due to symmetry. □

Lemma 3.17. Consider an l-DAG G , $(a, b), (c, d) \in E^*(G)$. If $(b, c) \notin E^*(G)$, $(a, b) \notin E^*(G - (c, d)^-)$ and $(c, d) \notin E^*(G - (a, b)^+)$, then

$$\sigma_G([(a, b)^+, (c, d)^-]) = \sigma_G([(c, d)^-, (a, b)^+]) \quad .$$

Proof. We set $G_1 = G - (a, b)^+$ and $G_2 = G - (c, d)^-$.

Case 1. $b = c$. From the definition of front edge elimination follows that $P_{G_1}(c) = P_G(c) \setminus \{a\}$. Moreover from the definition of back edge elimination follows $S_{G_2}(b) = S_G(b) \setminus \{d\}$. For the costs we get

$$\begin{aligned}\sigma_G([(a, b)^+, (c, d)^-]) &= |S_G(b)| + |P_{G_1}(c)| = |S_G(b)| + |P_G(c)| - 1 \\ &= |S_{G_2}(b)| + |P_G(c)| = \sigma_G([(c, d)^-, (a, b)^+]) \quad ,\end{aligned}$$

finishing the proof for this case.

Case 2. $b \neq c$. According to the assumption $(b, c) \notin E(G)$ and therefore the predecessor set of c is not changed by the front elimination of the edge (a, b) and hence $P_G(c) = P_{G_1}(c)$. Similarly as $(b, c) \notin E(G)$ the successor set of b is not changed by the back elimination of the edge (c, d) yielding $S_G(b) = S_{G_2}(b)$. Hence

$$\begin{aligned}\sigma_G([(a, b)^+, (c, d)^-]) &= |S_G(b)| + |P_{G_1}(c)| = |S_G(b)| + |P_G(c)| \\ &= |S_{G_2}(b)| + |P_G(c)| = \sigma_G([(c, d)^-, (a, b)^+]) \quad ,\end{aligned}$$

finishing the proof. □

The next example demonstrates the application of Lemma 3.15.

Example 3.11. Consider the l-DAG G as in Figure 3.5(a). The order of performing the front elimination of the edges (x_4, v_1) and (x_3, v_1) does not influence the resulting costs as

$$\sigma_G([(x_4, v_1)^+, (x_3, v_1)^+]) = 1 + 1 = 2 = 1 + 1 = \sigma_G([(x_3, v_1)^+, (x_4, v_1)^+]) \quad ,$$

compare Figure 3.8.

The next example deals with the application of Lemma 3.17.

Example 3.12. Consider the l-DAG G in Figure 3.5(a). The edge elimination sequences

$$[(x_4, v_1)^+, (v_3, y_2)^-] \quad \text{and} \quad [(v_3, y_2)^-, (x_4, v_1)^+]$$

yield the same costs as

$$\sigma_G([(x_4, v_1)^+, (v_3, y_2)^-]) = 1 + 1 = 2 = 1 + 1 = \sigma_G([(v_3, y_2)^-, (x_4, v_1)^+]) \quad ,$$

compare Figure 3.9.

We avoid to demonstrate an example for the application of Lemma 3.16 as this result is symmetrical to Lemma 3.15.

It is difficult to quantify the number of edge elimination sequences that build the search space for the EE problem, as new edges are created during an edge elimination sequence. As already mentioned, we do not have any theoretical results for this. A significant problem is the fact that the order of two edge eliminations cannot always be permuted without changing the resulting l-DAG or the resulting costs. Therefore it is hard to predict, how big the gain from exploiting the commutativity of edge eliminations is, as it depends on the actual l-DAG. On smaller l-DAGs, where a lot of edges are adjacent, we can avoid checking only small number of edge elimination sequences. For example in Example 6.3 we see that we avoid checking only 2 edge elimination sequences from an overall number of 22 different edge elimination sequences. As graphs grow, we draw more advantage from exploiting the commutativity of edge eliminations. For example exploiting the commutativity of edge eliminations on the lion graph when solving the EE problem can reduce the search space by a factor of 9. More details about the practical application of this result are given in Section 6.2.2.

3.3 Summary

In this chapter we have proved two properties "subgraph monotonicity" and the commutativity of vertex and edge elimination. The most important implication from the "subgraph monotonicity" for this thesis is the fact that for a subgraph G' of an l-DAG G holds:

- $\text{MEC}(G') \leq \text{MEC}(G)$,
- $\text{MECV}(G') \leq \text{MECV}(G)$,
- $\text{VE}(G') \leq \text{VE}(G)$,
- $\text{EE}(G') \leq \text{EE}(G)$.

This result allows us to develop new algorithms for lower bounds based on reduction to subgraphs and to prove several optimality preserving eliminations for all four problems. The fact that the order of vertex eliminations does not change the resulting l-DAG is another important outcome of this chapter. One implication of this result was discussed in this chapter, by showing that the search space of the VE problem is bigger than that of the MECV problem. The commutativity of vertex elimination also allows us to prove several optimality preserving eliminations and to show that vertex elimination solves the EE problem on absorption-free l-DAGs (see Section 4.3.1). The commutativity of edge elimination is not that easy to exploit, as the order of edge eliminations can have influence on the resulting l-DAG. Therefore this result is only used to reduce the search space within the branch and bound algorithm.

Chapter 4

Absorption-free l-DAGs

This chapter deals with the MEC, EE and VE problems on a special subclass of l-DAGs, the absorption-free l-DAGs. Absorption-free l-DAGs are DAGs that do not have parallel paths (Definition 4.1) and thus eliminations on these l-DAGs never cause absorption. In other publications absorption-free graphs are also called single connected graphs [Jab14]. This particular subset of the l-DAGs is of interest because you do not have to take care of absorptions. This fact makes it easier to develop strategies for solving the problems mentioned above. The hope is that one could port these strategies to non-absorption-free l-DAGs. One is also tempted to investigate whether the minimal edge count and the optimal Jacobian accumulation problems on this subclass are NP-hard or not. Several publications deal with this subclass of l-DAGs. In [Gri03] Griewank proved that face elimination solves the optimal Jacobian accumulation problem on this subset of l-DAGs. In [Nau04, p. 16] Naumann showed that the EE problem can be solved by vertex eliminations on absorption-free l-DAGs with at most two intermediate vertices. The generalization of this result, whether $VE = EE$ on absorption-free l-DAGs, has not been answered so far. We will not investigate face elimination in this chapter as this will go beyond the scope of this thesis as outlined in Chapter 1.

In this chapter we show that vertex elimination solves the minimal edge count problem on absorption-free l-DAGs. A linear time algorithm that solves the MEC problem on this subclass of l-DAGs is presented in Algorithm 4 showing that the minimal edge count problem on absorption-free l-DAGs is tractable. These results were already published in [MN12]. The main outcome of this chapter is the generalization of the result of Naumann by proving that vertex elimination solves the optimal edge elimination problem on absorption-free l-DAGs. To support solving the optimal edge (vertex) elimination problem on absorption-free l-DAGs, algorithms computing lower bounds and optimality preserving eliminations are developed in Section 4.3.3 and Section 4.3.2.

The chapter is structured as follows. In the first section (Section 4.1) we discuss the properties of absorption-free l-DAGs. Although absorption-free l-DAGs have been used in several publications, the properties of this subclass of graphs have never been explored. In Section 4.2 all results for the minimum edge count on absorption-free l-DAGs are summarized. Especially we prove that $MECV = MEC$ on absorption-free l-DAGs and develop the linear time algorithm for solving this problem. The last section (Section 4.3) deals with optimal edge and vertex elimination problems on absorption-free l-DAGs. There we show that $VE = EE$ on absorption-free l-DAGs (Section 4.3.1), prove various optimality preserving results (Section 4.3.2) and develop new lower bounds (Section 4.3.3).

4.1 Properties of absorption-free l-DAGs

The most important property for this thesis proved in this section is the fact that the number of successors and predecessors of any vertex is increasing during the edge or vertex eliminations sequence. We also prove upper bounds for the number of edges and intermediate vertices. Two algorithms are developed. One algorithm checks whether a given l-DAG is absorption-free or not (Algorithm 2). The other one builds an absorption-free subgraph of a given l-DAG (Algorithm 3), similar to the Kruskal's "minimal spanning tree" algorithm ([Kru56]). The second algorithm is used in Section 6.3.2 within an algorithm that determines the lower bounds by reduction to absorption-free subgraph (Algorithm 26).

The section is structured as follows. First we present the definition of absorption-free l-DAGs and prove that the number of predecessors and successors cannot be decreased by vertex or edge elimination. We then prove upper bounds for the number of edges and intermediate vertices in absorption-free l-DAGs. At the end of this section we develop the two algorithms mentioned in the previous paragraph.

Definition 4.1. *An l-DAG G is called absorption-free if any two of its vertices are connected by at most one directed path.*

For example, the lion graph (Figure 4.21(a)) is not absorption-free because the vertices v_1 and y_1 are connected by two directed paths. If we destroy one of these paths, we get an absorption-free l-DAG. Hence the lion graph without the edge (v_2, y_1) (Figure 3.10(c)) is absorption-free because all vertices are connected by at most one directed path.

Before we go on we must ensure that an absorption-free l-DAG is still absorption-free no matter what elimination was applied to it. This is proved in the next lemma.

Lemma 4.1. *Consider an absorption-free l-DAG G . For all $(i, j) \in E^*(G)$ and $v \in Z(G)$ holds*

1. $G - (i, j)^+$ is an absorption-free l-DAG,
2. $G - (i, j)^-$ is an absorption-free l-DAG,
3. $G - v$ is an absorption-free l-DAG.

Proof. This result follows trivially because any eliminations only contract existing paths and do not introduce new connections between vertices. Hence the number of paths between vertices cannot increase. $\square$

One important property of the absorption-free l-DAGs, which is used very extensively in later proofs, is the fact that the number of predecessors and successors of a vertex cannot be decreased by vertex elimination. In contrast, this can happen on arbitrary l-DAGs due to absorption. For example the vertex v_2 in l-DAG from Figure 3.5(a) has 5 predecessors. After the elimination of vertex v_1 it has only 4 predecessors (see Figure 3.5(b)). First we prove a corresponding property for front and back elimination of an edge.

Lemma 4.2. *Consider an absorption-free l-DAG G and $(i, j) \in E^*(G)$.*

1. *For all $v \in Z(G - (i, j)^+) \setminus \{j\}$ holds*

$$|S_{G'}(v)| \geq |S_G(v)| \quad \text{and} \quad |P_{G'}(v)| \geq |P_G(v)| \quad ,$$

where $G' = G - (i, j)^+$.

2. *For all $v \in Z(G - (i, j)^-) \setminus \{i\}$ holds*

$$|S_{G'}(v)| \geq |S_G(v)| \quad \text{and} \quad |P_{G'}(v)| \geq |P_G(v)| \quad ,$$

where $G' = G - (i, j)^-$.

Proof. 1).

Case 1. $|P_G(j)| > 1$. Hence (i, j) is the only edge removed by the front elimination of (i, j) . Therefore to prove the claim for this case it is sufficient to show that front elimination of (i, j) introduces at least one new edge with source i . j is not an output, hence $S_G(j) \neq \emptyset$. Let $k \in S_G(j)$. Front elimination of (i, j) introduces the edge (i, k) . As G is an absorption-free l-DAG $k \notin S_G(i)$ and hence the claim holds.

Case 2. $|P_G(j)| = 1$. Hence j is removed from G as a consequence of front elimination of (i, j) . For each $k \in S_G(j)$ the front elimination of (i, j) introduces the edge (i, k) . As G is an absorption-free l-DAG $i \notin P_G(k)$, and therefore the number of predecessors of k remains the same. Using the same argument as in **Case 1** one can show that the number successors of i is also not decreased by front elimination of (i, j) and hence the claim holds.

2). The proof is symmetrical to the proof of 1). $\square$

As vertex elimination is defined as a sequence of edge eliminations, the elimination of a vertex cannot reduce the predecessor and successor sets either, as is shown in the following lemma.

Lemma 4.3. *Consider an absorption-free l-DAG G and an intermediate vertex $v \in Z(G)$. For all $i \in Z(G - v)$ holds*

$$|S_{G-v}(i)| \geq |S_G(i)| \quad \text{and} \quad |P_{G-v}(i)| \geq |P_G(i)| \quad .$$

Proof. Follows directly from Lemma 4.2. $\square$

In the following lemma we investigate how a particular edge elimination changes the number of edges in the resulting graph. The main conclusion is that the number of edges in absorption-free graphs can be decreased only by eliminations causing the removal of a vertex. In all other cases the number of edges is either increased or remains the same.

Lemma 4.4. *Consider an absorption-free l-DAG G and an edge $(i, j) \in E(G)$.*

1. *For front elimination of (i, j) holds*

$$1a) \quad |P_G(j)| \geq 2 \Rightarrow |E(G - (i, j)^+)| = |E(G)| + |S_G(j)| - 1 \geq |E(G)|$$

$$1b) \quad |P_G(j)| = 1 \Rightarrow |E(G - (i, j)^+)| = |E(G)| - 1$$

2. *For back elimination of (i, j) holds*

$$2a) \quad |S_G(i)| \geq 2 \Rightarrow |E(G - (i, j)^-)| = |E(G)| + |P_G(i)| - 1 \geq |E(G)|$$

$$2b) \quad |S_G(i)| = 1 \Rightarrow |E(G - (i, j)^-)| = |E(G)| - 1 \quad .$$

Proof. 1a) As $|P_G(j)| \geq 2$ none of the vertices is removed from G . Hence front elimination of (i, j) leads to the removal of (i, j) and the insertion of (i, k) for all $k \in S_G(j)$. Hence, $|E(G - (i, j)^+)| = |E(G)| + |S_G(j)| - 1$ which completes the proof.

1b) The only difference to 1a) is that all edges that are incident to j have to be removed from the graph. Hence, $|E(G - (i, j)^+)| = |E(G)| + |S_G(j)| - 1 - |S_G(j)| = |E(G)| - 1$.

2a) The proof is similar to 1a).

2b) The proof is similar to 1b). $\square$

Now we transfer the previous result to vertex elimination. This result is the main idea behind the proof that vertex elimination solves the MEC problem on absorption-free l-DAGs.

Lemma 4.5. *Consider an absorption-free l-DAG G and an intermediate vertex $v \in Z(G)$.*

1. $v \in \mathcal{M}(G) \Rightarrow |E(G - v)| = |E(G)| - 1$
2. $v \notin \mathcal{M}(G) \Rightarrow |E(G - v)| \geq |E(G)|$.

Proof. This result follows directly from the repeated application of Lemma 4.4 to remove all in- or all out-edges of v . $\square$

One can say that absorption-free l-DAGs are trees or forests (if not connected) in the directed sense. If we recall the definition of a tree, tree is undirected graph that has no cycles, implying that two vertices in a tree are connected by only one path. Similar to the trees the number of edges in absorption-free graph is bounded by the number of vertices. We prove this fact in the next lemma.

Lemma 4.6. *The following inequality holds for all absorption-free l-DAGs G*

$$|E(G)| \leq |X(G)| \cdot |Y(G)| + |\mathcal{M}(G)| \quad .$$

Proof. Proof by induction over the number m of elements in $\mathcal{M}(G)$.

$m = 0$: Hence $\mathcal{M}(G) = \emptyset$. Let $[a_1, \dots, a_n]$ be a complete vertex elimination sequence in G . We set $G_0 = G$ and $G_{i+1} = G_i - a_{i+1}$. From Lemma 4.3 follows that $\mathcal{M}(G_{i+1}) \subseteq \mathcal{M}(G_i) = \emptyset$ for all $0 \leq i \leq n - 1$. Hence according to Lemma 4.5

$$|E(G_{i+1})| \geq |E(G_i)| \quad ,$$

for all $0 \leq i \leq n$. G_n is a bipartite graph with $X(G)$ and $Y(G)$ as partitions. Hence it can contain at most $|X(G)| \cdot |Y(G)|$ edges. Therefore

$$|E(G)| \leq |E(G_n)| \leq |X(G)| \cdot |Y(G)| \quad .$$

Hence the claim holds for $m = 0$.

$m \mapsto m + 1$: Hence $|\mathcal{M}(G)| = m + 1$. Let $v \in \mathcal{M}(G)$. According to Lemma 4.5

$$|E(G)| = |E(G - v)| + 1 \quad . \tag{4.1}$$

From Lemma 4.3 follows that $\mathcal{M}(G - v) \subseteq \mathcal{M}(G)$. After the elimination of vertex v the resulting l-DAG does not contain v and therefore $v \notin \mathcal{M}(G - v)$. Meaning that $|\mathcal{M}(G - v)| \leq m$. Hence we can apply induction assumption to $G - v$. Yielding

$$|E(G - v)| \leq |X(G - v)| \cdot |Y(G - v)| + |\mathcal{M}(G - v)| \quad . \tag{4.2}$$

The input and the output sets are not touched by the vertex elimination and therefore

$$X(G) = X(G - v) \quad \text{and} \quad Y(G) = Y(G - v) \quad . \tag{4.3}$$

Combining results from Equation (4.1), Equation (4.2) and Equation (4.3) together we get

$$|E(G)| = |E(G - v)| + 1 \leq |X(G)| \cdot |Y(G)| + \underbrace{|\mathcal{M}(G - v)|}_{\leq m} + 1 \leq |X(G)| \cdot |Y(G)| + |\mathcal{M}(G)| \quad .$$

$\square$

Replacing the set $\mathcal{M}(G)$ by the set of intermediate nodes, yields a less sharp but more straight forward upper bound for the number of edges in absorption-free l-DAGs.

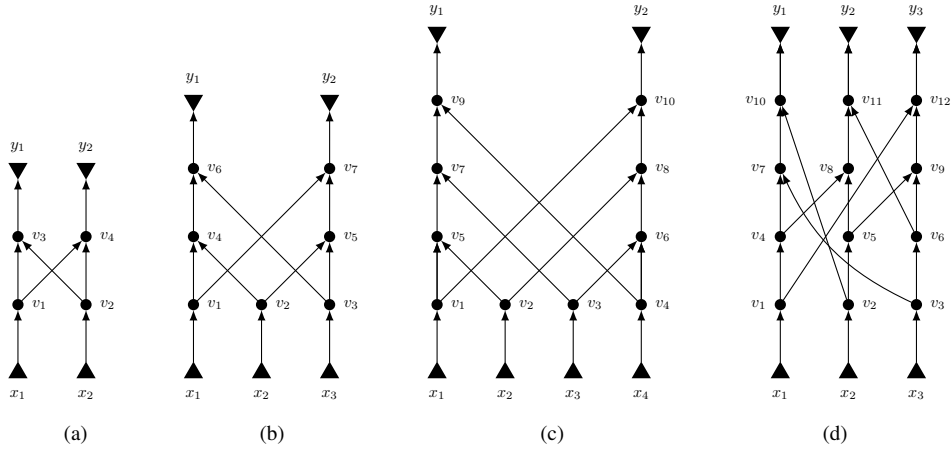


Figure 4.1: The bounds in Lemma 4.6, Corollary 4.1 and Lemma 4.7 cannot be improved

Corollary 4.1. *The following inequality holds for all absorption-free l-DAGs G*

$$|E(G)| \leq |X(G)| \cdot |Y(G)| + |Z(G)| \quad .$$

Proof. This result follows immediately from Lemma 4.6 by taking into account that $\mathcal{M}(G) \subseteq Z(G)$. $\square$

Another important property of absorption-free l-DAGs is that, under the assumption that the l-DAG has no vertices with Markowitz degree one, the number of intermediate nodes in an absorption-free l-DAG is bounded by the number of inputs and outputs. Of course absorption-free l-DAGs can contain a lot of vertices with Markowitz degree one. Still the existence of such vertices does not make the problem any harder. As we will show later, elimination of such vertices preserves optimality in the context of MEC (Lemma 4.10) and VE (Lemma 4.16) problems. Hence such vertices can (and will) be eliminated before performing any other steps. So we can securely assume that we deal with absorption-free l-DAG without vertices with Markowitz degree one.

Lemma 4.7. *For all absorption-free l-DAGs G with $|P(v)| \cdot |S(v)| > 1$ for all $v \in Z(G)$ holds*

$$|Z(G)| \leq 2|X(G)| \cdot |Y(G)| - |X(G)| - |Y(G)| \quad .$$

Proof. From $|P(v)| \cdot |S(v)| > 1$ follows that $|P(v)| + |S(v)| > 3$ for all $v \in Z(G)$ and therefore

$$\begin{aligned} 2|E(G)| &= \sum_{v \in V(G)} (|P(v)| + |S(v)|) = \sum_{v \in Z(G)} (|P(v)| + |S(v)|) + \sum_{v \in X(G)} |S(v)| + \sum_{v \in Y(G)} |P(v)| \\ &\geq 3|Z(G)| + |X(G)| + |Y(G)| \quad . \end{aligned}$$

From Corollary 4.1 follows

$$2|E(G)| \leq 2(|X(G)| \cdot |Y(G)| + |Z(G)|) = 2|X(G)| \cdot |Y(G)| + 2|Z(G)| \quad .$$

Hence

$$3|Z(G)| + |X(G)| + |Y(G)| \leq 2|X(G)| \cdot |Y(G)| + 2|Z(G)| \quad .$$

Therefore the claim holds. $\square$

In Figure 4.1 we see several examples, where the equality in the bounds presented Lemma 4.6, Corollary 4.1 and Lemma 4.7 holds. Hence we cannot improve these bounds.

As we can see on Figure 4.1(d), it can be very difficult to decide whether an l-DAG is an absorption-free or not. So we need an efficient algorithm to detect absorption-free l-DAGs. Such an algorithm is presented in Algorithm 2. The idea behind this algorithm is that an l-DAG with only one output is

Algorithm 2 Check that l-DAG is absorption-free

Require: l-DAG G

Ensure: **true** if G is absorption-free and **false** if not

```

1:  $L \leftarrow Y(G)$ 
2: absorption_free  $\leftarrow$  true
3: repeat
4:   choose  $y \in L$ 
5:    $H \leftarrow$  l-DAG induced by all paths connecting  $X(G)$  with  $y$ 
6:    $U \leftarrow Z(H) \cup X(G)$ 
7:   repeat
8:     choose  $v \in U$ 
9:     if  $|S_H(v)| > 1$  then
10:      absorption_free  $\leftarrow$  false
11:    end if
12:     $U \leftarrow U \setminus \{v\}$ 
13:  until  $U = \emptyset$  or  $\neg$ absorption_free
14:   $L \leftarrow L \setminus \{y\}$ 
15: until  $L = \emptyset$  or  $\neg$ absorption_free
16: return absorption_free

```

absorption-free if and only if every vertex has only one outgoing edge. Similarly, we can provide a result for l-DAGs with only one input. In the algorithm for each output y of the l-DAG G we build a subgraph consisting of all paths connecting all inputs with the output y . It turns out that if all these subgraphs are absorption-free, then G is also absorption-free. The next lemma proves the correctness of this approach.

Lemma 4.8. *Algorithm 2 checks if given l-DAG G is absorption-free.*

Proof. First we show that if Algorithm 2 returns **false** then G is not absorption-free. Assume that Algorithm 2 returns **false**. Let $y \in Y(G)$ be the output for which Algorithm 2 returns **false** and H is an l-DAG that is induced by all vertices and edges connecting y and $X(G)$. H must contain a vertex that has at least two outgoing edges. Let v be such a vertex that has two outgoing edges in H . Hence there exist at least two different paths from v to y in H . Therefore there exist at least two different paths from v to y in G implying that G is not absorption-free.

To finish the proof it is sufficient to show that Algorithm 2 returns **false** if G is not absorption-free. We assume that G is not absorption-free. This implies that there exist two vertices a and b connected by more than one path. W.l.o.g. $a \prec b$. Let $y_0 \in Y(G)$ be an output that is connected to b . We define H as an l-DAG that consists of all paths connecting y_0 and $X(G)$. As b is connected to y_0 , the l-DAG H contains a too. Hence H must contain all paths connecting a and b . Therefore H contains at least two different paths connecting a and b . Let

$$a = v_1, v_2, \dots, v_n = b \quad \text{and} \quad a = w_1, w_2, \dots, w_m = b$$

be two directed paths connecting a and b in H . We set j_0 as the greatest integer such that $v_i = w_i$ for all $1 \leq i \leq j_0$. Hence $v_{j_0+1} \neq w_{j_0+1}$ implying that v_{j_0} has at least two successors v_{j_0+1} and w_{j_0+1} . Therefore Algorithm 2 returns **false** in this case. $\square$

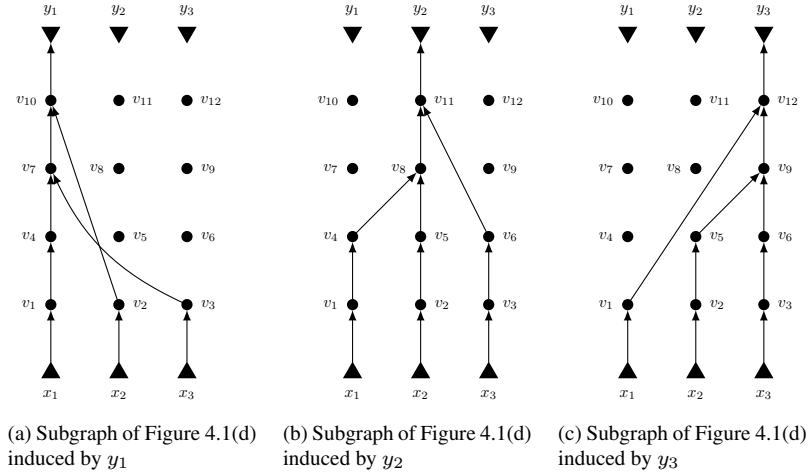


Figure 4.2: Application of Algorithm 2 to Figure 4.1(d) (illustrating Example 4.1)

Algorithm 2 is similar to the Algorithm presented in [Jab14]. Before we discuss Algorithm 2 on any example, let us first look at the complexity of this algorithm. The outer loop is repeated at most $|Y(G)|$ times as we need to loop over all outputs of a given l-DAG. Inside the loop we first need to build the l-DAG induced by all paths connecting the inputs of G with y . Therefore we could run a breadth first search algorithm starting from y . The breadth first search algorithm would use each edge connected to y once. In the worst case all edges that are not incident with other outputs are connected to y . Hence complexity of constructing H is $O(|E(G)|)$. Only the complexity of the inner loop is left. In the inner loop we need to check the number of successors of each vertex in H . Hence the complexity of the inner loop is $O(|V(G)|)$ as in worst case H can contain all vertices of G with except of the outputs. Hence the worst case complexity of Algorithm 2 is

$$O((|E(G)| + |V(G)|) \cdot |Y(G)|) \quad .$$

Let us apply Algorithm 2 to the l-DAG on the Figure 4.1(d) and show that this l-DAG is absorption-free.

Example 4.1. Consider l-DAG G as on Figure 4.1(d). The three subgraphs that are induced by the outputs from G are presented on Figure 4.2. All vertices of these subgraphs have only one successor and therefore according to Algorithm 2, the l-DAG G is an absorption-free l-DAG.

Using Algorithm 2 we can now formulate an algorithm that determines an absorption-free subgraph to a given l-DAG G , presented in Algorithm 3. This algorithm is used to produce lower bounds as in Section 5.4 and Section 6.3.2. It is based on the “spanning tree” algorithm from Kruskal, with the only difference that we have replaced the check for a undirected cycle with a check for absorption-free l-DAG as in Algorithm 2. The algorithm uses weights to decide which edges should be included in the subgraph. In the context of building lower bounds these weights can be used to force the algorithm to choose the absorption-free subgraph with better lower bounds. (Note: The algorithm does not construct a minimal “spanning tree” or absorption-free l-DAG. The weights can only be used to force the removal of some edges.)

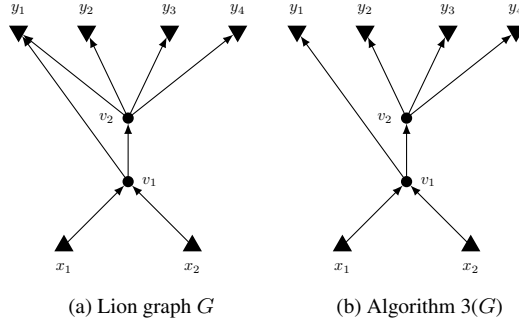
Let us discuss the complexity of the Algorithm 3. We assume that the edges are already sorted according to their weights. We have to repeat the **while** loop $|E(G)|$ times. Inside the loop we actually

Algorithm 3 Construction of an absorption-free subgraph of G **Require:** weighted l-DAG G with weights w **Ensure:** $E' \subseteq E(G)$ such $T[E']$ absorption-free subgraph of G

```

1:  $E' \leftarrow \emptyset$ 
2:  $L \leftarrow E$ 
3: while  $L \neq \emptyset$  do
4:   choose  $(i, j)$  with the smallest weight
5:    $L \leftarrow L \setminus \{(i, j)\}$ 
6:   if ALGORITHM 2( $T[E' \cup \{(i, j)\}]$ ) then
7:      $E' \leftarrow E' \cup \{(i, j)\}$ 
8:   end if
9: end while
10: return  $T[E']$ 

```

Figure 4.3: Application of Algorithm 3 to the lion graph with $w(v_2, y_1) = 10$ (illustrating Example 4.2)

only execute the Algorithm 2 the worst case complexity of which is $O((|E(G)| + |V(G)|) \cdot |Y(G)|)$. Hence Algorithm 3 has a worst case complexity of

$$O((|E(G)| + |V(G)|) \cdot |Y(G)| \cdot |E(G)|).$$

In the following example we apply Algorithm 3 to the lion graph.

Example 4.2. Consider l-DAG G is a lion graph (compare Figure 4.3(a)). We assume that all edges of G are weighted with 1 except (v_2, y_1) which is weighted with 10. If we apply Algorithm 3 to G , the algorithm will add edges to E' until it comes to the last edge (v_2, y_1) as only (v_2, y_1) makes the l-DAG induced by E' non-absorption-free. Hence Algorithm 3 will not keep the edge (v_2, y_1) . Therefore the algorithm will build an absorption-free subgraph of the lion graph as in Figure 4.3(b).

There is one big conceptual difference between the Algorithm 3 and the standard Kruskal algorithm. The standard Kruskal algorithm preserves all vertices in the resulting “spanning tree”. While the result of Algorithm 3 may contain less vertices as the original l-DAG. This happens once we build the subgraph T which is induced by the set of chosen edges E' . In this step we remove all vertices that are not connected to the inputs and outputs. Following example demonstrates this behavior.

Example 4.3. Consider l-DAG G is a lion graph as (compare Figure 4.4(a)). We assume that all edges of G are weighted with 1 except (v_1, v_2) which is weighted with 10. If we apply the Algorithm 3 to G it will construct the set E' that contains all edges save (v_1, v_2) (see Figure 4.4(b)). The resulting l-DAG as

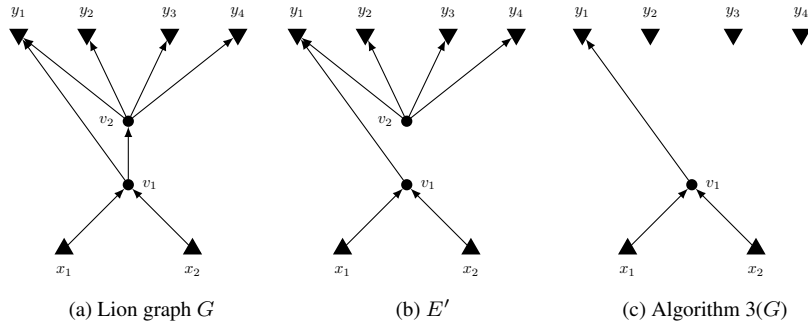


Figure 4.4: Application of Algorithm 3 to the lion graph with $w(v_1, v_2) = 10$ (illustrating Example 4.3)

shown in Figure 4.4(b) is not a subgraph of G because the vertex v_2 is an input of this l-DAG as it has no predecessors. Therefore to get an absorption-free subgraph of G , vertex v_2 must be removed together with all incident edges. This is what happens once we construct $T[E']$. The output of the Algorithm 3 is shown on Figure 4.4(c).

4.2 Minimal edge count problem on absorption-free l-DAGs

The results presented in this section were already published in [MN12]. The main contribution of this section is the proof that elimination of all vertices with one in-edge or one out-edge solves the minimal edge count problem for absorption-free l-DAGs (Theorem 4.2). Restriction to *shortest* optimal elimination sequences allows us to ignore eliminations that leave the number of edges in the graph unchanged such as the elimination of a vertex with two in- and two out-edges. Strictly decreasing edge counts simplify the proofs considerably. The restriction to the corresponding elimination sequences turns out to be feasible.

Important milestones for the proof of the main result are Lemma 4.9 and Lemma 4.10. A direct consequence of these two lemmas is that elimination of all vertices with one in-edge or one out-edge solves the MECV problem for absorption-free l-DAGs (Theorem 4.1).

Lemma 4.3 shows that the number of vertices whose elimination yields a reduction of the number of edges in the graph is strictly decreasing during the vertex elimination procedure.

In the following lemma we show that the shortest optimal vertex elimination sequence contains only eliminations of vertices with one in- or one out-edge.

Lemma 4.9. *Consider an absorption-free l-DAG G . If $[a_1, \dots, a_n]$ is a shortest vertex elimination sequence that solves the MECV problem, then $a_i \in \mathcal{M}(G_{i-1})$ for all $1 \leq i \leq n$ and $\mathcal{M}(G_n) = \emptyset$, where $G_i = G - [a_1, \dots, a_i]$.*

Proof. $\mathcal{M}(G_n)$ is empty because otherwise $|E(G_n - v)| \leq |E(G_n)|$ for all $v \in \mathcal{M}(G_n)$. Hence it is sufficient to show that $a_i \in \mathcal{M}(G_{i-1})$ for all $1 \leq i \leq n$. The proof is by contradiction. Let $[a_1, \dots, a_n]$ be a shortest vertex elimination sequence that solves the MECV problem and let $a_{i_0} \notin \mathcal{M}(G_{i_0-1})$. From Lemma 3.7 follows that without loss of generality we can assume $i_0 = n$. From Lemma 4.5 follows

$$|E(G_n)| \geq |E(G_{n-1})|$$

as $a_n \notin \mathcal{M}(G_{n-1})$. Therefore $[a_1, \dots, a_{n-1}]$ also solves the MECV problem, while containing only $n-1$ vertex eliminations, which is a contradiction to the assumption that $[a_1, \dots, a_n]$ is a shortest elimination sequence. $\square$

So we now know that the shortest optimal vertex elimination sequence contains only eliminations with one in- or one out- edge. The next lemma shows that elimination of such vertices preserves optimality.

Lemma 4.10. *Consider an absorption-free l-DAG G . For all $v \in Z(G)$ holds*

$$v \in \mathcal{M}(G) \quad \Rightarrow \quad \text{MECV}(G) = \text{MECV}(G - v) \quad .$$

Proof. Consider $[a_1, \dots, a_n]$ as a shortest vertex elimination sequence that solves the MECV problem for G and let $v \in \mathcal{M}(G)$. We set $G_0 = G$ and $G_{i+1} = G_i - a_{i+1}$ for $1 \leq i \leq n-1$. If $v = a_{i_0}$ for some $i_0 \in \{1, \dots, n\}$, then the claim is trivially true. Therefore we assume that $v \neq a_{i_0}$ for all $1 \leq i_0 \leq n$. Hence $v \notin \mathcal{M}(G_n)$ as otherwise $|E(G_n - v)| < |E(G_n)|$, which is a contradiction to the assumption that $[a_1, \dots, a_n]$ solves the MECV problem. Let i_0 be the smallest integer with $v \notin \mathcal{M}(G_{i_0})$.

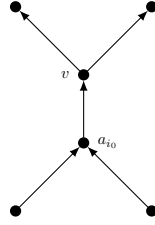


Figure 4.5: Proof of Lemma 4.10

Then a_{i_0} and v must be adjacent in G_{i_0-1} (see Figure 4.5). Without loss of generality we assume that $(a_{i_0}, v) \in E(G_{i_0-1})$. The other case is symmetric. The successor set of v is not modified by the elimination of a_{i_0} and hence $|S_{G_{i_0-1}}(v)| > 1$ as otherwise $v \in \mathcal{M}(G_{i_0})$. This observation also implies that $P_{G_{i_0-1}}(v) = \{a_{i_0}\}$, because $v \in \mathcal{M}(G_{i_0-1})$. Additionally $|P_{G_{i_0-1}}(a_{i_0})| > 1$, as otherwise $|P_{G_{i_0}}(v)| = 1$ and therefore $v \in \mathcal{M}(G_{i_0})$. From Lemma 4.9 follows that $a_{i_0} \in \mathcal{M}(G_{i_0-1})$ implying $S_{G_{i_0-1}}(a_{i_0}) = \{v\}$. So summing up the results of this paragraph we can say that v is the only successor of a_{i_0} and a_{i_0} is the only predecessor of v , while the successor set of v and predecessor set of a_{i_0} has more than one element (compare Figure 4.5). It is easy to see that $G_{i_0-1} - v \cong G_{i_0}$, under the isomorphism

$$f : V(G_{i_0-1} - v) \longrightarrow V(G_{i_0}), \quad f(x) = \begin{cases} v & \text{if } x = a_{i_0}, \\ x & \text{if } x \neq a_{i_0} \end{cases} .$$

Hence $[a_1, \dots, a_{i_0-1}, v, a_{i_0+1}, \dots, a_n]$ solves the MECV problem and therefore according to Lemma 3.7 $[v, a_1, \dots, a_{i_0-1}, a_{i_0+1}, \dots, a_n]$ also does, finishing the proof. $\square$

The next result proves that the shortest vertex elimination sequence that solves the minimum edge count problem by vertex elimination only eliminates vertices with one in- or one out- edge.

Theorem 4.1. *Consider an absorption-free l-DAG G . $[a_1, \dots, a_n]$ is a shortest vertex elimination sequence that solves the minimum edge count problem for vertex elimination if and only if $a_i \in \mathcal{M}(G_{i-1})$ for all $1 \leq i \leq n$ and $\mathcal{M}(G_n) = \emptyset$, where $G_i = G - [a_1, \dots, a_i]$.*

Proof. " $\Rightarrow$ " This direction follows immediately from Lemma 4.9.

” $\Leftarrow$ ” Consider a vertex elimination sequence $[a_1, \dots, a_n]$, where $a_i \in \mathcal{M}(G_{i-1})$ for all $1 \leq i \leq n$ and $\mathcal{M}(G_n) = \emptyset$. From Lemma 4.10 follows that

$$\text{MECV}(G) = \text{MECV}(G_n) \quad .$$

As $\mathcal{M}(G_n) = \emptyset$ it follows from Lemma 4.5 and Lemma 4.3 that

$$\text{MECV}(G) = \text{MECV}(G_n) = |E(G_n)| \quad .$$

Hence the claim holds. $\square$

With Theorem 4.1 we actually know how to solve the minimum edge count problem by vertex elimination. We simply eliminate the vertices with one in- or one out edge as long as possible (compare Algorithm 4). This approach already solves the MECV problem. If we want, we can additionally eliminate the vertices with Markowitz degree 4 (vertices with two in and two out edges). These eliminations will not change the overall edge count but we can remove the not needed vertices. This can help to save memory while storing the resulting l-DAG.

Next we will solve the minimum edge count problem by edge eliminations. We will use the same idea as before and restrict ourselves to *shortest* edge elimination sequence. First we prove that a shortest edge elimination sequence solving the MEC problem consists only of non-pure edge eliminations. A reminder: a non-pure edge elimination is an edge elimination that causes a removal of a vertex. Such edge eliminations are actually vertex eliminations.

Theorem 4.2. *Consider an absorption-free l-DAG G . If $[e_1, \dots, e_n]$ is a shortest edge elimination sequence that solves the MEC problem, then e_i is a non-pure edge elimination for all $1 \leq i \leq n$.*

Proof. The proof is by induction over the length of the elimination sequence. We set

$$G_i = G - [e_1, \dots, e_i] \quad .$$

$n = 1$: According to Lemma 4.4 the number of edges cannot be reduced by pure edge elimination.

$n = 2$: From the case $n = 1$ follows that e_2 is a non-pure edge elimination. Hence it remains to be shown that the first elimination is also not pure. Let e_1 be a pure edge elimination. Then the following inequality holds:

$$|E(G - [e_1, e_2])| \stackrel{\text{Lemma 4.4}}{\geq} |E(G - [e_1])| - 1 \geq |E(G)| - 1 \quad . \quad (4.4)$$

So $[e_1, e_2]$ reduces the number of edges by at most one. Hence there exists no vertex $v \in Z(G)$ with $|S_G(v)| = 1$ or $|P_G(v)| = 1$ as otherwise

$$|E(G - v)| = |E(G)| - 1 \quad .$$

This implies that there would exist an edge elimination reducing the number of edges by one, which is a contradiction to the choice of $[e_1, e_2]$ as the *shortest* edge elimination sequence solving the MEC problem. So as $|P_G(v)| > 1$ and $|S_G(v)| > 1$ for all $v \in Z(G)$ with Lemma 4.4 follows that

$$|E(G - [e_1])| \geq |E(G)| + 1 \quad . \quad (4.5)$$

By application of Equation (4.5) to Equation (4.4) we get

$$|E(G - [e_1, e_2])| \stackrel{\text{Eq(4.4)}}{\geq} |E(G - [e_1])| - 1 \stackrel{\text{Eq(4.5)}}{\geq} |E(G)| + 1 - 1 = |E(G)| \quad ,$$

which is a contradiction to the fact that $[e_1, \dots, e_n]$ is a shortest edge elimination sequence, as an edge elimination sequence with length 0 also solves the MEC problem in this case.

$n \mapsto n + 1$: Consider $[e_1, e_2]$ as a shortest edge elimination sequence that solves the MEC problem. According to the induction hypothesis, the elimination of e_i is not pure for $2 \leq i \leq n + 1$. It remains to be shown that e_1 is also not pure. Lets assume that e_1 is a pure edge elimination. Then from Lemma 4.4 follows that

$$\text{MEC}(G) = |E(G - [e_1, \dots, e_{n+1}])| \geq |E(G)| - n \quad . \quad (4.6)$$

Case 1. Let e_1 represent the front elimination of an arbitrary edge $(k, l) \in E(G)$. As already mentioned, e_i is a non-pure edge elimination for all $i \in \{2, \dots, n + 1\}$. Hence it can be represented by elimination of a vertex $v_i \in Z(G_i)$ implying that the elimination sequence can be rewritten as $[(k, l)^+, v_2, \dots, v_{n+1}]$. As $n \geq 2$ there exists $i_0 \in \{2, \dots, n + 1\}$, where v_{i_0} is either not incident with (k, l) or $v_{i_0} = l$. According to Lemma 3.7 and without loss of generality we can assume that $i_0 = 2$. If $v_2 = l$, then $G - [(k, l)^+, v_2] = G - v_2$ and hence $G - [(k, l)^+, v_2, \dots, v_{n+1}] = G - [v_2, \dots, v_{n+1}]$. Consequently vertex elimination solves the MEC problem for this case and the shortest sequence consists of at most n eliminations. From Theorem 4.1 follows that each vertex elimination of the shortest vertex elimination sequence can also be realized by a single edge elimination implying that there exists an edge elimination sequence with a length of at most n that solves this MEC problem. This statement is a contradiction to the assumption that $[e_1, \dots, e_{n+1}]$ is the shortest edge elimination sequence.

Consider the case where v_2 is not incident to (k, l) . From Lemma 3.8 follows that $G - [(k, l)^+, v_2] = G - [v_2, (k, l)^+]$. Hence the shortest elimination sequence that solves the MEC problem for $G - v_2$ contains n eliminations. From the induction hypotheses follows that there exists an elimination sequence $[a_1, \dots, a_n]$ consisting only of non-pure edge eliminations. $v_2 \in \mathcal{M}(G - (k, l)^+)$ as otherwise it will be a contradiction to the assumption that $[e_1, \dots, e_{n+1}]$ is the shortest edge elimination sequence (similar argument as above). According to the assumption $v_2 \neq l$ and hence from application of Lemma 4.2 to $v_2 \in \mathcal{M}(G - (k, l)^+)$ yields $v_2 \in \mathcal{M}(G)$. From Lemma 4.5 and Lemma 4.4 follows that

$$|E(G - [v_2, a_1, \dots, a_n])| = |E(G)| - n - 1 \quad .$$

The latter is a contradiction to Equation (4.6).

Case 2. (k, l) is back eliminated. This case is symmetric to Case 1 resulting in a similar argument. □

According to Definition 2.2, an edge elimination causes a removal of a vertex. The elimination of this vertex achieves the same result. Hence we can easily rewrite a sequence of edge eliminations into a sequence of corresponding vertex eliminations. This brings us to the proof of the next theorem showing that vertex elimination solves the MEC problem on absorption-free l-DAGs.

Theorem 4.3. $\text{MEC}(G) = \text{MECV}(G)$ for any absorption-free l-DAG G .

Proof. This result follows immediately from Theorem 4.2. □

Elimination of all vertices having a single in-edge or a single out-edge or both solves the MEC problem for absorption-free l-DAGs. This result leads to the linear-time algorithm presented in Algorithm 4. So we have a proof that vertex elimination solves the MEC problem on absorption-free graphs and moreover we have a linear time algorithm (Algorithm 4) solving this problem. Before we close this section let us discuss the application of Algorithm 4 to the l-DAG on Figure 4.6(a).

Example 4.4. Consider l-DAG G as on Figure 4.6(a). If we apply Algorithm 4 to G , the algorithm will first remove one of the vertices that have in- or one out-edge. As you can see, all three intermediate vertices fulfill this requirement. Assume that the algorithm first eliminates the vertex v_2 . The resulting l-DAG is shown on Figure 4.6(b). On the resulting $G - v_2$ only the vertex v_3 has one in- or one out-edge. Hence Algorithm 4 will eliminate this vertex next. The resulting l-DAG is presented on Figure 4.6(c). After this the algorithm stops. So we could reduce the number of edges in G by two.

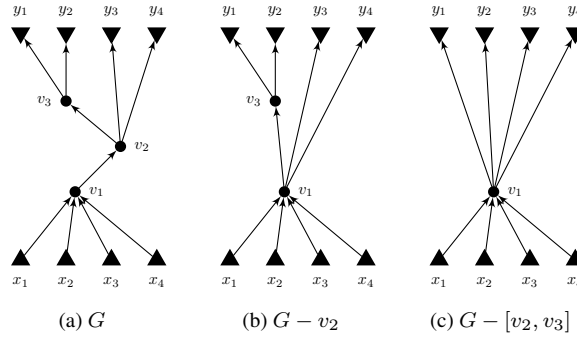
Algorithm 4 Solving $\text{MEC}(G)$ for absorption-free l-DAGs**Require:** absorption-free l-DAG G **Ensure:** G with $|E(G)| = \text{MEC}(G)$ 1: **while** $\mathcal{M}(G) \neq \emptyset$ **do**2: choose $v \in \mathcal{M}(G)$ 3: $G \leftarrow G - v$ 4: **end while**

Figure 4.6: Application of Algorithm 4 (illustrating Example 4.4)

4.3 Optimal vertex and edge elimination problems on absorption-free l-DAGs

This section deals with the vertex and edge eliminations problem on absorption-free l-DAGs. Unlike the minimal edge count problem on absorption-free l-DAGs the optimal Jacobian accumulation problem on this subclass of l-DAGs was already examined in the literature. In [GW08] Griewank proved that face elimination solves the structural optimal Jacobian accumulation problem on absorption-free graphs. In [Nau04] Naumann made first steps to show that vertex elimination solves the optimal edge elimination problem on absorption-free l-DAGs. There he proved that VE and EE problems are equivalent (with respect to the optimal solution) on absorption-free l-DAGs with at most two intermediate vertices. A lower bound based on sum of Markowitz degrees for absorption-free l-DAGs was presented in [NH08].

In this section we will extend the result from [Nau04] and prove that vertex elimination solves the optimal edge elimination problem on absorption-free l-DAGs, develop various optimality preserving eliminations and discuss their application. At the end we will present a new idea for computing lower bounds and present algorithms exploiting this idea. We will prove that the new lower bounds on absorption-free l-DAGs are better than the bound based on sum of Markowitz degrees.

This section is structured as follows, first we show that $\text{VE} = \text{EE}$ on absorption-free l-DAGs (Section 4.3.1). The optimality preserving eliminations are discussed in the next subsection (Section 4.3.2). In the last subsection (Section 4.3.3) we will develop the new lower bounds.

4.3.1 Vertex elimination solves the edge elimination problem on absorption-free l-DAGs

This subsection contains the proof that $\text{EE} = \text{VE}$ on absorption-free l-DAGs. As already mentioned previously, this is a generalization of the result of Naumann in [Nau04, p. 16]. The proof makes extensive

use of the results from Section 3.2.1, Section 3.2.2 and Section 3.2.5. The central part of this section is the proof of Lemma 4.15. The fact that $EE = VE$ (Theorem 4.4) on absorption-free l-DAGs is a direct implication of this result.

Before we can start with the proof of $EE = VE$ we need some auxiliary results. We already know that we can permute the order of two subsequent vertex eliminations without changing the costs, if these vertices are not adjacent (compare Lemma 3.13). Here we extend this result to vertex elimination sequences with more than two eliminations. Therefore we want to know in which situations moving one of the later vertex eliminations to the very beginning of the elimination sequence does not change the overall costs as this vertex is not adjacent to any of the vertices that are eliminated prior to it. Before we discuss this observation a small example, we first need the following definition.

Definition 4.2. Consider l-DAG G . We refer to the l-DAG $H = (V, S)$, where

$$S = \{\{i, j\} \mid (i, j) \in E(G)\}$$

as the underlying undirected graph of G .

Example 4.5. Consider the l-DAG G as on Figure 4.7(a) and the vertex elimination sequence $[v_2, v_1, v_3]$. Obviously we cannot change the order of elimination of v_2 and v_1 as they are adjacent in G . The vertices v_1 and v_3 are not adjacent in G but these vertices are adjacent in $G - v_2$ (compare Figure 4.7(b)). So changing the order of elimination of vertices v_3 and v_1 change the costs of the elimination sequence as

$$\sigma_{G-v_2}([v_1, v_3]) = 12 + 8 = 20 \quad \text{and} \quad \sigma_{G-v_2}([v_3, v_1]) = 2 + 16 = 18 \quad .$$

The vertices v_1 and v_3 are adjacent in $G - v_2$ although they are not adjacent in G because there exists a directed path from v_1 to v_3 through v_2 . So both sets $\{v_2, v_1\}$ and $\{v_2, v_1, v_3\}$ are connected on the underlying undirected graph.

In vertex elimination sequence $[v_1, v_3, v_2]$ we can change the order of v_1 and v_3 as they are not adjacent or in other words $\{v_3, v_1\}$ is not connected on the underlying undirected graph.

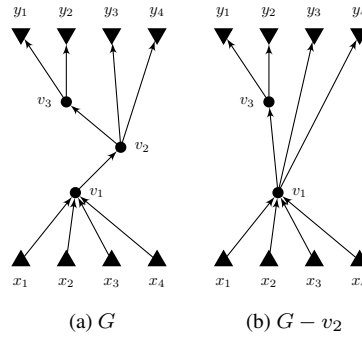


Figure 4.7: Illustration of Example 4.5

This leads us to the following definition.

Definition 4.3. Consider an l-DAG G . A vertex elimination sequence $[v_1, \dots, v_n]$ is called forward connected in G if $\{v_1, \dots, v_i\}$ is connected on the underlying undirected graph of G for all $i \leq n$.

Next we prove that if a vertex elimination sequence is not forward connected then there exists at least one vertex that can be eliminated at the beginning without changing the overall costs. Meaning that forward connectivity of a vertex elimination sequence is exactly the criteria we were looking for.

Lemma 4.11. Consider l -DAG G . A vertex elimination sequence $[v_1, \dots, v_n]$ is forward connected if and only if for each $1 < i_0 \leq n$ there exists $1 \leq j_0 < i_0$ such that v_{i_0} and v_{j_0} are adjacent in G_{j_0-1} , where $G_0 = G$ and $G_i = G_{i-1} - v_i$.

Proof. First we prove the forward direction.

“ $\Rightarrow$ ”: $[v_1, \dots, v_n]$ is forward connected and hence $\{v_1, \dots, v_{i_0}\}$ is connected on the underlying undirected graph of G . This implies there exists $j_0 < i_0$ such that v_{i_0} and v_{j_0} are adjacent in G and thereby also in G_{j_0-1} .

“ $\Leftarrow$ ”: The reverse direction can be easily proven by induction over the length of the elimination sequence. $\square$

So far we looked at not adjacent vertices as their order does not change the costs. But what about adjacent vertices in the elimination sequence as we saw in the Example 4.5? On absorption-free graphs we can formulate an easy-to-check criterion to decide which order of two subsequent vertex eliminations yields smaller costs. To prove this criterion we first need to determine the costs for both elimination orders $[a, b]$ and $[b, a]$ for two adjacent vertices a and b .

Lemma 4.12. Consider an absorption-free l -DAG $G = (V, E)$. For any edge $(a, b) \in E^*(G)$ holds

1. $\sigma[a, b] = |P(a)| \cdot |S(a)| + (|P(b)| - 1 + |P(a)|) \cdot |S(b)|$ and,
2. $\sigma[b, a] = |P(b)| \cdot |S(b)| + (|S(a)| - 1 + |S(b)|) \cdot |P(a)|$.

Proof. Ad 1. From definition of vertex elimination follows that

$$S_{G-a}(b) = S(b) \quad \text{and} \quad P_{G-a}(b) = P(a) \cup P(b) \setminus \{a\} \quad .$$

Due to the fact that G is absorption-free, none of the predecessors of a is also predecessor of b and therefore

$$|P_{G-a}(b)| = |P(b)| - 1 + |P(a)| \quad . \quad (4.7)$$

Hence the costs of the elimination sequence $[a, b]$ are

$$\begin{aligned} \sigma_G([a, b]) &= |P(a)| \cdot |S(a)| + |P_{G-a}(b)| \cdot \underbrace{|S_{G-a}(b)|}_{=|S(b)|} \\ &\stackrel{\text{Eq(4.7)}}{=} |P(a)| \cdot |S(a)| + (|P(b)| - 1 + |P(a)|) \cdot |S(b)| \quad , \end{aligned}$$

finishing the proof.

Ad 2. Analogous to 1. due to symmetry. $\square$

As we can see the number of predecessors and successors determines which order of eliminations yields smaller costs.

Corollary 4.2. Consider an absorption-free l -DAG G . For any edge $(a, b) \in E^*(G)$ holds

1. $\sigma_G([a, b]) < \sigma_G([b, a]) \Leftrightarrow |P(a)| < |S(b)|$
2. $\sigma_G([a, b]) = \sigma_G([b, a]) \Leftrightarrow |P(a)| = |S(b)|$
3. $\sigma_G([a, b]) > \sigma_G([b, a]) \Leftrightarrow |P(a)| > |S(b)|$

Proof. From Lemma 4.12 follows

$$\sigma_G([a, b]) - \sigma_G([b, a]) = |P(a)| - |S(b)| \quad . \quad (4.8)$$

The claim follows immediately from Equation (4.8). $\square$

With the help of the previous result we know in which order two subsequent eliminations of vertices a and b should be executed to minimize the costs without actually performing these eliminations. If the vertices are not adjacent it does not matter. If a is a predecessor of b we simply have to compare the number successors of b with the number of predecessors for a . If $|P(a)| < |S(b)|$ then a should be eliminated before b if $|P(a)| > |S(b)|$ then we should eliminate b before we eliminate a . If $|P(a)| = |S(b)|$ then the order of eliminations does not change the costs. The case that b is a predecessor of a is symmetrical.

Example 4.6. Consider an l-DAG as on Figure 4.7(a). Which order of eliminations yields smaller costs v_1 followed by v_2 or vice versa? The vertex v_1 has 4 predecessors while v_2 has 3 successors hence according Corollary 4.2

$$\sigma_G([v_1, v_2]) > \sigma_G([v_2, v_1]) \quad .$$

Similarly $\sigma_G([v_2, v_3]) < \sigma_G([v_3, v_2])$ as v_2 has only 1 predecessor and v_3 2 successors.

Next we transfer this result to vertex elimination sequences with more than two eliminations. The biggest problem hereby is to know whether we have to use the predecessor or the successors set of a vertex in the comparison. For example for two vertices v_1 and v_2 we need to compare the size of the predecessor set of v_1 with the size of the successor set of v_2 if v_1 is a predecessor of v_2 and vice versa if v_1 is a successor of v_2 we need to compare the size of the successor set of v_1 with the size of the predecessor set of v_2 . To circumvent this problem we use the following notation.

Definition 4.4. Consider an l-DAG G , $i, j \in V(G)$. We set

$$I_{i,j}^G = \begin{cases} |P_G(a_i)|, & (i, j) \in E(G) \\ |S_G(a_i)|, & (j, i) \in E(G) \\ 0, & \text{otherwise} \end{cases} \quad .$$

We refer to $I_{i,j}^G$ as bridge degree of i and j on G .

This notation is used in the following way. Assume that we want to know which order of vertex eliminations yields smaller costs $[v_1, v_2]$ or $[v_2, v_1]$. If v_1 is a predecessor of v_2 on G then $I_{v_1, v_2}^G = |P(v_1)|$ and $I_{v_2, v_1}^G = |S(v_2)|$. If v_2 is a predecessor of v_1 on G then $I_{v_1, v_2}^G = |S(v_1)|$ and $I_{v_2, v_1}^G = |P(v_2)|$. Hence according to Corollary 4.2 if $I_{v_1, v_2}^G < I_{v_2, v_1}^G$ then v_1 should be eliminated before v_2 , if $I_{v_1, v_2}^G > I_{v_2, v_1}^G$ then v_2 should be eliminated before v_1 . In the case of equality the costs of both elimination orders are the same. As you can see using the notation from Definition 4.4 we avoid mentioning the predecessor and successor sets directly. This makes future proofs shorter as we avoid case analysis. Using this notation we can prove the following inequality for optimal vertex elimination sequences.

Lemma 4.13. Consider absorption-free l-DAG G and optimal vertex elimination sequence $[v_1, \dots, v_n]$. The following holds for all $1 \leq i < n$

$$I_{v_i, v_{i+1}}^{G_{i-1}} \leq I_{v_{i+1}, v_i}^{G_{i-1}} \quad ,$$

where $G_0 = G$ and $G_i = G_{i-1} - v_i$.

Proof. Follows immediately from Corollary 4.2 and Definition 4.4. □

In forward connected vertex elimination sequences we typically have series of successive vertex eliminations, where each two vertices v_i and v_{i+1} are adjacent on $G - [v_1, \dots, v_{i-1}]$. We refer to such series as chains.

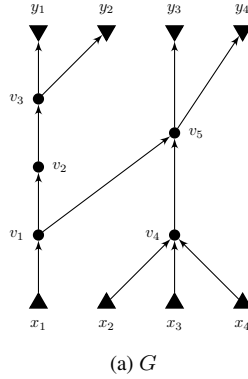


Figure 4.8: Illustration of Example 4.7

Example 4.7. Consider l -DAG G as on Figure 4.8 and a forward connected vertex elimination sequence $[v_1, v_2, v_3, v_5, v_4]$. This elimination sequence has two chains $[v_1, v_2, v_3]$ and $[v_5, v_4]$. $[v_1, v_2, v_3]$ is a chain because $(v_1, v_2) \in E(G)$ and $(v_2, v_3) \in E(G - v_1)$. v_3 and v_5 are not adjacent on $G - [v_1, v_2]$ and therefore the chain ends here. $[v_5, v_4]$ is also a chain because v_5 and v_4 are adjacent in $G - [v_1, v_2, v_3]$.

In optimal forward connected vertex elimination sequences the bridge degree is growing throughout the chains. This means that once we made a decision with a certain bridge degree all following decisions in this chain must have at least the same degree. This is especially true for the first bridge degree as the next lemma shows. This knowledge is important for the proof of Lemma 4.15.

Lemma 4.14. Consider absorption-free l -DAG G . If $[a_1, \dots, a_n]$ is an optimal forward connected vertex elimination sequence in G then

$$I_{a_1, a_2}^G \leq I_{a_k, a_n}^{G_{k-1}} \leq I_{a_n, a_k}^{G_{k-1}},$$

where k is the largest integer with a_k and a_n are adjacent in G_{k-1} , $G_0 = G$ and $G_i = G_{i-1} - a_i$.

Proof. From the choice of k follows that a_n is not adjacent to all a_i with $i > k$. Hence the vertex elimination sequence $[a_1, \dots, a_{k-1}, a_n, a_k, \dots, a_{n-1}]$ is also an optimal elimination sequence. Therefore from Lemma 4.13 follows that $I_{a_k, a_n}^{G_{k-1}} \leq I_{a_n, a_k}^{G_{k-1}}$. So only the first part of the inequality remains to be shown. This result is proved by induction over n .

$n = 2$. $I_{a_1, a_2}^G \leq I_{a_1, a_2}^G$ follows from Lemma 4.13.

$n = 3$. Due to the fact that $[a_1, a_2, a_3]$ is forward connected, a_1 and a_2 are adjacent in G . Therefore we distinguish between two main cases $(a_1, a_2) \in E(G)$ and $(a_2, a_1) \in E(G)$. Each of these cases is divided into four subcases covering all possible situations for a_3 .

Case 1. $(a_1, a_2) \in E(G)$.

Case 1.1. $(a_2, a_3) \in E(G)$ (compare Figure 4.9(a)). Hence $k = 2$ for this case. From the fact that G is an absorption-free l -DAG follows that $P_{G_1}(a_2) = P(a_2) \dot{\cup} P(a_1) \setminus \{a_1\}$ and hence

$$I_{a_1, a_2}^G = |P(a_1)| \leq |P_{G_1}(a_2)| = I_{a_2, a_3}^{G_1}.$$

Case 1.2. $(a_3, a_2) \in E(G)$ (compare Figure 4.10(a)). $k = 2$ for this case. From $[a_1, a_2, a_3]$ is an optimal elimination sequence and Corollary 4.2 follows that $|P(a_1)| \leq |S(a_2)|$. Additionally we know that the successor set of a_2 is not influenced by the elimination of a_1 and therefore $S(a_2) = S_{G_1}(a_2)$ yielding

$$I_{a_1, a_2}^G = |P(a_1)| \leq |S(a_2)| = |S_{G_1}(a_2)| = I_{a_2, a_3}^{G_1}.$$

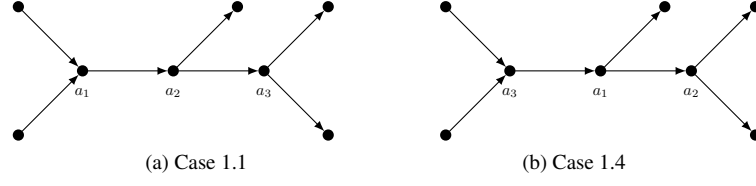


Figure 4.9: Proof of Lemma 4.14

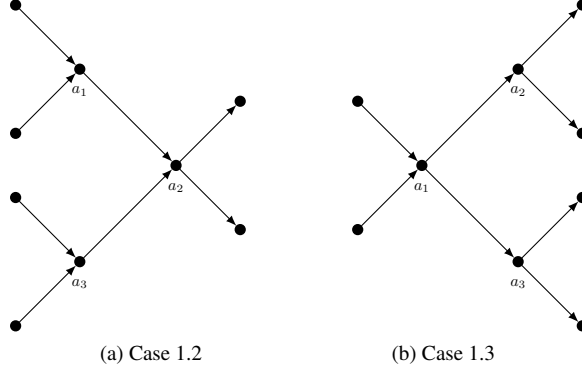


Figure 4.10: Proof of Lemma 4.14

Case 1.3. $(a_1, a_3) \in E(G)$ (compare Figure 4.10(b)). $k = 1$ for this case and therefore

$$I_{a_1, a_2}^G = |P(a_1)| = I_{a_1, a_3}^G \quad .$$

Case 1.4. $(a_3, a_1) \in E(G)$ (compare Figure 4.9(b)). $k = 2$ for this case. From $[a_1, a_2, a_3]$ is optimal elimination sequence and Corollary 4.2 follows $|P(a_1)| \leq |S(a_2)|$. The successor set of a_2 is not influenced by the elimination of a_1 and therefore $S(a_2) = S_{G_1}(a_2)$ yielding

$$I_{a_1, a_2}^G = |P(a_1)| \leq |S(a_2)| = |S_{G_1}(a_2)| \leq I_{a_2, a_3}^{G_1} \quad .$$

Case 2. $(a_2, a_1) \in E(G)$

Case 2.1. $(a_2, a_3) \in E(G)$. This case is symmetrical to **Case 1.2**.

Case 2.2. $(a_3, a_2) \in E(G)$. This case is symmetrical to **Case 1.1**.

Case 2.3. $(a_1, a_3) \in E(G)$. This case is symmetrical to **Case 1.4**.

Case 2.4. $(a_3, a_1) \in E(G)$. This case is symmetrical to **Case 1.3**

$n \mapsto n + 1$: To prove the inductive step we distinguish between three cases:

- a_n and a_{n+1} are not adjacent in G_{n-1} ,
- $(a_n, a_{n+1}) \in E(G_{n-1})$ and
- $(a_{n+1}, a_n) \in E(G_{n-1})$.

Case 1. a_n and a_{n+1} are not adjacent in G_{n-1} implying that $k < n$. Hence according to Lemma 3.7 and Lemma 3.13 $[a_1, \dots, a_{n-1}, a_{n+1}, a_n]$ is also an optimal elimination sequence. Moreover this elimination sequence is forward connected. Hence as the beginning of an optimal forward connected vertex elimination sequence, $[a_1, \dots, a_{n-1}, a_{n+1}]$ is also an optimal forward connected vertex elimination sequence. The claim follows directly from the induction assumption.

Case 2. $(a_n, a_{n+1}) \in E(G_{n-1})$. Let l be the largest integer such that a_l and a_n are adjacent in G_{l-1} . From $[a_1, \dots, a_{n+1}]$ is optimal elimination sequence of G follows that $[a_l, \dots, a_{n+1}]$ is optimal elimination sequence of G_{l-1} .

Case 2.1. The elimination sequence $[a_l, \dots, a_{n+1}]$ is forward connected in G_{l-1} . Applying the induction hypothesis to G with forward connected optimal elimination sequence $[a_1, \dots, a_n]$ and $k = l$ yield

$$I_{1,2}^G \leq I_{l,n}^{G_{l-1}}.$$

Hence it is sufficient to show that

$$I_{l,n}^{G_{l-1}} \leq I_{n,n+1}^{G_{n-1}}.$$

The latter follows immediately by applying the induction hypothesis to $G = G_{l-1}$ with forward connected optimal elimination sequence $[a_l, \dots, a_{n+1}]$ and $k = n$ finishing the proof for this case.

Case 2.2. The elimination sequence $[a_l, \dots, a_{n+1}]$ is not forward connected in G_{l-1} . Let k_0 be the first integer such that $\{a_l, a_{l+1}, \dots, a_{k_0}\}$ is not connected in G_{l-1} . According to Lemma 3.7 and Lemma 3.13

$$[a_1, \dots, a_{l-1}, a_{k_0}, a_l, \dots, a_{k_0-1}, a_{k_0+1}, \dots, a_{n+1}]$$

is also optimal elimination sequence. Moreover this sequence is also forward connected (see (*) for the proof). Hence we can apply the **Case 2** to $[a_1, \dots, a_{l-1}, a_{k_0}, a_l, \dots, a_{k_0-1}, a_{k_0+1}, \dots, a_{n+1}]$. We keep repeating this process until **Case 2.1** is applicable.

(*) To show that $[a_1, \dots, a_{l-1}, a_{k_0}, a_l, \dots, a_{k_0-1}, a_{k_0+1}, \dots, a_{n+1}]$ is forward connected, it is sufficient to show that $\{a_1, \dots, a_{l-1}, a_{k_0}\}$ is connected in G . The fact that all other sets are connected follows directly from the fact that $\{a_1, \dots, a_i\}$ is connected for all $1 < i \leq n+1$. From $[a_1, \dots, a_{n+1}]$ is forward connected in G follows that $\{a_1, \dots, a_{k_0}\}$ is connected in G . Hence there exists a path from a_1 to a_{k_0} containing only elements from $\{a_1, \dots, a_{k_0}\}$. According to the choice of k_0 this path cannot contain elements from $\{a_l, \dots, a_{k_0-1}\}$. Hence this path contains only elements of $\{a_1, \dots, a_{l-1}, a_{k_0}\}$ finishing the proof.

Case 3. $(a_{n+1}, a_n) \in E(G_{n-1})$. The proof is symmetrical to that of **Case 2**.

□

In the following we will look at elimination sequences that can contain both edge and vertex eliminations. We refer to such elimination sequences as *transformation* sequences. The length of the transformation sequence is defined by the number of eliminations it consists of. Both vertex and edge elimination count as one elimination. For example the length of the transformation sequence $[v_1, v_2]$ on the lion graph is 2, while as $[(x_1, v_1)^+, (x_2, v_1)^+, v_2]$ on the same graph has length 3 although both transformation sequences yield the same graph and even the same costs. Clearly an optimal complete transformation sequence is also a solution to the optimal edge elimination problem. The complete transformation sequence is shorter the more vertex elimination it contains. In the following lemma we prove that the shortest complete optimal transformation sequence on absorption-free graphs contains only vertex eliminations. The direct consequence of this result is that vertex elimination solves the optimal edge elimination problem on absorption-free graphs.

Lemma 4.15. *Consider an absorption-free l-DAG G . A shortest complete optimal transformation sequence contains only vertex eliminations. (Specifically it contains no pure edge eliminations).*

Proof. Proof by induction over the length n of the shortest complete optimal transformation sequence.

$n = 1$: To transform an l-DAG into a bipartite graph we need to remove all intermediate vertices. Hence each complete transformation sequence contains at least one vertex elimination (last transformation is always vertex elimination). Hence the claim is trivially true in case we have only one transformation.

$n = 2$: As already mentioned, the last transformation must be a vertex elimination. Hence let us assume that the first transformation is a pure edge elimination. This implies that the l-DAG has only one intermediate vertex. Hence we can achieve the same costs by eliminating this vertex directly without the preceding edge elimination, which is a contradiction to the assumption of the *shortest* complete optimal transformation sequence.

$n \mapsto n + 1$: Assume that $[w_1, \dots, w_{n+1}]$ is the shortest complete optimal transformation sequence on G . We set

$$G_0 = G \quad \text{and} \quad G_{i+1} = G_i - w_i \quad .$$

G_1 is absorption-free l-DAG with shortest complete transformation sequence of length n , hence according to the induction assumption $w_2, \dots, w_{n+1}$ are all vertex eliminations. To finish the proof we will show that w_1 must be a vertex elimination. Therefore we will prove by contradiction that w_1 cannot be front or back elimination of an edge.

Case 1. w_1 is a vertex elimination. Obviously the claim holds in this case.

Case 2. $w_1 = (b, a)^-$. Note $b \in Z(G_1)$ as $(b, a)^-$ is a pure edge elimination. We will now show that this case is not possible. This case is divided into three subcases.

- **Case 2.1:** $w_2 \neq a \wedge w_2 \neq b$,
- **Case 2.2:** $w_2 = b$,
- **Case 2.3:** $w_2 = a$.

The last case is the most complicated and builds the central part of this proof.

Case 2.1. $w_2 \neq a \wedge w_2 \neq b$. Hence applying the Lemma 3.9 we get the following equality

$$G_2 = G - [(b, a)^-, w_2] = G - [w_2, (b, a)^-] \quad ,$$

implying that

$$\begin{aligned} \sigma([(b, a)^-, w_2, w_3, \dots, w_{n+1}]) - \sigma([w_2, (b, a)^-, w_3, \dots, w_{n+1}]) = \\ = \sigma([(b, a)^-, w_2]) - \sigma([w_2, (b, a)^-]) \quad . \end{aligned} \quad (4.9)$$

Lets compare the costs of these sequences

$$\sigma([(b, a)^-, w_2]) = |P(b)| + |S_{G_1}(w_2)| \cdot \underbrace{|P_{G_1}(w_2)|}_{=|P(w_2)|, w_2 \neq a} = |P(b)| + |S_{G_1}(w_2)| \cdot |P(w_2)| \quad (4.10)$$

$$\sigma([w_2, (b, a)^-]) = |S(w_2)| \cdot |P(w_2)| + |P_{G-w_2}(b)| \quad . \quad (4.11)$$

To finish the proof of this case we need to explore two subcases $w_2 \notin P(b)$ and $w_2 \in P(b)$.

Case 2.1.1. $w_2 \notin P(b)$. Hence $S_{G_1}(w_2) = S(w_2)$ and $P_{G-w_2}(b) = P(b)$ implying that

$$\sigma([(b, a)^-, w_2]) = \sigma([w_2, (b, a)^-]) \quad .$$

This means that the sequence $[w_2, w_1, w_3, \dots, w_{n+1}]$ is also a complete optimal transformation sequence on G . Therefore $G - w_2$ is an absorption-free graph with shortest complete optimal elimination sequence of length n . According to the induction assumption $G - w_2$ has n intermediate vertices which is a contradiction to the fact that $w_1 = (b, a)^-$ is pure edge elimination.

Case 2.1.2. $w_2 \in P(b)$ (see Figure 4.11(a)). Therefore

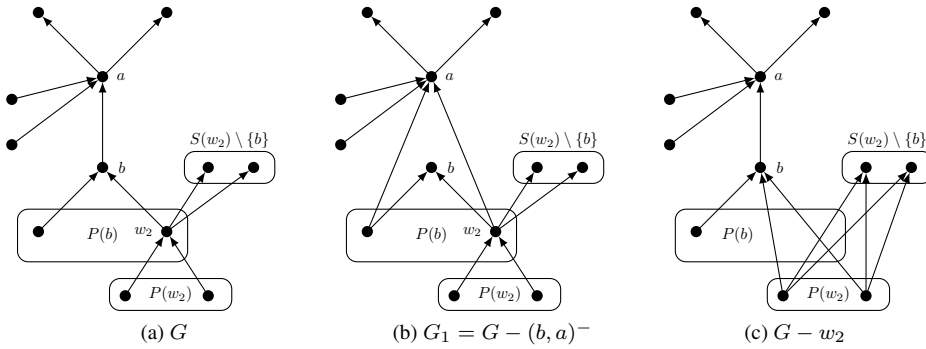


Figure 4.11: Proof of Lemma 4.15: Case 2.1.2

$$S_{G_1}(w_2) = S(w_2) \dot{\cup} \{a\} \quad (\text{see Figure 4.11(b)})$$

and

$$P_{G-w_2}(b) = (P(b) \setminus \{w_2\}) \dot{\cup} P(w_2) \quad (\text{see Figure 4.11(c)}),$$

where $\dot{\cup}$ denotes a disjoint union of two set. This implies that $|S_{G_1}(w_2)| = |S(w_2)| + 1$ and $|P_{G-w_2}(b)| = |P(b)| - 1 + |P(w_2)|$. Putting the results together we get

$$\begin{aligned} \sigma([(b, a)^-, w_2]) &\stackrel{\text{Eq(4.10)}}{=} |P(b)| + |S_{G_1}(w_2)| \cdot |P(w_2)| = |P(b)| + (|S(w_2)| + 1) |P(w_2)| \\ &= |P(b)| + |S(w_2)| \cdot |P(w_2)| + |P(w_2)| \\ &> |S(w_2)| \cdot |P(w_2)| + \underbrace{|P(b)| - 1 + |P(w_2)|}_{=|P_{G-w_2}(b)|} \stackrel{\text{Eq(4.11)}}{=} \sigma([w_2, (b, a)^-]) \quad , \end{aligned}$$

hence

$$\sigma([w_1, w_2, w_3, \dots, w_{n+1}]) > \sigma([w_2, w_1, w_3, \dots, w_{n+1}]) \quad ,$$

which is a contradiction to the fact that $[w_1, \dots, w_{n+1}]$ is a complete *optimal* transformation sequence.

Case 2.2. $w_2 = b$ (see Figure 4.12(a)). Obviously $G_2 = G - w_2$ and $\sigma([w_1, w_2]) = \sigma(w_2)$ (compare Figure 4.12(b) and Figure 4.12(c)). Hence $[w_2, \dots, w_{n+1}]$ is also complete optimal transformation sequence on G which is a contradiction to the fact that $[w_1, \dots, w_{n+1}]$ is the *shortest* complete optimal transformation sequence on G .

Case 2.3. $w_2 = a$. Consider $i_0 \in \{3, \dots, n+1\}$ such that $w_{i_0} = b$. Without loss of generality we chose the transformation sequence $[w_1, \dots, w_{n+1}]$ such that i_0 is as small as possible.

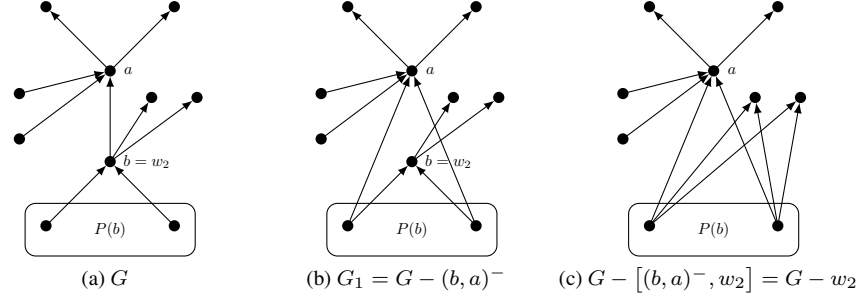


Figure 4.12: Proof of Lemma 4.15: Case 2.2

Case 2.3.1. $i_0 = 3$. From $a = w_2$ and $b = w_3$ are not adjacent in G_1 follows that

$$G - [w_1, w_2, w_3] = G - [w_1, a, b] \stackrel{\text{Lemma 3.7}}{=} G - [w_1, b, a] \stackrel{\text{Case 2.2}}{=} G - [b, a] = G - [w_3, w_2]$$

and

$$\sigma([w_1, w_2, w_3]) = \sigma([w_1, a, b]) \stackrel{\text{Lemma 3.13}}{=} \sigma([w_1, b, a]) = \sigma([w_1, w_3, w_2]) \stackrel{\text{Case 2.2}}{=} \sigma([w_3, w_2]) \quad ,$$

Hence

$$\sigma([w_1, w_2, w_3, w_4, \dots, w_{n+1}]) = \sigma([w_3, w_2, w_4, \dots, w_{n+1}]) \quad .$$

Therefore $[w_3, w_2, w_4, \dots, w_{n+1}]$ is also complete optimal transformation sequence of length $n - 1$. Which is a contradiction to the assumption fact that $[w_1, \dots, w_{n+1}]$ is a *shortest* complete optimal transformation sequence.

Case 2.3.2. $i_0 > 3$. First we discuss the case that b is the only predecessor of a (this case builds the central part of the proof). The case that a has more than one predecessor is covered at the end and is an easy to prove generalization of the first case.

Case 2.3.2.1. $P_G(a) = \{b\}$ (compare Figure 4.13(a)). The proof of this case is based on several subclaims. To improve the readability the proofs of these subclaims are placed at the end of the proof of this case. The main idea of the proof is to show that there exists an alternative transformation sequence which

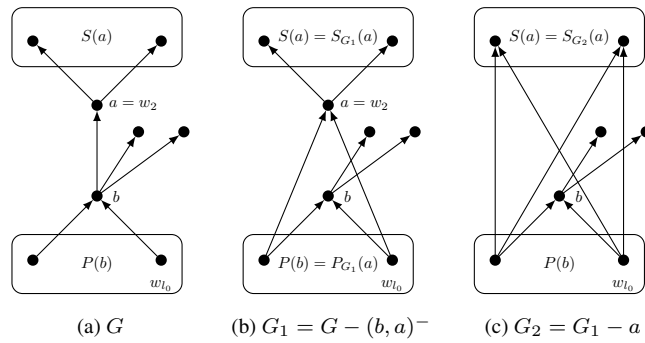


Figure 4.13: Proof of Lemma 4.15: Case 2.3.2.1

does not perform the elimination $w_1 = (b, a)^-$ and has smaller costs than that of $[w_1, \dots, w_{n+1}]$.

The transformation sequence $[w_1, \dots, w_{i_0}]$ has following properties

Subclaim 1. $[w_2, \dots, w_{i_0}]$ is forward connected in G_1 (proof on page 79).

Subclaim 2. $w_{i_0-1} \in S_{G_{i_0-2}}(b) \cup P_{G_{i_0-2}}(b)$ (proof on page 79).

Subclaim 3. There exists at least one $l_0 \in \{3, \dots, i_0 - 1\}$ with $w_{l_0} \in P_{G_{l_0-1}}(b)$ (proof on page 79).

To define an alternative transformation sequence with smaller costs than that of the $[w_1, \dots, w_n]$ we need further notations. Let l_0 be the smallest integer with $w_{l_0} \in P_{G_{l_0-1}}(b)$ (compare Subclaim 3) implying that $w_{l_0} \in P(b)$ (as discussed in Proof of Subclaim 3.). We set

$$T_i = S_{G'_i}(b) \setminus S_{G_i}(b) \quad , \quad Y = \{w_i | w_i \in T_{i-1}, 2 < i < l_0\}$$

where $G'_i = G - [w_2, \dots, w_i]$. We write y_k , $1 \leq k \leq |Y|$ for elements of Y and t_k , $1 \leq k \leq |T|$ for elements of $T = T_{l_0}$ (compare Figure 4.14(a) and Figure 4.14(b)). The reason why we are interested in the elements of T_i is that these vertices build the difference between G_i and G'_i . On G_i they are connected to the predecessors of b while on G'_i they are connected to the vertex b itself (for more details see Proof of Subclaim 4.). $T_i \neq T_{i-1}$ only if $w_i \in Y$ therefore we are also interested in this set. For easier notation

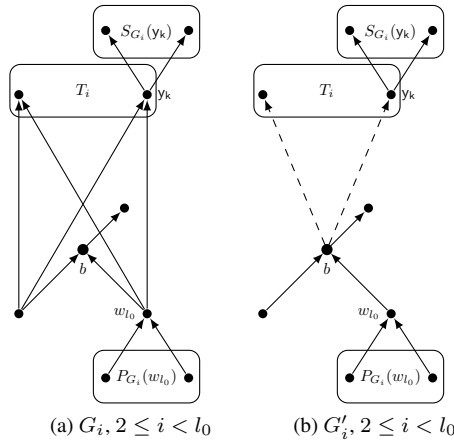


Figure 4.14: Proof of Lemma 4.15: Case 2.3.2.1: Elements of Y and T_i . (Elements of T_i are the targets of dashed (arrows) edges).

we set

- $y_k = |S_{G_{i_k-1}}(y_k)| - 1$ where $w_{i_k} = y_k$ and
- $x = |P_{G_{l_0-1}}(w_{l_0})| - 1$.

We will now show that

$$\sigma([w_2, \dots, w_{l_0}, (b, t_1)^-, \dots, (b, t_{|T|})^-, w_{l_0+1}, \dots, w_{n+1}]) < \sigma([w_1, \dots, w_{n+1}]) \quad .$$

This would be a contradiction to the assumption that $[w_1, \dots, w_{n+1}]$ is a complete *optimal* transformation sequence.

Subclaim 4. $G_{l_0} = G'_{l_0} - [(b, t_1)^-, \dots, (b, t_{|T|})^-]$ (proof on page 80).

Therefore it is sufficient to show that $\sigma([w_1, \dots, w_{l_0}]) > \sigma([w_2, \dots, w_{l_0}, (b, t_1)^-, \dots, (b, t_{|T|})^-])$. To compare the cost of these transformation sequences we need to know the costs of eliminating w_i from G_{i-1} and G'_{i-1} presented in the Subclaim 5, Subclaim 6, Subclaim 7 and Subclaim 8.

Subclaim 5. $\sigma_{G_{l_0-1}}(w_{l_0}) = (x+1)\left(|S(a)| + 1 + \sum_{k=1}^{|Y|} y_k\right)$ and $\sigma_{G'_{l_0-1}}(w_{l_0}) = x+1$.

Subclaim 6. For $Y \ni w_i = y_k$: $\sigma_{G_{i-1}}(w_i) = (y_k+1)|P(b)|$ and $\sigma_{G'_{i-1}}(w_i) = (y_k+1)$.

Subclaim 7. $w_i \notin Y \cup \{w_{l_0}\}$: $\sigma_{G_{i-1}}(w_i) = \sigma_{G'_{i-1}}(w_i)$.

Subclaim 8. $\sigma_{G'_{l_0+s-1}}((b, t_s)^-) = |P(b)| + x$, where $G'_{l_0+s-1} = G'_{l_0} - [(b, t_1)^-, \dots, (b, t_{s-1})^-]$.

Hence we get the following cost for $[w_1, \dots, w_{l_0}]$:

$$\begin{aligned}
 \sigma([w_1, \dots, w_{l_0}]) &= \sigma(w_1) + \sigma_{G_1}(w_2) + \sum_{w_i \in Y} \sigma_{G_{i-1}}(w_i) + \underbrace{\sum_{\substack{i=3 \\ w_i \notin Y}}^{l_0-1} \sigma_{G_{i-1}}(w_i) + \sigma_{G_{l_0-1}}(w_{l_0})}_{=:B} \\
 &= |P(b)| + |S(a)| \cdot |P(b)| + |P(b)| \sum_{k=1}^{|Y|} (y_k + 1) + B \\
 &\quad + (x+1)\left(|S(a)| + 1 + \sum_{k=1}^{|Y|} y_k\right) \\
 &= |P(b)| + |S(a)| \cdot |P(b)| + |Y| \cdot |P(b)| + |P(b)| \sum_{k=1}^{|Y|} y_k + B \\
 &\quad + x\left(|S(a)| + \sum_{k=1}^{|Y|} y_k\right) + x + 1 + |S(a)| + \sum_{k=1}^{|Y|} y_k.
 \end{aligned}$$

The costs of the alternative transformation sequence $[w_2, \dots, w_{l_0}, (b, t_1)^-, \dots, (b, t_{|T|})^-]$ are

$$\begin{aligned}
 &\sigma([w_2, \dots, w_{l_0}, (b, t_1)^-, \dots, (b, t_{|T|})^-]) \\
 &= \sigma(w_2) + \sum_{w_i \in Y} \sigma_{G'_{i-1}}(w_i) + \underbrace{\sum_{\substack{i=3 \\ w_i \notin Y}}^{l_0-1} \sigma_{G'_{i-1}}(w_i) + \sigma_{G'_{l_0-1}}(w_{l_0})}_{=:B(\text{Subclaim 7})} + \sum_{i=1}^{|T|} \sigma_{G'_{l_0+i-1}}((b, t_i)^-) \\
 &= |S(a)| + \sum_{k=1}^{|Y|} (y_k + 1) + B + (x+1) + \left(|S(a)| + \sum_{k=1}^{|Y|} y_k\right)(|P(b)| + x) \\
 &= |S(a)| + \sum_{k=1}^{|Y|} y_k + |Y| + B + (x+1) + x\left(|S(a)| + \sum_{k=1}^{|Y|} y_k\right) + |S(a)| \cdot |P(b)| + |P(b)| \sum_{k=1}^{|Y|} y_k.
 \end{aligned}$$

Note that $|T| = \left(|S(a)| + \sum_{k=1}^{|Y|} y_k\right)$ as $T_1 = S(a)$ and $T_i \neq T_{i-1}$ only if $w_i \in Y$ and by definition of y_k

$$|T_i| = |T_{i-1}| + |S_{G_{i-1}}(w_i)| = |T_{i-1}| + y_k.$$

$$\sigma([w_1, \dots, w_{l_0}]) - \sigma([w_2, \dots, w_{l_0}, (b, t_1)^-, \dots, (b, t_{|T|})^-]) = |P(b)| + |Y| \cdot \underbrace{|P(b)|}_{\geq 1} - |Y| \geq |P(b)| > 0.$$

This implies that $[w_1, \dots, w_{n+1}]$ is not an optimal elimination sequence leading to a contradiction. Hence the claim holds for this case.

Proof of Subclaim 1. Otherwise there exists l_0 , $2 < l_0 \leq i_0$ with $\{w_2, \dots, w_{l_0}\}$ is not connected in G_1 . W.l.o.g., let l_0 be the smallest integer with this property. Hence w_{l_0} is not adjacent to w_i for all $2 \leq i \leq l_0 - 1$. Therefore from Lemma 3.13 follows that

$$\sigma_{G_1}([w_2, \dots, w_{l_0}]) = \sigma_{G_1}([w_{l_0}, w_2, \dots, w_{l_0-1}]) \quad ,$$

while the corresponding remainder graphs are equal (Lemma 3.7). Hence

$$[w_1, w_{l_0}, w_2, \dots, w_{l_0-1}, w_{l_0+1}, \dots, w_{n+1}]$$

is also shortest complete optimal transformation sequence of G . $w_{l_0} \neq a$ (as $a = w_2$) hence we can use the proofs of Case 2.1 ($w_{l_0} \neq b$) and Case 2.2 ($w_{l_0} = b$) to show the contradiction. $\square$

Proof of Subclaim 2. Otherwise $b = w_{i_0}$ and w_{i_0-1} are not adjacent in G_{i_0-2} implying that

$$\sigma_G([w_1, \dots, w_{i_0-1}, w_{i_0}]) = \sigma_G([w_1, \dots, w_{i_0}, w_{i_0-1}]) \quad ,$$

yielding a contradiction to the choice of i_0 as the smallest integer. $\square$

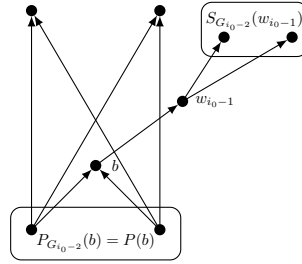


Figure 4.15: Proof of Subclaim 3 in Lemma 4.15 illustrating G_{i_0-2}

Proof of Subclaim 3. We prove this property by contradiction. Assume that $w_i \notin P_{G_{i-1}}(b)$ for all i , $2 < i < i_0$. From Subclaim 2 follows that $w_{i_0-1} \in S_{G_{i_0-2}}(b)$ (compare Figure 4.15). We also know that $P_{G_{i-1}}(b) = P(b)$ for all i , $2 < i < i_0$, as the predecessor set of b can only be changed by the elimination of one of its predecessors.

From Subclaim 1 follows that $\{w_2, \dots, w_{i_0}\}$ is connected in G_1 . Hence there exists an undirected path from $w_2 = a$ to $w_{i_0} = b$ through $\{w_2, \dots, w_{i_0}\}$ on the underlying undirected version of G_1 . This path must contain a successor of a in G_1 because $w_i \notin P_{G_1}(a) = P(b)$. The equality $P_{G_1}(a) = P(b)$ follows from the assumption $P(a) = \{b\}$ (compare Figure 4.13(b)).

Let j_0 be the smallest integer with $w_{j_0} \in S_{G_1}(a) = S(a)$ where w_{j_0} lies on an undirected path from $w_2 = a$ to $w_{i_0} = b$ through $\{w_2, \dots, w_{i_0}\}$ on the underlying undirected version of G_1 . w_{j_0} is not adjacent to $\{w_3, \dots, w_{j_0-1}\}$ as otherwise it would be a contradiction to the choice of j_0 . Hence w.l.o.g. we can assume that $j_0 = 3$ (Lemma 3.13 and Lemma 3.7). From the application of Lemma 4.14 to G_1 with forward connected vertex elimination sequence $[w_2, \dots, w_{i_0}]$ and $w_{i_0-1} \in S_{G_{i_0-2}}(b)$ (compare Figure 4.15) follows

$$|P(b)| = |P_{G_1}(a)| = |P_{G_1}(w_2)| = I_{w_2, w_3}^{G_1} \leq I_{w_{i_0-1}, w_{i_0}}^{G_{i_0-2}} \leq I_{w_{i_0}, w_{i_0-1}}^{G_{i_0-2}} = |P_{G_{i_0-2}}(b)| = |P(b)| \quad .$$

Hence $I_{w_{i_0-1}, w_{i_0}}^{G_{i_0-2}} = I_{w_{i_0}, w_{i_0-1}}^{G_{i_0-2}}$ and therefore as $w_{i_0-1} \in S_{G_{i_0-2}}(b)$

$$|S_{G_{i_0-2}}(w_{i_0-1})| = I_{w_{i_0-1}, w_{i_0}}^{G_{i_0-2}} = I_{w_{i_0}, w_{i_0-1}}^{G_{i_0-2}} = |P_{G_{i_0-2}}(b)| \quad .$$

From Corollary 4.2

$$\sigma_{G_{i_0-2}}([w_{i_0-1}, w_{i_0}]) = \sigma_{G_{i_0-2}}([w_{i_0}, w_{i_0-1}]) \quad ,$$

while the corresponding resulting 1-DAGs $G_{i_0-2} - [w_{i_0-1}, w_{i_0}]$ and $G_{i_0-2} - [w_{i_0}, w_{i_0-1}]$ are equal, which is a contradiction to the choice of i_0 and hence the desired property is proved. $\square$

Proof of Subclaim 4. First of all we compare the graphs G_2 (Figure 4.16(a)) and G'_2 (Figure 4.16(b)). Obviously $V(G_2) = V(G'_2)$. The difference in the edge sets between these graphs, is that in G_2 pre-

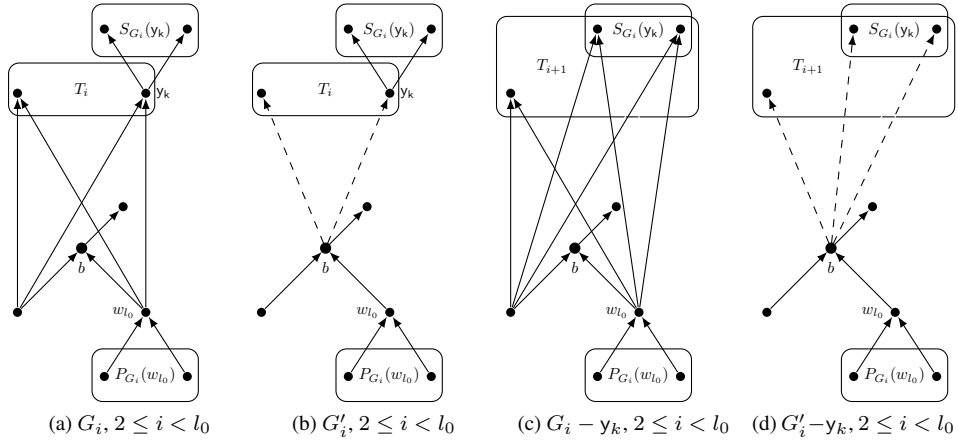


Figure 4.16: Proof of Subclaim 4 in Lemma 4.15, illustrating the difference between G_i and G'_i after elimination of y_k

decessors of b are directly connected with successors of a (elements of T_2). However in G'_2 they are connected through vertex b . Hence we can transform G'_2 in G_2 by back elimination of all edges contained in $\{b\} \times T_2 = \{(b, v) \mid v \in T_2\}$ (edges marked with dashed arrows on Figure 4.16(b)). Let us now compare G_i and G'_i for $2 < i \leq l_0$. Therefore we observe the evolution of the difference between G_2 and G'_2 during the elimination sequence $[w_3, \dots, w_{l_0}]$. Elimination of

- $w_i \notin Y \cup \{w_{l_0}\}$ removes and introduces the same edge sets on both graphs. Hence the difference between the graphs is invariant relative to this operation.
- $y_k = w_i \in Y$ simply increases the number of elements in T_i by y_k . As

$$T_i = T_{i-1} \dot{\cup} S_{G'_{i-1}}(w_i) \setminus \{w_i\}$$

in this case (compare Figure 4.16(c) and Figure 4.16(d)). Hence the structure of the differences remains the same. Meaning that after such elimination we can still transform G'_i in G_i by back elimination of all edges contained in $\{b\} \times T = \{(b, v) \mid v \in T\}$.

- w_{l_0} . This elimination increases the number of predecessors of b by x on both G_{l_0} and G'_{l_0} (compare Figure 4.17) as

$$P_{G_{l_0}}(b) = P_{G_{l_0-1}}(b) \dot{\cup} P_{G_{l_0-1}}(w_{l_0}) \setminus \{w_{l_0}\}$$

and

$$P_{G'_{l_0}}(b) = P_{G'_{l_0-1}}(b) \dot{\cup} P_{G'_{l_0-1}}(w_{l_0}) \setminus \{w_{l_0}\} \quad .$$

Hence back elimination of all edges contained in $\{b\} \times T = \{(b, v) \mid v \in T\}$ transforms G'_{l_0} in G_{l_0} .

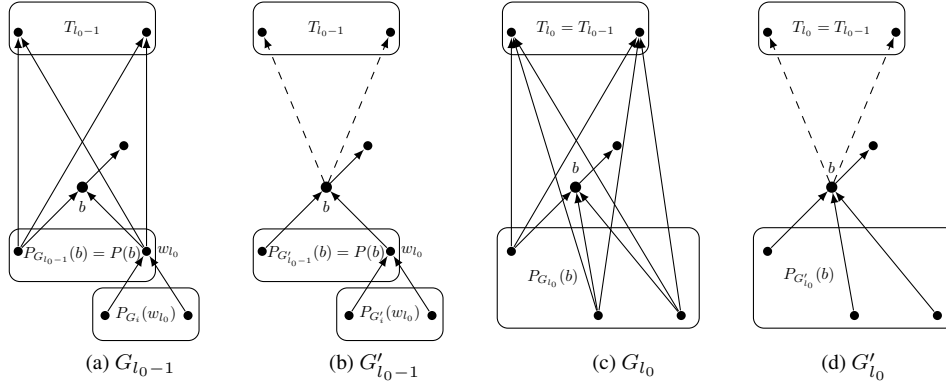


Figure 4.17: Proof of Subclaim 4 in Lemma 4.15, illustrating the difference between G_{l_0} and G'_{l_0} after elimination of w_{l_0}

□

Proof of Subclaim 5. W.l.o.g. we assume that $S_{G'_{l_0-1}}(w_{l_0}) = \{b\}$ (compare Figure 4.17(b)). This assumption is equivalent to $S_{G_{l_0-1}}(w_{l_0}) = \{b\} \cup T_{l_0-1}$ (compare Figure 4.17(a)). The existence of other successors of w_{l_0} would yield the same additional costs for the elimination of w_{l_0} on both G_{l_0-1} and G'_{l_0-1} . As this result will be used only to compare the costs of the two transformation sequences we can safely make this assumption.

To compute the costs of the elimination of w_{l_0} we need to know the number of elements in its predecessor and successor sets. From definition of x follows that

$$|P_{G_{l_0-1}}(w_{l_0})| = |P_{G'_{l_0-1}}(w_{l_0})| = x + 1 \quad .$$

Next we examine the successor sets of w_{l_0} in G_{l_0-1} and G'_{l_0-1} . From the assumption $S_{G'_{l_0-1}}(w_{l_0}) = \{b\}$ follows that

$$|S_{G'_{l_0-1}}(w_{l_0})| = 1 \quad ,$$

and hence

$$\sigma_{G'_{l_0-1}}(w_{l_0}) = |P_{G'_{l_0-1}}(w_{l_0})| \cdot |S_{G'_{l_0-1}}(w_{l_0})| = x + 1$$

To count the number of successors of w_{l_0} in G_{l_0-1} is a bit more complicated. Therefore let us look at the development of this number during the elimination sequence. First of all due to the choice of l_0 as the smallest integer with $w_{l_0} \in P_{G_{l_0-1}}(b)$, w_{l_0} is also a predecessor of b in G . From applying Lemma 4.3 to the assumption $S_{G'_{l_0-1}}(w_{l_0}) = \{b\}$ follows that $S(w_{l_0}) = \{b\}$ (compare Figure 4.13)

$$|S_{G_2}(w_{l_0})| = |S(a) \cup \{b\}| = |S(a)| + 1 \quad .$$

From definition of the vertex elimination follows that the elimination of $w_i \notin Y \cup \{b\}$ does not change the successor set of w_{l_0} . However the elimination of $w_i = y_k \in Y$ connects w_{l_0} with all successors of w_i and

then removes w_i (Figure 4.16(a) and Figure 4.16(c)). Hence the number of successors of w_{l_0} increases by y_k each time y_k is eliminated, in all other cases the successor set of w_{l_0} is not altered (remember that $w_i \neq b$ for all $2 \leq i \leq l_0$), yielding

$$|S_{G_{l_0-1}}(w_{l_0})| = |S(a)| + 1 + \sum_{k=1}^{|Y|} y_k .$$

Hence the costs to eliminate the vertex w_{l_0} on G_{l_0-1} are

$$\sigma_{G_{l_0-1}}(w_{l_0}) = |P_{G_{l_0-1}}(w_{l_0})| \cdot |S_{G_{l_0-1}}(w_{l_0})| = (x+1) \left(|S(a)| + 1 + \sum_{k=1}^{|Y|} y_k \right) .$$

□

Proof of Subclaim 6. Let us now discuss the costs of eliminating of $w_i = y_k \in Y$. Using the same argument as in proof of Subclaim 5 w.l.o.g. we assume that $P_{G'_{i-1}}(y_k) = \{b\}$ (compare Figure 4.16(b)). This assumption is equivalent to $P_{G_{i-1}}(y_k) = P_{G_{i-1}}(b) = P_{G'_{i-1}}(b)$ (compare Figure 4.16(a)).

From the choice of l_0 as the smallest integer with $w_{l_0} \in P_{G_{l_0}}(b)$ follows

$$P_{G'_{i-1}}(b) = P(b)$$

for all $2 < i < l_0$. Therewith

$$\sigma_{G_{i-1}}(w_i) = \sigma_{G_{i-1}}(y_k) = |S_{G_{i-1}}(y_k)| \cdot \underbrace{|P_{G_{i-1}}(y_k)|}_{=P(b)} = |S_{G_{i-1}}(y_k)| \cdot |P(b)|$$

and

$$\sigma_{G'_{i-1}}(w_i) = \sigma_{G'_{i-1}}(y_k) = |S_{G'_{i-1}}(y_k)| \cdot \underbrace{|P_{G'_{i-1}}(y_k)|}_{=\{b\}} = |S_{G'_{i-1}}(y_k)| .$$

From the definition of y_k and y_k follows that

$$|S_{G_{i-1}}(y_k)| = |S_{G'_{i-1}}(y_k)| = y_k + 1 .$$

Hence

$$\sigma_{G_{i-1}}(w_i) = \sigma_{G_{i-1}}(y_k) = (y_k + 1)|P(b)| \quad \text{and} \quad \sigma_{G'_{i-1}}(w_i) = \sigma_{G'_{i-1}}(y_k) = (y_k + 1) .$$

□

Proof of Subclaim 7. As discussed in Proof of Subclaim 4. on page 80, w_i has the same successor [predecessor] set in G_{i-1} as in G'_{i-1} and therefore

$$\sigma_{G_{i-1}}(w_i) = |P_{G_{i-1}}(w_i)| \cdot |S_{G_{i-1}}(w_i)| = |P_{G'_{i-1}}(w_i)| \cdot |S_{G'_{i-1}}(w_i)| = \sigma_{G'_{i-1}}(w_i) .$$

□

Proof of Subclaim 8. The choice of l_0 as the smallest integer with $w_{l_0} \in P(b)$ yields $P_{G'_i}(b) = P(b)$ for all $2 < i < l_0$, therewith $P_{G'_{l_0}}(b) = P(b) \dot{\cup} P_{G'_{l_0-1}}(w_{l_0}) \setminus w_{l_0}$. (compare Figure 4.17(b) and Figure 4.17(d)). Hence as $|P_{G'_{l_0-1}}(w_{l_0})| = x + 1$

$$|P_{G'_{l_0}}(b)| = |P(b)| + |P_{G'_{l_0-1}}(w_{l_0})| - 1 = |P(b)| + x$$

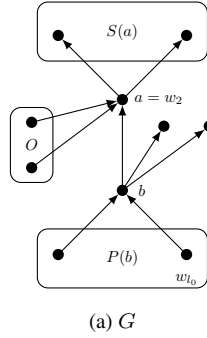


Figure 4.18: Proof of Lemma 4.15: Case 2.3.2.2

The back elimination of edge (b, t_l) does not change the predecessor set of b hence

$$\sigma_{G_{i_0+s-1}}((b, t_s)^-) = |P_{G_{i_0}}(b)| = |P(b)| + x \quad .$$

□

Case 2.3.2.2. $P(a) \setminus \{b\} \neq \emptyset$ (compare Figure 4.18) We set $O = P(a) \setminus \{b\}$ and denote the elements of O with o_i for $1 \leq i \leq |O|$. It is easy to see that

$$G_2 = G - [w_1, w_2] = G - [(o_1, a)^+, \dots, (o_{|O|}, a)^+, w_1, w_2]$$

and

$$\sigma([w_1, w_2]) = \sigma([(o_1, a)^+, \dots, (o_{|O|}, a)^+, w_1, w_2]) \quad ,$$

implying that

$$\sigma([w_1, \dots, w_{n+1}]) = \sigma([(o_1, a)^+, \dots, (o_{|O|}, a)^+, w_1, \dots, w_{n+1}]) \quad .$$

We set $G' = G - [(o_1, a)^+, \dots, (o_{|O|}, a)^+]$. $P_{G'}(a) = \{b\}$ hence G' fulfills all prerequisites from **Case 2.3.2.1**. Therefore from **Case 2.3.2.1** follows that $[w_1, \dots, w_{n+1}]$ is not optimal elimination sequence for G' . Hence $[(o_1, a)^+, \dots, (o_{|O|}, a)^+, w_1, \dots, w_{n+1}]$ is not an *optimal* elimination sequence of G . Leading to a contradiction and finishing the proof for this case.

Case 3. $w_1 = (b, a)^+$. Analogous to **Case 2** due to symmetry.

□

The fact that VE and EE are equivalent on absorption-free l-DAGs is a direct consequence of the previous result.

Theorem 4.4. Consider absorption-free l-DAG G .

$$\text{VE}(G) = \text{EE}(G)$$

Proof. Follows immediately from Lemma 4.15

□

The result that VE and EE are comparable on absorption-free l-DAGs is not only interesting from the theoretical point of view as shown in the next subsection.

4.3.2 Optimality preserving eliminations and critical degree

In this subsection we talk about different optimality preserving eliminations on absorption-free graphs. In general we deal with two different approaches. The first one explores in which situations the elimination of a vertex with only one in or one out edge preserves optimality (compare Lemma 4.16 and Lemma 4.17). This result is a generalization of the result presented in [MN12]. The generalization is a consequence of $EE = VE$ on absorption-free l-DAGs. The second approach presented in this subsection is a more generic one. It is based on the idea of estimating the contribution of an edge to the overall cost. If we find a possibility to transform the given l-DAG by edge eliminations into a subgraph using the same number of fmas as in our estimation then these eliminations preserve optimality. In contrast to the MEC problem these results do not allow us to solve the EE problem for arbitrary absorption-free l-DAGs. Still we present an algorithm solving the EE problem for absorption-free l-DAGs with maximal path length 3. The most important result from this subsection is an under estimator for the contribution of an edge to the overall costs of any edge (vertex) elimination sequence (Theorem 4.5). It not only allows optimality preserving eliminations to be performed but it also builds the base for the algorithms computing lower bounds on absorption-free developed in this thesis. Thus it is implicitly used in all algorithms for lower bounds.

This subsection has the following structure. First we prove the optimality preserving eliminations based on elimination of a vertex with only one in- or one out-edge in Lemma 4.16 and Lemma 4.17. The algorithm exploiting this results is presented in Algorithm 5. Afterwards the second approach is discussed. The definition of the contribution of an edge to the overall costs is shown in Definition 4.5. The general idea of the second approach is proved in Lemma 4.18. The under estimator for the contribution of a edge to the overall costs is proved in Lemma 4.19. For intermediate edge in Lemma 4.20. Theorem 4.5 combines both under estimators in one formula. In Lemma 4.21 and Lemma 4.22 we develop optimality preserving eliminations based on the previous results which are then used in Algorithm 6 and Algorithm 7. Finally in Lemma 4.23 and Algorithm 8 we prove and present an algorithm that solves the EE problem on absorption-free l-DAGs with maximal path length 3.

In [MN12] we proved that on absorption-free l-DAGs if vertex a is the only predecessor of vertex b and b the only successor of a the elimination of the vertex with the smaller bridge degree preserves optimality. We refer to such edges (a, b) (where a is the only predecessor of vertex b and b the only successor of a) as *bridges*. For example in l-DAG G as on Figure 4.7(a) the edge (v_1, v_2) is a bridge. The bridge degree is $I_{v_1, v_2}^G = |P(v_1)| = 4 > 3 = |S(v_2)| = I_{v_2, v_1}^G$. Hence the elimination of v_2 preserves optimality. On the resulting l-DAG $G - v_2$ (compare Figure 4.7(b)) we cannot perform any optimality preserving elimination because v_3 is not the only successor of v_1 . Using the fact that $VE(G) = EE(G)$ on absorption-free graphs we can extend the results of [MN12] such that they are applicable in this case too.

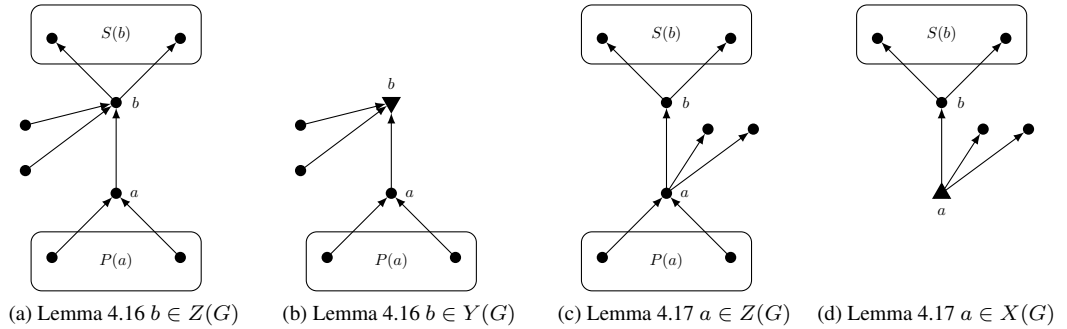


Figure 4.19: Illustration of the assumptions in Lemma 4.16 and Lemma 4.17

Lemma 4.16. Consider absorption-free l -DAG G . $(a, b) \in E^*(G)$, $a \in Z(G)$ with $S(a) = \{b\}$. If $|P(a)| \leq |S(b)|$ or $b \in Y(G)$ then

$$\text{VE}(G) = \text{VE}(G - a) + \sigma_G(a) \quad ,$$

implying that the elimination of a preserves optimality (for illustration compare Figure 4.19(a) for $b \in Z(G)$ and Figure 4.19(b) for $b \in Y(G)$).

Proof. Consider $[v_1, \dots, v_n]$ is an optimal vertex elimination sequence of G . We set $G_0 = G$ and $G_i = G_{i-1} - v_i$.

Case 1. a is eliminated before b . (Note this case also covers the case when $b \in Y(G)$). Let $v_i = a$. We compare the costs for the elimination sequences $[v_1, \dots, v_{i-1}, a]$ and $[a, v_1, \dots, v_{i-1}]$. Since a has only one successor, the costs for the elimination of the v_j , $1 \leq j \leq i-1$ are the same in both elimination sequences. Hence we only have to compare the costs for the elimination of a in both sequences. From Lemma 4.3 follows that

$$\sigma_G(a) = |P_G(a)| \leq |P_{G_{i-1}}(a)| = \sigma_{G_{i-1}}(a) \quad .$$

According to Lemma 3.7 both elimination sequences yield the same remainder graph and therefore

$$\sigma_G([a, v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n]) \leq \sigma_G([v_1, \dots, v_n])$$

finishing the proof for this case.

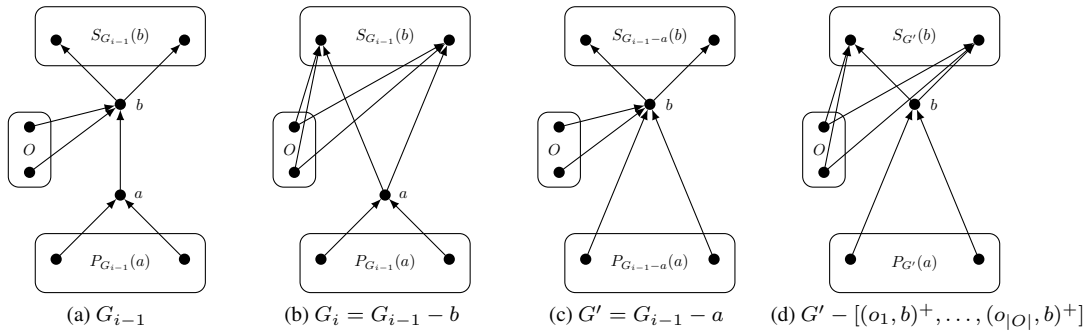


Figure 4.20: Proof of **Case 2** in Lemma 4.16. G_i and $G - [v_1, \dots, v_{i-1}, a, (o_1, b)^+, \dots, (o_{|O|}, b)^+]$ are isomorphic

Case 2. b is eliminated before a . Let $v_i = b$. We set $O = P_{G_{i-1}}(b) \setminus \{a\}$ and denote the elements of O with o_j for $1 \leq j \leq |O|$. Let us compare the costs of the following two elimination sequences:

$$[v_1, \dots, v_{i-1}, b, v_{i+1}, \dots, v_n] \quad \text{and} \quad [a, v_1, \dots, v_{i-1}, (o_1, b)^+, \dots, (o_{|O|}, b)^+, v_{i+1}, \dots, v_n] \quad .$$

Obviously G_i and $G_{i-1} - [a, (o_1, b)^+, \dots, (o_{|O|}, b)^+]$ are isomorphic (compare Figure 4.20). Therefore according to Lemma 3.7 G_i and $G - [a, v_1, \dots, v_{i-1}, (o_1, b)^+, \dots, (o_{|O|}, b)^+]$ are isomorphic. Hence we need to compare the costs only of the first part of these elimination sequences namely

$$[v_1, \dots, v_{i-1}, b] \quad \text{and} \quad [a, v_1, \dots, v_{i-1}, (o_1, b)^+, \dots, (o_{|O|}, b)^+] \quad .$$

As already discussed in **Case 1** a has only one successor and therefore

$$\sigma_G([v_1, \dots, v_{i-1}]) = \sigma_{G-a}([v_1, \dots, v_{i-1}]) \quad .$$

Hence it is sufficient to compare the costs of simple elimination of vertex b in l-DAG G_{i-1} and the elimination of vertex a on G together with front elimination of the edges connecting the set O with vertex b in l-DAG $G' = G - [a, v_1, \dots, v_{i-1}]$ (compare Figure 4.20):

$$\sigma_G(a) + \sigma_{G'}([(o_1, b)^+, \dots, (o_{|O|}, b)^+]) = |P_G(a)| + |O| \cdot \underbrace{|S_{G'}(b)|}_{=|S_{G_{i-1}}(b)|} = |P_G(a)| + |O| \cdot |S_{G_{i-1}}(b)| \quad ,$$

$$\sigma_{G_{i-1}}(b) = |P_{G_{i-1}}(b)| \cdot |S_{G_{i-1}}(b)| = (1 + |O|)|S_{G_{i-1}}(b)| = |S_{G_{i-1}}(b)| + |O| \cdot |S_{G_{i-1}}(b)| \quad .$$

From Lemma 4.3 and the assumption that $|P_G(a)| \leq |S_G(b)|$ follows that $|P_G(a)| \leq |S_{G_{i-1}}(b)|$ and therefore

$$\sigma_G(a) + \sigma_{G'}([(o_1, b)^+, \dots, (o_{|O|}, b)^+]) - \sigma_{G_{i-1}}(b) = |P_G(a)| - |S_{G_{i-1}}(b)| \leq 0 \quad .$$

Hence $\text{VE}(G) \geq \sigma_G(a) + \text{EE}(G - a)$. As G is absorption-free l-DAG, so is $G - a$ (Lemma 4.1) and hence from Theorem 4.4 follows that $\text{EE}(G - a) = \text{VE}(G - a)$ implying

$$\text{VE}(G) = \sigma_G(a) + \text{VE}(G - a) \quad ,$$

finishing the proof. □

Lemma 4.17. *Consider absorption-free l-DAG G . $(a, b) \in E^*(G)$, $b \in Z(G)$ with $P(b) = \{a\}$. If $|S(b)| \leq |P(a)|$ or $a \in X(G)$, then*

$$\text{VE}(G) = \text{VE}(G - b) + \sigma_G(b) \quad ,$$

implying that the elimination of b preserves optimality (for illustration see Figure 4.19(c) for $a \in Z(G)$ and Figure 4.19(d) for $a \in X(G)$).

Proof. Symmetrical to the proof of Lemma 4.16. □

We refer to the edges (a, b) as described in prerequisites of Lemma 4.16 and Lemma 4.17 as *semi-bridges*. Using the optimality preserving elimination for semi-bridges we can solve the vertex elimination problem on the l-DAG G from Figure 4.7(a). At the beginning of this subsection we already asserted that the elimination of vertex v_2 preserves optimality but we had no idea how to solve the vertex elimination problem on the resulting l-DAG $G - v_2$ (compare Figure 4.7(b)). Using Lemma 4.17 we know that elimination of v_3 preserves optimality as v_3 is the only successor of v_1 and $|S(v_2)| = 2 < 4 = |P(v_1)|$. Hence the elimination sequence $[v_2, v_3, v_1]$ solves the vertex elimination problem on this l-DAG.

The following algorithm performs for a given l-DAG G all possible optimality preserving vertex eliminations according to Lemma 4.16 and Lemma 4.17 and computes their costs. (Please note: this algorithm takes a variable costs as an input. It computes incremental costs because it will be used as part of other algorithms later where some eliminations could have been performed prior to running Algorithm 5).

Please note: Algorithm 5 is meant to demonstrate one way of performing all possible optimality preserving vertex eliminations according to Lemma 4.16 and Lemma 4.17 and therefore it was not optimized in terms of computation complexity.

Algorithm 5 Optimality preserving eliminations according to Lemma 4.16 and Lemma 4.17

Require: absorption-free l-DAG G , costs**Ensure:** G , costs

```

1: changed  $\leftarrow$  true                                // Indicates whether elimination has been performed
2: while changed = true do
3:   changed  $\leftarrow$  false
4:    $M_1 = \{a \in Z(G) \mid |P(a)| = 1\}$                 // Candidates for elimination according to Lemma 4.17
5:   while  $M_1 \neq \emptyset$  do
6:     choose  $(v, a)$  with  $a \in M_1$ 
7:     if  $|S(a)| \leq |P(v)|$  or  $v \in X(G)$  then          // Lemma 4.17 is applicable
8:        $G \leftarrow G - a$ 
9:       costs  $\leftarrow$  costs +  $|S(a)|$ 
10:      changed  $\leftarrow$  true
11:    end if
12:     $M_1 \leftarrow M_1 \setminus \{a\}$ 
13:  end while
14:   $M_2 = \{a \in Z(G) \mid |S(a)| = 1\}$                 // Candidates for elimination according to Lemma 4.16
15:  while  $M_2 \neq \emptyset$  do
16:    choose  $(a, v)$  with  $a \in M_2$ 
17:    if  $|P(a)| \leq |S(v)|$  or  $v \in Y(G)$  then          // Lemma 4.16 is applicable
18:       $G \leftarrow G - a$ 
19:      costs  $\leftarrow$  costs +  $|P(a)|$ 
20:      changed  $\leftarrow$  true
21:    end if
22:     $M_2 \leftarrow M_2 \setminus \{a\}$ 
23:  end while
24: end while

```

In the algorithm we first build the set of candidates for optimality preserving eliminations according to Lemma 4.17. For each vertex in this set we check whether it fulfils the prerequisites of the Lemma 4.17, if so, we perform the elimination. As the eliminated vertex has only one predecessor, its elimination does not change the number of predecessors for any other vertex. Therefore we do not have to update the set of candidates inside the loop. Similarly we do not need to update the set of candidates (according to Lemma 4.16) in the second loop. The variable `changed` is used to identify the point when no further eliminations can be performed. It is easy to see that a worst case complexity of Algorithm 5 is $O(|V(G)|)$ as sets M_1 and M_2 are strictly decreasing (compare Lemma 4.3). It does not matter in which order the optimality preserving eliminations according to Lemma 4.16 and Lemma 4.17 are performed. As already mentioned the optimality preserving elimination according Lemma 4.17 does not change the predecessor set of any vertex with only one predecessor. So it cannot destroy any existing optimality preserving situation according to Lemma 4.17. It also cannot destroy any existing optimality preserving eliminations according to Lemma 4.16, as only the sizes of the successor sets are influenced by such an elimination.

The next optimality preserving approach is a more generic one. It can be applied on arbitrary l-DAGs. The idea behind this approach is to identify the “contribution” of a given edge to the overall costs in an optimal elimination sequence. First of all we need to define the “contribution” of an edge.

Definition 4.5. Consider an l-DAG G . An edge $(i, j) \in E^*(G)$ is called n -critical for vertex [edge] elimination if and only if

$$\text{VE}(G) - \text{VE}(G') \geq n \quad (\text{EE}(G) - \text{EE}(G') \geq n) \quad ,$$

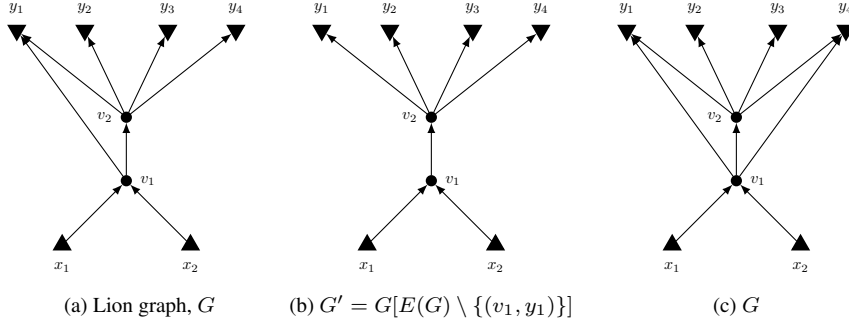


Figure 4.21: Comparing the critical degree for edge and vertex elimination (illustration of Example 4.8).

where $G' = G[E(G) \setminus \{(i, j)\}]$. For $n = 1$ we say that (i, j) is critical. We set

$$\Delta_G^e((i, j)) = \text{EE}(G) - \text{EE}(G') \quad \text{and} \quad \Delta_G^v((i, j)) = \text{VE}(G) - \text{VE}(G') \quad ,$$

and refer to them as critical degree of (i, j) in G for edge or vertex elimination respectively.

The critical degree is the “contribution” of an edge to an optimal edge (vertex) elimination sequence. Note that on absorption-free l-DAGs $\Delta_G^e((i, j)) = \Delta_G^v((i, j))$ as a result from Theorem 4.4. Therefore to make the notation easier we use $\Delta_G((i, j))$ to denote the critical degree of an edge throughout the Chapter 4. In general the critical degree for edge elimination can be greater or smaller than the critical degree for vertex elimination as shown in the next example.

Example 4.8. Assume that G is the lion graph as on Figure 4.21(a). Let us now compute the critical degree of the edge (v_1, y_1) . We already know that $\text{EE}(G) = 11$ while $\text{VE}(G) = 12$. $G' = G[E(G) \setminus \{(v_1, y_1)\}]$ is an absorption-free l-DAG (compare Figure 4.21(b)). Hence $\text{EE}(G') = \text{VE}(G') = 10$ yielding

$$1 = \Delta_G^e(v_1, y_1) = \underbrace{\text{EE}(G)}_{=11} - \underbrace{\text{EE}(G')}_{=\text{VE}(G')} < \underbrace{\text{VE}(G)}_{=12} - \text{VE}(G') = \Delta_G^v(v_1, y_1) = 2$$

Let us now discuss an l-DAG where $\Delta_G^e(i, j) > \Delta_G^v(i, j)$. Therefore consider an l-DAG G as on Figure 4.21(c). Obviously

$$\sigma([v_1, v_2]) = 6 + 8 = 14 \quad \text{and} \quad \sigma([v_2, v_1]) = 4 + 8 = 12 \quad ,$$

and hence $\text{VE}(G) = 12$. As shown in Example 6.23 the costs of the complete optimal edge elimination sequence are also 12 meaning that $\text{EE}(G) = \text{VE}(G)$. Let us now determine the critical degree of the edge (v_1, y_1) . Therefore we set $G' = G[E(G) \setminus \{(v_1, y_1)\}]$. G' is a lion graph hence $\text{VE}(G') = 12$ while as $\text{EE}(G') = 11$ yielding

$$1 = \Delta_G^e(v_1, y_1) = \underbrace{\text{EE}(G)}_{=12} - \underbrace{\text{EE}(G')}_{=11} > \underbrace{\text{VE}(G)}_{=12} - \underbrace{\text{VE}(G')}_{=12} = \Delta_G^v(v_1, y_1) = 0 \quad .$$

As already mentioned, one can see the critical degree of an edge as a lower bound for the contribution of this edge to the overall costs of any complete optimal elimination sequence applied to the l-DAG. In the next lemma we show the usage of critical degree as a generic tool for finding optimality preserving eliminations.

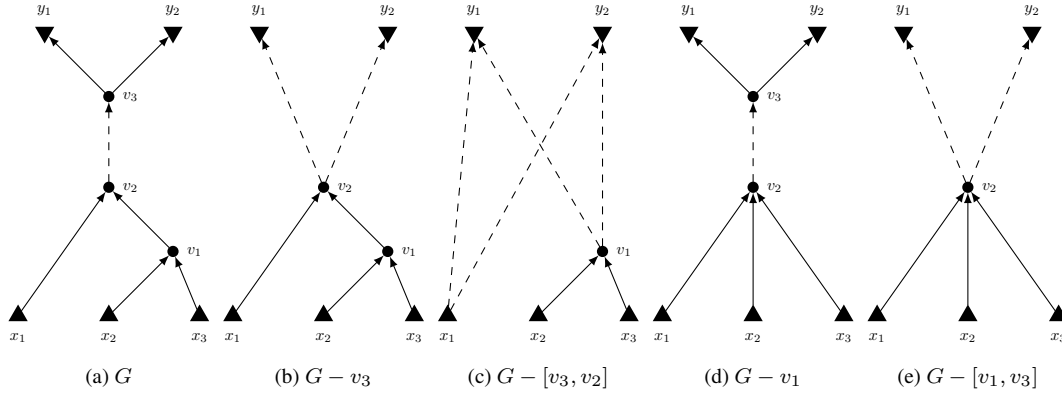


Figure 4.22: Cost contribution of an edge (illustrating Example 4.9).

Lemma 4.18. *If $(i, j) \in E^*(G)$ is n -critical and if there exists an elimination sequence transforming G into G' , where $G' \subseteq G$ and $(i, j) \notin E(G')$, at cost of n , then this elimination sequence preserves optimality.*

Proof. The proof follows immediately from the Definition 4.5. □

To visualize the contribution of an edge to a given elimination sequence one can use bi-coloring. Color this edge with 1 and all other edges with 0. Apply the elimination sequence to this bi-colored l-DAG. Count the number of multiplications involving labels of edges colored with 1. This number is the contribution of the edge to the given elimination sequence. Lets discuss it in a bit more detail in the following example.

Example 4.9. *Consider l-DAG G as on Figure 4.22(a). Let us now observe the contribution of the edge (v_2, v_3) to the overall costs of the following two vertex elimination sequences*

- $[v_3, v_2, v_1]$ and
- $[v_1, v_3, v_2]$.

First we discuss the vertex elimination sequence $[v_3, v_2, v_1]$. The elimination of v_3 involves two multiplications with a 1-colored edge. Although after this elimination the edge (v_2, v_3) is no longer present in the resulting l-DAG, $G - v_3$ still contains 1-colored edges (compare Figure 4.22(b)). Meaning that the impact of (v_2, v_3) is still present after this elimination. The elimination of v_2 involves four multiplications with 1-colored edges. Hence, up till now the edge (v_2, v_3) contributed 6 multiplications. Elimination of v_1 from $G - [v_3, v_2]$ yield another four multiplications with 1-colored edges (edges induced by the existence of edge (v_2, v_3)). Therefore the overall impact of (v_2, v_3) on number of multiplications in $[v_3, v_2, v_1]$ is 10.

Let us now look at the second elimination sequence $[v_1, v_3, v_2]$. The elimination of vertex v_1 does not involve any multiplications with (v_2, v_3) (compare Figure 4.22(a)). The resulting l-DAG is shown in Figure 4.22(d). The elimination of v_3 yields two multiplications with the 1-colored edge. The elimination of v_2 from the resulting graph remainder adds another six multiplications with 1-colored edges (compare Figure 4.22(e)). Hence, the overall impact of edge (v_2, v_3) on $[v_1, v_3, v_2]$ is 8.

To be able to apply the result from Lemma 4.18 we need some criteria to determine a critical degree of an edge. The problem is that we typically need the solution of EE (VE) problem to determine its critical

degree. Which is counter productive if we want to solve this problem. Still we can try to find some good lower bounds for the critical degree. In Example 4.9 we saw that the impact of edge is present as long as we have 1-colored edges in the l-DAG. From the definition of edge and vertex elimination on bi-colored l-DAGs we know that 1-colored edges can disappear only through absorption. Which is impossible in absorption-free graphs. Hence we have to propagate the 1-colored edges through the l-DAG until all of them connect the inputs with the outputs to completely remove the impact of a given edge. As we already saw in the Example 4.9, the impact of edge on the given edge (vertex) elimination sequence is smaller if we remove the vertices which are incident to this edge at the very end of the elimination sequence. This is the idea behind the lower bounds for the critical degree of an edge presented in the next two lemmas. Although the idea is easy to understand and one could even believe this statement without a proof, the rigorous proofs are long as they have to cover a lot of cases. Still the proofs are straight forward and do not contain any specific tricks or ideas.

We prove these results only for the vertex elimination and use Theorem 4.4 to ensure that the claim holds also for edge elimination. We could also prove the same lower bounds for the edge elimination directly. This proof is similar but significantly longer as we have to cover more cases. First we prove the lower bound for non-intermediate edges.

Lemma 4.19. *Consider absorption-free l-DAG G , $(a, b) \in \{(i, j) \in E^*(G) \mid i \in X(G) \vee j \in Y(G)\}$. For any complete vertex elimination sequence $[v_1, \dots, v_n]$ holds*

$$\sigma([v_1, \dots, v_n]) \geq \sigma_{G'}([v_1, \dots, v_n]) + |X(a)| \cdot |Y(b)| \quad ,$$

where G' is a subgraph of G with $(a, b) \notin E(G')$.

Proof. We set $G_i = G - [v_1, \dots, v_i]$ and $G'_i = G' - [v_1, \dots, v_i]$. Proof by induction over the length $n = |Z(G)|$ of the elimination sequence.

$n = 1$:

Case 1. $b \in Z(G)$. Hence $a \in X(G)$. From the definition of vertex elimination follows that b can be eliminated from G by front elimination of all its incoming edges, yielding

$$\begin{aligned} \sigma(b) &= \sum_{v \in P(b)} \sigma((v, b)^+) = \sum_{v \in P(b) \setminus \{a\}} \sigma((v, b)^+) + \sigma((a, b)^+) \\ &= \sum_{v \in P(b) \setminus \{a\}} |S(b)| + |S(b)| \geq \sum_{v \in P_{G'}(b)} |S_{G'}(b)| + \underbrace{|S(b)|}_{=|Y(b)|} \\ &= \sum_{v \in P_{G'}(b)} \sigma_{G'}((v, b)^+) + \underbrace{|X(a)|}_{=1} |Y(b)| = \sigma_{G'}(b) + |X(a)| \cdot |Y(b)| \quad . \end{aligned}$$

Hence the claim holds for this case.

Case 2. $a \in Z(G)$. This case is symmetrical to **Case 1**.

$n \mapsto n + 1$:

Case 1. $b \in Z(G)$. Hence $a \in X(G)$.

Case 1.1. $v_1 \neq b$. By induction assumption follows

$$\sigma_{G-v_1}([v_2, \dots, v_n]) \geq \sigma_{G'-v_1}([v_2, \dots, v_n]) + |X_{G-v_1}(a)| \cdot |Y_{G-v_1}(b)| \quad .$$

As $a \in X(G)$ and $v_1 \neq b$ we know that

$$|X_{G-v_1}(a)| = |X(a)| = 1 \quad \text{and} \quad |Y_{G-v_1}(b)| = |Y(b)| \quad ,$$

and as from Lemma 3.6 follows that $\sigma(v_1) \geq \sigma_{G'}(v_1)$ we get

$$\begin{aligned} \sigma([v_1, \dots, v_n]) &= \sigma(v_1) + \sigma_{G-v_1}([v_2, \dots, v_n]) \\ &\geq \sigma_{G'}(v_1) + \sigma_{G'-v_1}([v_2, \dots, v_n]) + |X_{G-v_1}(a)| \cdot |Y_{G-v_1}(b)| \\ &= \sigma_{G'}([v_1, \dots, v_n]) + |X(a)| \cdot |Y(b)| \quad . \end{aligned}$$

Hence the claim holds for this case.

Case 1.2. $v_1 = b$. Hence $a \in X(G)$ and $|X(a)| = 1$. From the definition of vertex elimination follows that $v_1 (= b)$ can be eliminated from G by subsequent front elimination of all its incoming edges. Taking into account that $(a, b) \notin E(G')$ we get

$$\begin{aligned} \sigma(v_1) &= \sum_{v \in P(b)} \sigma((v, b)^+) = \sum_{v \in P(b) \setminus \{a\}} \sigma((v, b)^+) + \sigma((a, b)^+) \\ &= \sum_{v \in P(b) \setminus \{a\}} \sigma((v, b)^+) + |S(b)| \geq \sum_{v \in P_{G'}(b)} \sigma_{G'}((v, b)^+) + |S(b)| \\ &= \sigma_{G'}(v_1) + |S(b)| \quad . \end{aligned} \tag{4.12}$$

After the elimination of $v_1 (= b)$ the l-DAG $G - v_1$ can contain edges that are induced (a, b) meaning that we have not considered the full contribution of the edge (a, b) so far. The l-DAG $G - v_1$ contains such edges if $S = S(b) \setminus Y(b) \neq \emptyset$ (compare Figure 4.23).

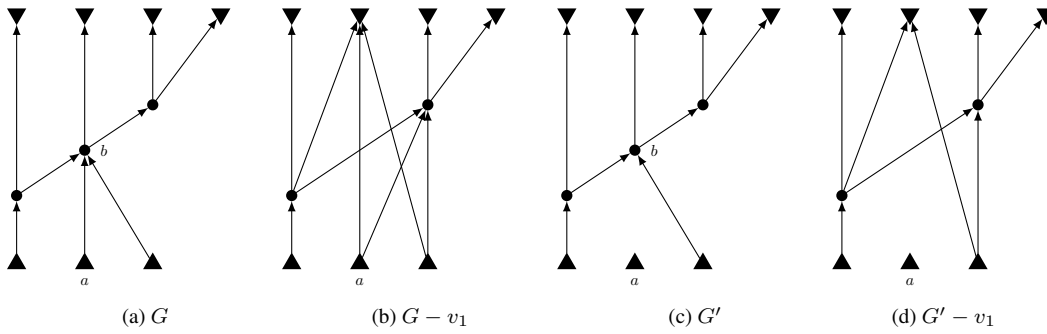


Figure 4.23: Proof of Case 1.2 in Lemma 4.19.

Repeated application of the induction assumption to all (a, w) with $w \in S$ yields

$$\begin{aligned} \sigma_{G_1}([v_2, \dots, v_{n+1}]) &\geq \sigma_{G'-v_1}([v_2, \dots, v_{n+1}]) + \sum_{w \in S} \underbrace{|X_{G_1}(a)|}_{=1} \cdot |Y_{G_1}(w)| \\ &= \sigma_{G'-v_1}([v_2, \dots, v_{n+1}]) + \sum_{w \in S} |Y_{G_1}(w)| \quad . \end{aligned} \tag{4.13}$$

From the definition of vertex elimination follows that the elimination of $v_1 = b$ preserves all paths connecting $w \in S(b)$ with the outputs of G and hence $Y(w) = Y_{G_1}(w)$ for all $w \in S(b)$. We also know that in absorption-free l-DAGs

$$Y(b) = \bigcup_{w \in S(b)} Y(w) = \bigcup_{w \in S(b)} Y_{G_1}(w) = \bigcup_{w \in S} Y_{G_1}(w) \cup (S(b) \setminus S) \quad .$$

We set $T = S(b) \setminus S$. Hence

$$|Y(b)| = \sum_{w \in S} |Y_{G_1}(w)| + |T| \quad . \quad (4.14)$$

Combining the results from Equation (4.12), Equation (4.13) and Equation (4.14) we get

$$\begin{aligned} \sigma([v_1, \dots, v_{n+1}]) &= \sigma(v_1) + \sigma_{G_1}([v_2, \dots, v_{n+1}]) \\ &\stackrel{\text{Eq(4.12)}}{\geq} \sigma_{G'}(v_1) + \sigma_{G'-v_1}([v_2, \dots, v_{n+1}]) + \underbrace{|S(b)|}_{\geq |T|} + \sum_{w \in S} |Y_{G_1}(w)| \\ &\stackrel{\text{Eq(4.13)}}{\geq} \sigma_{G'}(v_1) + \sigma_{G'-v_1}([v_2, \dots, v_{n+1}]) + |T| + \sum_{w \in S} |Y_{G_1}(w)| \\ &\stackrel{\text{Eq(4.14)}}{=} \sigma_{G'}([v_1, \dots, v_{n+1}]) + \underbrace{|X(a)|}_{=1} \cdot |Y(b)| \quad , \end{aligned}$$

finishing the proof for this case.

Case 2. $a \in Z(G)$. This case is symmetrical to **Case 1**.

□

For intermediate edges we can provide a better lower bound for the critical degree.

Lemma 4.20. *Consider absorption-free l-DAG G , $(a, b) \in \{(a, b) \in E^*(G) \mid a, b \in Z(G)\}$. For any complete vertex elimination sequence $[v_1, \dots, v_n]$ holds*

$$\sigma([v_1, \dots, v_n]) \geq \sigma_{G'}([v_1, \dots, v_n]) + |X(a)| \cdot |Y(b)| + \min\{|X(a)|, |Y(b)|\} \quad ,$$

where G' is a subgraph of G with $(a, b) \notin E(G')$.

Proof. We set $G_i = G - [v_1, \dots, v_i]$ and $G'_i = G' - [v_1, \dots, v_i]$. Proof by induction over the length $n = |Z(G)|$ of the elimination sequence. To insure the property that $a, b \in Z(G)$, G has to contain at least two intermediate vertices. Therefore we start the induction by $n = 2$.

$n = 2$: As a and b are the only vertices in the l-DAG G and $(a, b) \in E(G)$

$$P(a) = X(a) \quad \text{and} \quad S(b) = Y(b) \quad .$$

Case 1. $v_1 = b$. From the definition of vertex elimination follows that elimination of vertex $v_1 (= b)$ from G can be simulated by front elimination of all its incoming edges. Taking into account that $(a, b) \notin E(G')$ we get

$$\begin{aligned} \sigma(v_1) &= \sum_{v \in P(b)} \sigma((v, b)^+) = \sum_{v \in P(b) \setminus \{a\}} \sigma((v, b)^+) + \sigma((a, b)^+) \\ &= \sum_{v \in P(b) \setminus \{a\}} |S(b)| + |S(b)| \geq \sum_{v \in P_{G'}(b)} |S(b)| + \underbrace{|S(b)|}_{=|Y(b)|} \\ &= \sigma_{G'}(v_1) + |Y(b)| \quad . \end{aligned} \quad (4.15)$$

From $(a, b) \notin E(G')$ follows that $(a, w) \notin E(G' - b)$ while $(a, w) \in E(G - b)$ for all $w \in S(b)$. Hence $S_{G_1}(a) = S(a) \dot{\cup} S(b)$ while $S_{G'_1}(a) = S_{G'}(a) \subseteq S(a) \setminus \{b\}$ yielding

$$\begin{aligned}
\sigma_{G_1}(v_2) &= \sigma_{G_1}(a) = \sum_{v \in S_{G_1}(a)} \sigma_{G_1}((a, v)^-) = \sum_{v \in S(a) \dot{\cup} S(b)} \sigma_{G_1}((a, v)^-) \\
&= \sum_{v \in S(a)} |P_{G_1}(a)| + \sum_{v \in S(b)} |P_{G_1}(a)| = \sum_{v \in S(a)} \underbrace{|P_{G_1}(a)|}_{=|P(a)|} + \sum_{v \in S(b)} \underbrace{|P_{G_1}(a)|}_{=|P(a)|} \\
&\geq \sum_{v \in S_{G'_1}(a)} \sigma_{G'_1}((a, v)^-) + \sum_{v \in S(b)} |P(a)| \geq \sigma_{G'_1}(a) + \sum_{v \in S(b)} |P(a)| \\
&= \sigma_{G'_1}(a) + \underbrace{|S(b)|}_{=|Y(b)|} \cdot \underbrace{|P(a)|}_{=|X(a)|} = \sigma_{G'_1}(v_2) + |X(a)| \cdot |Y(b)| \quad . \tag{4.16}
\end{aligned}$$

Combining the results from Equation (4.15) and Equation (4.16), we get

$$\begin{aligned}
\sigma([v_1, v_2]) &= \sigma(v_1) + \sigma_{G_1}(v_2) \\
&\stackrel{\text{Eq(4.15)}}{\geq} \sigma_{G'}(v_1) + |Y(b)| + \sigma_{G'_1}(v_2) + |X(a)| \cdot |Y(b)| \\
&\stackrel{\text{Eq(4.16)}}{\geq} \sigma_{G'}([v_1, v_2]) + |Y(b)| + |X(a)| \cdot |Y(b)| \\
&\geq \sigma_{G'}([v_1, v_2]) + \min\{|X(a)|, |Y(b)|\} + |X(a)| \cdot |Y(b)| \quad ,
\end{aligned}$$

finishing the proof for this case.

Case 2. $v_1 = a$. This case is symmetrical to **Case 1**.

$n \mapsto n + 1$

Case 1. $v_1 \neq a$ and $v_1 \neq b$. Hence $X_{G_1}(a) = X(a)$ and $Y_{G_1}(b) = Y(b)$. The claim follows immediately from applying the induction assumption to G_1 and the fact that $\sigma(v_1) \geq \sigma_{G'}(v_1)$ according to Lemma 3.6.

Case 2. $v_1 = b$. From the definition of vertex elimination follows that $v_1 (= b)$ can be eliminated from G by subsequent front elimination of all its incoming edges. Taking into account that $(a, b) \notin E(G')$, we get

$$\begin{aligned}
\sigma(v_1) &= \sum_{v \in P(b)} \sigma((v, b)^+) = \sum_{v \in P(b) \setminus \{a\}} \sigma((v, b)^+) + \sigma((a, b)^+) \\
&= \sum_{v \in P(b) \setminus \{a\}} \sigma((v, b)^+) + |S(b)| \geq \sum_{v \in P_{G'}(b)} \sigma_{G'}((v, b)^+) + |S(b)| \\
&= \sigma_{G'}(v_1) + |S(b)| \quad . \tag{4.17}
\end{aligned}$$

We set $S = S(b) \setminus Y(b)$ and $T = S(b) \cap Y(b)$. $(a, b) \notin E(G')$ and therefore for all $w \in S(b)$: $(a, w) \notin E(G'_1)$ while $(a, w) \in E(G_1)$. From induction assumption follows that the contribution of the edge (a, w) with $w \in S$ to the overall costs of the vertex elimination sequence $[v_2, \dots, v_{n+1}]$ on G_1 is at least

$$\sigma_{G_1}([v_2, \dots, v_n]) \geq \sigma_{G'_1}([v_2, \dots, v_n]) + \min\{|Y_{G_1}(w)|, |X_{G_1}(a)|\} + |Y_{G_1}(w)| \cdot |X_{G_1}(a)| \quad . \tag{4.18}$$

For all $w \in T$ the edge (a, w) is a edge on G_1 . According to Lemma 4.19 the contribution of this edge to the overall costs of the vertex elimination sequence $[v_2, \dots, v_{n+1}]$ on G_1 is at least

$$\sigma_{G_1}([v_2, \dots, v_n]) \geq \sigma_{G'_1}([v_2, \dots, v_n]) + |X_{G_1}(a)| \quad . \tag{4.19}$$

Hence repeated application of the Equation (4.18) and Equation (4.19) to all $(a, w) \in E^*(G_1)$ yields

$$\begin{aligned}
\sigma_{G_1}([v_2, \dots, v_{n+1}]) &\geq \sigma_{G'_1}([v_2, \dots, v_{n+1}]) \\
&\quad + \sum_{v \in S} \left(\min\{|Y_{G_1}(v)|, |X_{G_1}(a)|\} + |Y_{G_1}(v)| \cdot |X_{G_1}(a)| \right) + \sum_{v \in T} |X_{G_1}(a)| \\
&= \sigma_{G'_1}([v_2, \dots, v_{n+1}]) \\
&\quad + \sum_{v \in S} \min\{|Y_{G_1}(v)|, |X_{G_1}(a)|\} + |X_{G_1}(a)| \sum_{v \in S} |Y_{G_1}(v)| + |T| \cdot |X_{G_1}(a)| \quad .
\end{aligned} \tag{4.20}$$

Obviously as G is absorption-free l-DAG

$$Y(b) = \left(Y(b) \setminus T \right) \dot{\cup} T = \bigcup_{v \in S} Y(v) \dot{\cup} T = \bigcup_{v \in S} Y_{G_1}(v) \dot{\cup} T \tag{4.21}$$

and hence applying Equation (4.21) to Equation (4.20) we get

$$\begin{aligned}
\sigma_{G_1}([v_2, \dots, v_{n+1}]) &\geq \sigma_{G'_1}([v_2, \dots, v_{n+1}]) \\
&\quad + \sum_{v \in S} \min\{|Y_{G_1}(v)|, |X_{G_1}(a)|\} + |X_{G_1}(a)| (|Y(b)| - |T|) + |T| \cdot |X_{G_1}(a)| \\
&= \sigma_{G'_1}([v_2, \dots, v_{n+1}]) + \sum_{v \in S} \min\{|Y_{G_1}(v)|, |X_{G_1}(a)|\} + |X_{G_1}(a)| \cdot |Y(b)| \quad .
\end{aligned} \tag{4.22}$$

Combining all results of Equation (4.17), Equation (4.22) and Equation (4.21) and taking into account that $X(a) = X_{G_1}(a)$, we get

$$\begin{aligned}
\sigma([v_1, \dots, v_{n+1}]) &\geq \sigma_{G'}([v_1, \dots, v_{n+1}]) + |S(b)| + |X(a)| \cdot |Y(b)| + \sum_{v \in S} \min\{|Y_{G_1}(v)|, |X(a)|\} \\
&\geq \sigma_{G'}([v_1, \dots, v_{n+1}]) + \underbrace{|S(b)|}_{\geq |T|} + |X(a)| \cdot |Y(b)| + \min \left\{ \sum_{v \in S} |Y_{G_1}(v)|, |X(a)| \right\} \\
&\geq \sigma_{G'}([v_1, \dots, v_{n+1}]) + |T| + |X(a)| \cdot |Y(b)| + \min\{|Y(b)| - |T|, |X(a)|\} \\
&\geq \sigma_{G'}([v_1, \dots, v_{n+1}]) + |X(a)| \cdot |Y(b)| + \min\{|Y(b)|, |X(a)|\} \quad ,
\end{aligned}$$

finishing the proof for this case.

Case 3. $v_1 = a$. This case is symmetrical to **Case 2**.

□

We can now combine the lower bounds for intermediate and non-intermediate edges into one formula.

Theorem 4.5. *The following holds for any absorption-free l-DAG G and $(a, b) \in E^*(G)$.*

$$\Delta_G((a, b)) \geq \min\{|X(a) \setminus \{a\}|, |Y(b) \setminus \{b\}|\} + |X(a)| \cdot |Y(b)|$$

Proof. Follows directly from Lemma 4.19 and Lemma 4.20 using that

$$\min\{|X(a) \setminus \{a\}|, |Y(b) \setminus \{b\}|\} = \begin{cases} 0, & a \in X(G) \text{ or } b \in Y(G) \\ \min\{|X(a)|, |Y(b)|\}, & a \notin X(G) \text{ and } b \notin Y(G) \end{cases} \quad .$$

□

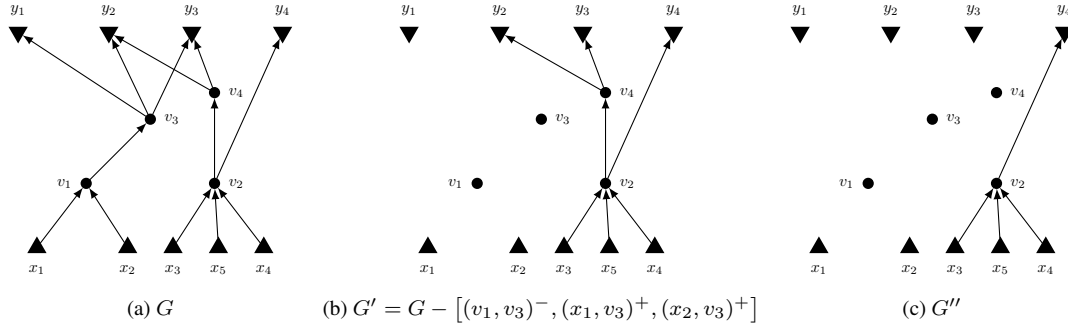


Figure 4.24: Optimality preserving eliminations using critical degree of an edge (illustrating Example 4.10).

The result from Theorem 4.5 is one of the most important results of this thesis. It not only allows to perform optimality preserving eliminations. It also builds the base for the algorithms computing lower bounds on absorption-free developed in this thesis.

We now have a lower bound for the critical degree of an edge. If we are able to transform a graph by edge (vertex) eliminations to a subgraph that does not contain this edge, using the same number of fmas as provided by the lower bound, then these eliminations preserve optimality according to Lemma 4.18. In the following example we will demonstrate the application of the Theorem 4.5 to compute the lower bound for the critical degree of an edge and show how to use the result for optimality preserving eliminations.

Example 4.10. Consider l-DAG G as on Figure 4.24(a). According to Theorem 4.5 the critical degree of the edge (v_1, v_3) is at least 8. Now let us see whether we can find an edge elimination sequence that transforms G into G' such that G' is a subset of G and $(v_1, v_3) \notin E(G')$ at costs of 8. Obviously $[(v_1, v_3)^-, (x_1, v_3)^+, (x_2, v_3)^+]$ is a such elimination sequence. The resulting l-DAG $G' = G - [(v_1, v_3)^-, (x_1, v_3)^+, (x_2, v_3)^+]$ is presented on Figure 4.24(b). Hence, according to Lemma 4.18 this edge elimination sequence preserves optimality. Similar, the critical degree of the edge (v_2, v_4) in G' is also at least 8 and therefore the edge elimination sequence $[(v_2, v_4)^+, (v_4, y_2)^-, (v_4, y_3)^-]$ preserves optimality. The resulting l-DAG $G'' = G' - [(v_2, v_4)^+, (v_4, y_2)^-, (v_4, y_3)^-]$ is presented on Figure 4.24(c). Again using the Theorem 4.5 we know that the critical degree of the edge (v_2, y_4) on l-DAG G'' is 3. Obviously the back elimination of the edge (v_2, y_4) preserves optimality according to Lemma 4.18. So we solved the optimal edge elimination problem for the l-DAG G using 19 fmas.

As you can see from the previous example, we can always find an optimality preserving elimination sequence for an intermediate edge (a, b) when all predecessors of a and all successors b are inputs or outputs respectively. To remove such an edge optimally we first have to compare the number of predecessors of a with the number or successors of b . If $|P(a)| \leq |S(b)|$, we can remove this edge by back eliminating it followed by front elimination of all edges connecting the predecessors of a with b (as we did with the edge (v_1, v_3) in Example 4.10). If $|P(a)| > |S(b)|$ we first front eliminate the edge (a, b) followed by back elimination of all edges connecting a with successors of b (e.g. edge (v_2, v_4) in Example 4.10). The next Lemma proves this strategy.

Lemma 4.21. Consider absorption-free l-DAG G , (a, b) an intermediate edge of G such that $P(a) \subseteq X(G)$ and $S(b) \subseteq Y(G)$.

1. If $|P(a)| \leq |S(b)|$ then the edge elimination sequence $[(a, b)^-, (v_1, b)^+, \dots, (v_{|P(a)|}, b)^+]$, where $v_i \in P(a)$ for all $1 \leq i \leq |P(a)|$ and $v_i \neq v_j$ for $i \neq j$ preserves optimality for edge elimination.

2. If $|P(a)| \geq |S(b)|$ then the edge elimination sequence $[(a, b)^+, (a, v_1)^-, \dots, (a, v_{|S(b)|})^-]$, where $v_i \in S(b)$ for all $1 \leq i \leq |S(b)|$ and $v_i \neq v_j$ for $i \neq j$ preserves optimality for edge elimination.

Proof. Ad 1: Obviously

$$G - [(a, b)^-, (v_1, b)^+, \dots, (v_{|P(a)|}, b)^+] \subseteq G \quad .$$

From the fact that $P(a) \subseteq X(G)$ and $S(b) \subseteq Y(G)$ follows that $P(a) = X(a)$ and $S(b) = Y(b)$. Hence the costs of the edge elimination sequence are

$$\begin{aligned} \sigma\left([(a, b)^-, (v_1, b)^+, \dots, (v_{|P(a)|}, b)^+]\right) &= |P(a)| + \sum_{v \in P(a)} |S(b)| = |P(a)| + |P(a)| \cdot |S(b)| \\ &= |X(a)| + |X(a)| \cdot |Y(b)| \\ &= \min\{|X(a)|, |Y(b)|\} + |X(a)| \cdot |Y(b)| \quad . \end{aligned}$$

The claims follow immediately from Lemma 4.18 and Theorem 4.5.

Ad 2: Symmetrical to 1. □

Algorithm 6 performs optimality preserving eliminations according to Lemma 4.21 for all edges that satisfy the requirements of the lemma and computes the costs for these eliminations. This algorithm computes incremental costs such that it can be used in other algorithms later (e.g. Algorithm 8).

Algorithm 6 Optimality preserving eliminations according to Lemma 4.21

Require: absorption-free l-DAG G , costs

Ensure: G , costs

```

1:  $U = \{(i, j) \in E^*(G) \mid P(i) \subseteq X(G) \text{ and } S(j) \subseteq Y(G)\}$ 
2: for all  $(a, b) \in U$  do
3:   if  $|P(a)| \leq |S(b)|$  then
4:      $G' \leftarrow G - (a, b)^-$ 
5:      $\text{costs} \leftarrow \text{costs} + |P(a)|$ 
6:     for all  $v \in P_G(a)$  do
7:        $G' \leftarrow G' - (v, b)^+$ 
8:        $\text{costs} \leftarrow \text{costs} + |S(b)|$ 
9:     end for
10:  else
11:     $G' \leftarrow G - (a, b)^+$ 
12:     $\text{costs} \leftarrow \text{costs} + |S(b)|$ 
13:    for all  $v \in S_G(a)$  do
14:       $G' \leftarrow G' - (a, v)^-$ 
15:       $\text{costs} \leftarrow \text{costs} + |P(a)|$ 
16:    end for
17:  end if
18:   $G \leftarrow G'$ 
19: end for
```

We do not examine the complexity of Algorithm 6 as this depends on the complexity of finding the set U (set of edges that can be eliminated preserving optimality) and we have not presented any specific algorithm to determine this set U . The complexity of performing the optimality preserving eliminations is not of interest as eliminations have to be performed in any case if we want to transform the l-DAG to bipartite graph.

We can also develop a similar strategy for an edge (a, b) . If $a \in Z(G)$ and all predecessors of a are inputs, then back elimination of (a, b) preserves optimality (compare (v_2, y_4) in Example 4.9). Similarly, if $b \in Z(G)$ and all successors of b are outputs the front elimination of the edge (a, b) preserves optimality. Let us prove this idea too.

Lemma 4.22. *Consider absorption-free l-DAG G , (a, b) a non-intermediate edge.*

1. *If $a \in X(G)$ and $S(b) \subseteq Y(G)$ then forward elimination of edge (a, b) preserves optimality for edge elimination*
2. *If $b \in Y(G)$ and $P(a) \subseteq X(G)$ then back elimination of edge (a, b) preserves optimality for edge elimination*

Proof. Ad 1: Obviously $G - (a, b)^+ \subseteq G$. The costs of front elimination of (a, b) are

$$\sigma((a, b)^+) = |S(b)| = |Y(b)| \quad .$$

Hence the claim follows immediately from Lemma 4.18 and Theorem 4.5.

Ad 1: Symmetrical to 1. □

Algorithm 7 performs optimality preserving eliminations according to Lemma 4.22 for all edges that satisfy the requirements of this lemma. Similar to Algorithm 6 this algorithm computes incremental costs such that it can be used in other algorithms later (e.g. Algorithm 8).

Algorithm 7 Optimality preserving eliminations according to Lemma 4.22

Require: absorption-free l-DAG G , costs

// Computes incremental costs

Ensure: G , costs

- 1: $U \leftarrow \{(i, j) \in E^*(G) \mid (i \in X(G) \wedge S(j) \subseteq Y(G)) \text{ or } (j \in Y(G) \wedge P(i) \subseteq X(G))\}$
 - 2: **for all** $(a, b) \in U$ **do**
 - 3: **if** $a \in Z(G)$ **then**
 - 4: $G \leftarrow G - (a, b)^-$
 - 5: costs $\leftarrow$ costs + $|P(a)|$
 - 6: **else**
 - 7: $G \leftarrow G - (a, b)^+$
 - 8: costs $\leftarrow$ costs + $|S(b)|$
 - 9: **end if**
 - 10: **end for**
-

We do not examine the complexity of Algorithm 7 for the same reasons given for Algorithm 6.

The combination of Algorithm 6 and Algorithm 7 yields an algorithm that finds a complete optimal edge elimination sequence for absorption-free l-DAGs with maximal path length 3. This algorithm is presented in Algorithm 8. Please refer to Example 4.9 for an example of application of Algorithm 8. Let

Algorithm 8 Solve EE for absorption-free l-DAG G with maximal path length 3.

Require: absorption-free l-DAG G with maximal path length 3

Ensure: costs

- 1: costs $\leftarrow$ 0
 - 2: ALGORITHM 6(G , costs)
 - 3: ALGORITHM 7(G , costs)
-

us now prove the correctness of the Algorithm 8.

Lemma 4.23. *Consider absorption-free l-DAG G , with maximal path length 3. Algorithm 8 finds a complete optimal edge elimination sequence.*

Proof. According to Lemma 4.21 Algorithm 6 performs optimality preserving edge eliminations. Obviously after it finishes the l-DAG contains only paths of length 2. Hence the prerequisites of the Lemma 4.22 are fulfilled and therefore the routine Algorithm 7 also performs optimality preserving edge elimination sequences. Hence Algorithm 8 computes a complete optimal edge elimination sequence. $\square$

Finding the sets U for Algorithm 6 and Algorithm 7 can be done in linear time. Moreover for each edge in these sets we know how to eliminate them while preserving the optimality. Hence Algorithm 8 solves the edge elimination problem for an l-DAG with maximal path length 3 in linear time. Using similar ideas, we could also develop an algorithm that finds a complete optimal vertex elimination sequence. This algorithm would look for an intermediate edge (a, b) such that $\min\{|X(a)|, |Y(b)|\}$ is smallest among all intermediate edges. Once you have found such an edge, you have to eliminate vertex a if $|X(a)| \leq |Y(b)|$ or b if $|X(a)| > |Y(b)|$. Then you repeat this procedure, until the l-DAG contains only non-intermediate edges. You can then eliminate the remaining vertices in arbitrary order.

We will omit writing down this algorithm and proving its correctness as the Algorithm 8 is faster and thus should be preferred in practice. Still although the optimal vertex elimination problem has a much smaller search space compared to the edge elimination problem, sometimes switching to edge elimination allows the problem to be solved more efficiently. We will come back to this in later chapters.

4.3.3 Lower bounds

In the previous subsection we made significant progress in solving the optimal edge [vertex] elimination problem for absorption-free l-DAGs. Still we are not able to solve this problem completely. There are still absorption-free l-DAGs that cannot be solved by the technique presented so far (compare l-DAG on Figure 4.25(a)). As we can see, we cannot apply any of the optimality techniques from the previous subsection to this l-DAG. For such cases good lower bounds can help us solve the optimal edge [vertex] elimination problem on such l-DAGs efficiently. In this subsection we will discuss existing lower bound based on Markowitz degree (compare Definition 2.14) first presented [NH08] and develop new (better) lower bounds based on critical degree and optimality preserving eliminations from the previous subsection. For the lower bounds based on critical degree we exploit the result from Theorem 4.5.

This subsection is structured as follows: first we discuss the lower bound based on Markowitz degree and present the corresponding algorithm to compute the lower bound (Algorithm 9). Then we discuss the basic algorithm for computing lower bounds based on critical degree of an edge (Algorithm 10). Afterwards we compare both approaches and show that neither of them is superior. Although in most cases Algorithm 10 provides better results. We then extend the basic algorithm by using the optimality preserving results from Section 4.3.2. The resulting algorithm is shown in Algorithm 11. We prove that lower bounds provided by Algorithm 11 are superior to the lower bounds based on Markowitz degree. At the end of this subsection we apply Algorithm 11 to several absorption-free graphs. These results will be reused in Chapter 6 and Chapter 7.

As the number of successors and predecessors is not decreasing in the absorption-free l-DAG, the sum of Markowitz degrees of all intermediate vertices is obviously a lower bound for the number of multiplications for any complete vertex or edge elimination sequence as shown in the next lemma.

Lemma 4.24. *Consider absorption-free l-DAG G .*

$$VE(G) = EE(G) \geq \sum_{v \in Z(G)} \text{mark}_G(v) \quad .$$

Proof. Follows immediately from Lemma 4.3 and Definition 2.14. $\square$

The next algorithm uses the result from Lemma 4.24 to compute the lower bounds. (Note: the algorithm computes incremental lower bound for usage in later algorithms).

Algorithm 9 Lower bound for absorption-free l-DAGs based on Markowitz degree

Require: absorption-free l-DAG G , lb

Ensure: $lb \leq EE(G)$

for all $v \in Z(G)$ **do**

$lb \leftarrow lb + |P(v)| \cdot |S(v)|$

end for

Often there is a big gap between the bounds computed by the Algorithm 9 and the number of fmas actually performed during an optimal edge (vertex) elimination sequence, as shown in the next example.

Example 4.11. Consider l-DAG G as on Figure 4.24(a). The sum of Markowitz degrees of all intermediate vertices in this l-DAG is

$$\sum_{i=1}^4 \text{mark}_G(v_i) = 2 + 3 + 6 + 2 = 13 \quad ,$$

while the optimal solution requires 19 fmas as shown in Example 4.10.

The reason for this is, that this approach does not take into account the fact that the Markowitz degree of vertices may (and typically does) increase during the elimination sequence. For example the Markowitz degree of the v_3 on l-DAG G from Figure 4.24(a) after the front elimination of (v_1, v_2) is 6 and not 3 as before the edge elimination.

One idea to address this problem was presented in [CHMU12]. A lower bound for the number of fmas of a given vertex elimination sequence was required to speed up the branch and bound algorithm for solving the VE problem. The vertex elimination problem was reformulated as integer programming. The authors suggested replacing the Markowitz degree of the last vertex in the elimination sequence with the number of independent variables connected to this vertex times the number of dependent variables connected to it. While for given vertex elimination sequence this idea improves the lower bound for the number of fma's, it may fail to provide a valid lower bound for an l-DAG in general as shown in the next example.

Example 4.12. Consider l-DAG G as on Figure 4.21(b). As we already know from Example 4.8 $VE(G) = EE(G) = 10$. If we assume that v_1 is eliminated last, the lower bound we get is not valid as

$$\text{mark}_G(v_2) + |X_G(v_1)| \cdot |Y_G(v_1)| = 4 + 2 \cdot 4 = 12 > 10 \quad .$$

In the following we will use the under estimator of the critical degree of an edge (Theorem 4.5) to compute a lower bound for the optimal edge elimination problem. The basic idea behind this approach exploits the fact that $EE(G) = \Delta_G(i, j) + EE(G[E(G) \setminus \{(i, j)\}])$. So to compute the lower bound we can choose an arbitrary edge (i, j) , compute the lower bound for the critical degree of this edge with the results from Theorem 4.5 and then repeat this procedure with the resulting l-DAG $G[E(G) \setminus \{(i, j)\}]$. Let us discuss this approach on an example.

Example 4.13. Consider l-DAG G as on Figure 4.24(a). According to Theorem 4.5 $\Delta_G(v_1, v_3) \geq 2 + 2 \cdot 3 = 8$. $G' = (G[E(G) \setminus \{(v_1, v_3)\}])$ is presented on Figure 4.24(b). Similarly the critical degree of (v_2, v_4) on G' is at least 8 and the resulting l-DAG $G'' = (G'[E(G') \setminus \{(v_2, v_4)\}])$ is shown on Figure 4.24(c). Last we estimate the critical degree for (v_2, v_4) on G'' which is according to Theorem 4.5 at least 3. Hence we get a lower bound of $8 + 8 + 3 = 19$.

Obviously we should never estimate a critical degree for an edge if it lays on a path longer than 2, as this approach yields worse lower bounds. Instead of that we should rather estimate the critical degree of an intermediate edges that lays on the same path. As demonstrated in the following example.

Example 4.14. Consider l-DAG G as on Figure 4.24(a). The edges (x_1, v_1) and (x_2, v_1) induce the same part of G as the edge (v_1, v_3) as $G[E(G) \setminus \{(v_1, v_3)\}] = G[E(G) \setminus \{(x_1, v_1), (x_2, v_1)\}]$. If we use Theorem 4.5 to estimate the critical degree of the edges (x_1, v_1) and (x_2, v_1) , we get a lower bound of 6. Which is smaller compared to the lower bound we get if we estimate the critical degree of (v_1, v_3) which gives us a lower bound of 8 for the same part of G (as already discussed in Example 4.13).

This leads us to the basic algorithm for lower bounds based on critical-degree presented in Algorithm 10. (Note the algorithm computes incremental lower bounds for usage in later algorithms).

Algorithm 10 Lower bound based on critical-degree (basic algorithm)

Require: absorption-free l-DAG G , $1b$

Ensure: $1b \leq EE(G)$

$U \leftarrow E^*(G) \cap (Z(G) \times Z(G))$

while $U \neq \emptyset$ **do**

// Intermediate edges

 choose $(i, j) \in U$

$1b \leftarrow 1b + \min\{|X(i)|, |Y(j)|\} + |X(i)| \cdot |Y(j)|$

$G \leftarrow G[E(G) \setminus \{(i, j)\}]$

$U \leftarrow E^*(G) \cap (Z(G) \times Z(G))$

end while

for all $(i, j) \in E^*(G)$ **do**

// Non-intermediate edges

$1b \leftarrow 1b + |X(i)| \cdot |Y(j)|$

$G \leftarrow G[E(G) \setminus \{(i, j)\}]$

end for

Algorithm 10 works as follows. As long as the set of intermediate edges in l-DAG G is not empty, the algorithm chooses one of the intermediate edges, estimates its critical degree according to Theorem 4.5 and increases the lower bound $1b$ by this value. Then the l-DAG G is set to the subgraph of G without the edge (i, j) and we apply the same procedure to the new l-DAG G . Once the set of intermediate edges of G is empty, the Algorithm 10 applies a similar approach for non-intermediate edges of G . The correctness of the Algorithm 10 follows immediately from the Lemma 4.19 and Lemma 4.20.

The complexity of Algorithm 10 depends on the complexity of building a subgraph of G without the edge (i, j) and the costs of determining the size of the sets $X(i)$ and $Y(j)$. To build the subgraph we need to remove the edge (i, j) and then recursively remove all intermediate vertices that have no incoming or no outgoing edges. The removal of each vertex also reduces the number of edges in the graph by at least one, such that we do not have to estimate the critical degree for this edge anymore. Therefore we assume that constructing a subgraph costs one operation. The costs of determining the sizes of $X(i)$ and $Y(j)$ strongly depend on the given implementation. Discussing all various approaches goes beyond the scope of this thesis. At this point I would only like to note that on absorption-free graphs the sizes of the sets of $X(i)$ and $Y(j)$ can be determined efficiently as

$$|X(v)| = \sum_{i \in P(v)} |X(i)| \quad \text{and} \quad |Y(v)| = \sum_{i \in S(v)} |Y(i)| \quad .$$

In summary, the Algorithm 10 has a worst case complexity of $O(|E^*(G)| \cdot F)$, where F is the complexity of computing the size of $X(i)$ and $Y(j)$.

It is easy to see that if the l-DAG G has maximal path length 3, then the Algorithm 10 computes a perfect lower bound. This means that the computed lower bound is equal to the number of fmas that are performed during an optimal edge (vertex) elimination sequence as discussed in the next example.

Example 4.15. *If we apply this algorithm to the l-DAG on Figure 4.24(a) the algorithm computes the lower bound for the critical-degree of the intermediate edges (v_1, v_3) and (v_2, v_4) . The computed lower bound is 8 for both edges. For the resulting l-DAG $G[E(G) \setminus \{(v_1, v_3), (v_2, v_4)\}]$ (compare Figure 4.24(c)) the algorithm would provide a lower bound of 3. All together we get a lower bound of 19 which is equal to the number of fmas performed by the optimal elimination sequence as demonstrated in Example 4.10.*

Let us now apply the Algorithm 10 to a slightly bigger l-DAG where the algorithm does not compute a perfect lower-bound.

Example 4.16. *Consider G as on Figure 4.25(a). (We assume that Algorithm 10 picks the edge with smallest source first. If there is more than one edge with the smallest source then it picks the one with the smallest sinks). According to this assumption the algorithm first estimates the contribution of the edge (v_1, v_2) . This edge is connected to 4 inputs and 7 outputs. Hence*

$$\Delta_G(v_1, v_2) \geq 4 + 7 \cdot 4 = 32 \quad .$$

The resulting l-DAG $G' = G[E(G) \setminus \{(v_1, v_2)\}]$ is shown on Figure 4.25(b) Next the algorithm picks the

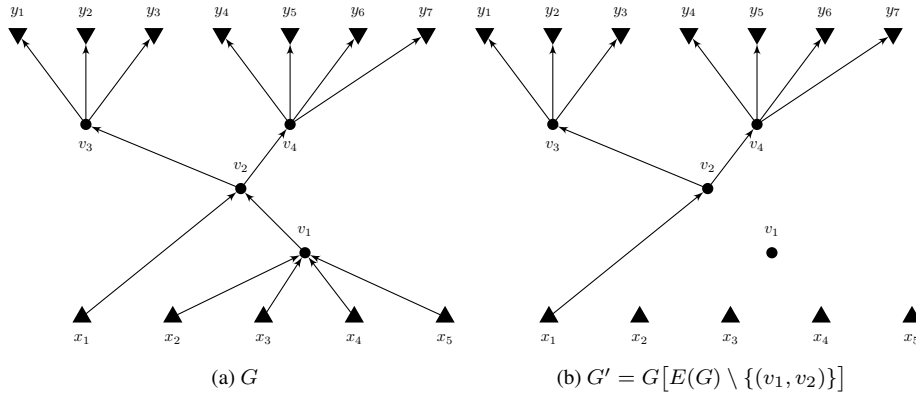


Figure 4.25: Application of Algorithm 10 (illustrating Example 4.16).

edge (v_2, v_3) . This edge is connected to 1 input and 3 outputs in G' . Hence, according to Theorem 4.5.

$$\Delta_{G'}(v_2, v_3) \geq 1 + 1 \cdot 3 = 4 \quad .$$

Then the algorithm estimates the contribution of the edge (v_2, v_4) on the l-DAG $G'' = G[E(G) \setminus \{(v_1, v_2), (v_2, v_3)\}]$. The contribution of this edge is the same as on G' as (v_2, v_3) and (v_2, v_4) do not have any common paths. (v_2, v_4) is connected to 1 input and 4 outputs and therefore

$$\Delta_{G''}(v_2, v_4) \geq 1 + 1 \cdot 4 = 5 \quad .$$

Hence, the lower bound computed by the Algorithm 10 for l-DAG G is $32 + 4 + 5 = 41$. The costs of the elimination sequence $[v_1, v_3, v_4, v_2]$ are 46 and therefore the lower bound is at most 5 fmas away from

the optimal solution. Using the lower bound based on Markowitz degree we get a lower bound of

$$\sum_{i=1}^4 \text{mark}_G(v_i) = 4 + 4 + 3 + 5 = 15 \quad ,$$

showing the superiority of the new approach for this example.

The choice of the edges does have an influence on the produced bounds as the following example demonstrates.

Example 4.17. Consider an l-DAG G as on Figure 4.25(a). In the Example 4.16 we saw a lower bound of 41 fmas. Now we show that the Algorithm 10 can provide a better bound if we choose different edges to estimate the critical degree. According to Theorem 4.5

$$\Delta_G(v_2, v_3) \geq 3 + 3 \cdot 5 = 18 \quad \text{and} \quad \Delta_{G'}(v_2, v_4) \geq 4 + 4 \cdot 5 = 24 \quad ,$$

where $G' = (G[E(G) \setminus \{(v_2, v_3)\}])$. Hence we get a lower bound of $18 + 24 = 42$.

Although Algorithm 10 seems to provide very tight lower bounds, it is not always superior to the sum of Markowitz degree as shown in the next example.

Example 4.18. Consider l-DAG G as on Figure 4.26(a). Using the Algorithm 9 to compute the lower bound yields a lower bound of

$$\text{lb} = \sum_{i=1}^3 \text{mark}_G(v_i) = 2 + 1 + 1 = 4 \quad .$$

Let us now look what lower bound is computed by Algorithm 10. If the algorithm chooses the edge

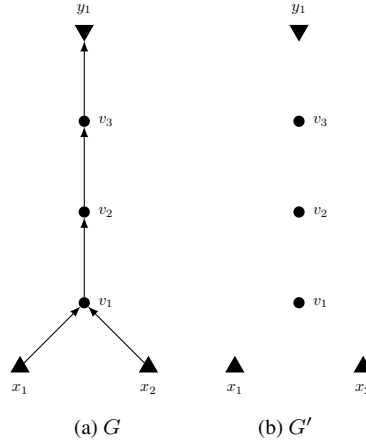


Figure 4.26: Algorithm 9 is better than Algorithm 10 (illustration of Example 4.18).

(v_1, v_2) , we get the lower bound of

$$\text{lb} = \min\{|X(v_1)|, |Y(v_2)|\} + |X(v_1)| \cdot |Y(v_2)| = \min\{2, 1\} + 1 \cdot 2 = 3 \quad .$$

The resulting l-DAG $G' = (G[E(G) \setminus (v_1, v_2)])$ does not contain any eliminatable edges (compare Figure 4.26(b)). Therefore 3 is the final lower bound. Hence, the result produced by the lower bounds based on Markowitz degree is better than that of Algorithm 10.

The reason why the Algorithm 10 produces a bad lower bound is that after removing the edge (v_1, v_2) , v_1 does not have any outgoing edges and therefore we have to remove this vertex from the resulting l-DAG recursively. Similarly the vertex v_2 does not have any incoming edges and therefore we also remove this vertex from the l-DAG recursively. So the Algorithm 10 simply ignores the existence of the vertex v_1 . Hence, to improve the lower bounds we should try not to remove an edge that is the only outgoing edge of its source or the only incoming edge of its sink. Instead of that we should try to exploit the optimality preserving results from Lemma 4.16 and Lemma 4.17. So we extend our basic lower bound algorithm with the optimality preserving eliminations, we developed in previous section. The resulting algorithm is presented in Algorithm 11.

Algorithm 11 Lower bound based on critical-degree and optimality preserving eliminations

Require: absorption-free l-DAG G , lb
Ensure: $lb \leq EE(G)$

```

1: changed  $\leftarrow$  true
2: while  $E^*(G) \neq \emptyset$  do
3:   if changed then                                     // Perform optimality preserving eliminations
4:      $G' \leftarrow G$ 
5:     ALGORITHM 5( $G$ ,  $lb$ )
6:     ALGORITHM 6( $G$ ,  $lb$ )
7:     ALGORITHM 7( $G$ ,  $lb$ )
8:     changed  $\leftarrow (G' \neq G)$ 
9:   else                                                   // Choose an edge and compute a lower bound for its critical degree
10:    choose  $(i, j) \in E^*(G)$ 
11:     $lb = lb + \min\{|X(i) \setminus \{i\}|, |Y(j) \setminus \{j\}|\} + |X(i)| \cdot |Y(j)|$ 
12:     $G = G[E(G) \setminus \{(i, j)\}]$ 
13:    changed  $\leftarrow$  true
14:   end if
15: end while

```

Although we only extended the Algorithm 10 it looks completely different. We do not have to compute the lower bounds for the non-intermediate edges explicitly, as this is done by optimality preserving eliminations through Algorithm 7.

If we apply Algorithm 11 to the l-DAG G from Example 4.18, we get the same lower bound as with the bound based on Markowitz degree because the elimination sequence $[v_1, v_2, v_3]$ preserves optimality.

We do not discuss the runtime complexity of the this algorithm, as it strongly depends on the way of updating the sets of possible candidates for Algorithm 5, Algorithm 6 and Algorithm 7. There have not been made any implementation of this algorithm so far. More research is needed to make an efficient implementation of this algorithm.

The choice of the edge in the last branch of the Algorithm 11 is crucial. You should try to choose (i, j) such that $|S(i)| > 1$ and $|P(j)| > 1$, as otherwise this causes a removal of a vertex which typically makes bounds worse, especially if the removal of a vertex causes recursive removal of other vertices as in Example 4.18. During this Ph.D. thesis I had no time to find the best strategy for the choice of an edge neither in Algorithm 11 nor Algorithm 10 so there is still room for improvement and further research at this point. For the best choice of the edge to remove one should consider that the lower bounds based on critical degree and the optimality preserving results from the previous section can build a perfect symbiosis. If we cannot perform any optimality preserving eliminations, we can simply remove edges until an optimality preserving elimination is possible. Using this strategy we can perform optimality preserving eliminations rather than remove vertices. Let us discuss this on the following example.

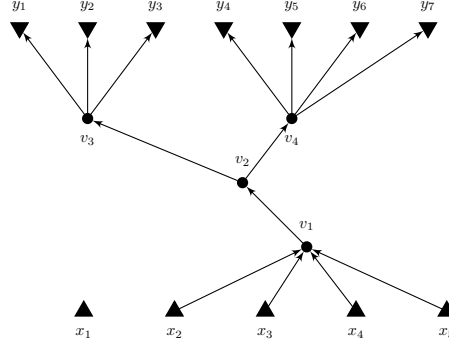


Figure 4.27: Illustration of $G' = G[E(G) \setminus \{(x_1, v_1)\}]$ in Example 4.19.

Example 4.19. Consider l-DAG G as on Figure 4.25(a). Obviously we cannot apply any of the optimality preserving results from the previous section but after the removal of (x_1, v_2) we could (compare Figure 4.27). According to Theorem 4.5

$$\Delta_G(x_1, v_2) \geq 7 \quad .$$

According to Lemma 4.16 and Lemma 4.17 the vertex elimination sequence $[v_2, v_3, v_4, v_1]$ preserves optimality on the resulting l-DAG $G' = G[E(G) \setminus \{(x_1, v_1)\}]$. The costs of this elimination sequence are

$$\sigma_{G'}([v_2, v_3, v_4, v_1]) = 2 + 3 + 4 + 7 \cdot 4 = 37 \quad .$$

Hence Algorithm 11 will provide us a lower bound of $7 + 37 = 44$ in this case.

The next lemma shows that Algorithm 11 computes better lower bounds than Algorithm 9 if it always chooses a non-intermediate edge in line 10. As already mentioned previously, choosing a non-intermediate edge is definitely not the best strategy and one should rather try and find proper intermediate edges to remove. Still this assumption simplifies the proof a lot and is good enough to show the advantages of the new approach.

Lemma 4.25. The lower bound produced by Algorithm 11 is better than that of Algorithm 9 if the edge in line 10 of Algorithm 11 is always a non-intermediate edge.

Proof. Proof by induction over the number n of iterations of the outer loop in Algorithm 11.

$n = 1$: If the iteration performs one of the optimality preserving eliminations then there is nothing to show. Let us now assume that the algorithm does not perform an optimality preserving elimination. Hence it computes a lower bound for the critical degree of a non-intermediate edge. We will show that this case is impossible because in this case we could also apply one of the optimality preserving eliminations. If we compute a lower bound for a non-intermediate edge (i, j) then $G[E(G) \setminus \{(i, j)\}]$ does not contain any edges as there is only one loop iteration. W.l.o.g. we assume that $i \in X(G)$ (the case that $j \in Y(G)$ is symmetrical). Hence $P(j) = \{i\}$ and $S(i) = \{j\}$ as otherwise $G[E(G) \setminus \{(i, j)\}]$ would still contain edges. Hence j satisfies the requirements from Lemma 4.17 leading to the desired contradiction.

$n \mapsto n+1$: Again the claim is trivially true if Algorithm 11 performs an optimality preserving elimination during the first loop iteration. Therefore we assume that in the first iteration the lower bound for the critical degree of a non-intermediate edge (i, j) is computed. W.l.o.g. we assume that $i \in X(G)$ (the case that $j \in Y(G)$ is symmetrical). We know that $|P(j)| > 1$ as otherwise the Algorithm 11 would perform an optimality preserving elimination according to Lemma 4.17. Hence $Z(G) = Z(G')$ and

$E(G) \setminus \{(i, j)\} = E(G')$, where $G' = G[E(G) \setminus \{(i, j)\}]$ and therefore $P(v) = P_{G'}(v)$ for all $v \in Z(G) \setminus \{j\}$ and $S(v) = S_{G'}(v)$ for all $v \in Z(G)$. From induction assumption follows

$$\begin{aligned}
 \text{Algorithm 11}(G) &= |Y(j)| + \text{Algorithm 11}(G') \geq |Y(j)| + \sum_{v \in Z(G')} |P_{G'}(v)| \cdot |S_{G'}(v)| \\
 &= |Y(j)| + |P_{G'}(j)| \cdot |S_{G'}(j)| + \sum_{v \in Z(G') \setminus \{j\}} |P_{G'}(v)| \cdot |S_{G'}(v)| \\
 &= |Y(j)| + (|P(j)| - 1)|S(j)| + \sum_{v \in Z(G) \setminus \{j\}} |P(v)| \cdot |S(v)| \\
 &\geq |P(j)| + (|P(j)| - 1)|S(j)| + \sum_{v \in Z(G) \setminus \{j\}} |P(v)| \cdot |S(v)| \\
 &= \sum_{v \in Z(G)} |P(v)| \cdot |S(v)| = \text{Algorithm 9}(G) \quad .
 \end{aligned}$$

□

As we can see that there exists at least one strategy for choosing (i, j) in Line 10 of Algorithm 11, such that the algorithm produces better lower bounds than Markowitz degree. You might have noticed that the choice of (i, j) creates new optimality preserving situations for Lemma 4.16 and Lemma 4.17 as already mentioned above. So if we always choose (i, j) in Line 10 of Algorithm 11 as described in Lemma 4.25, we get better lower bounds as from Markowitz degree. Still this is definitely not the optimal strategy. As already mentioned one get better lower bounds by estimating the critical degree for intermediate edges. Thus we need a strategy of choosing intermediate edges similar to the one presented in Lemma 4.25. The problem is that the choice of such an intermediate edge is more complicated. As on the one hand you aim to create new optimality preserving situations by removing this edge. On the other hand you want to assure that removal of this edge does not lead to a removal of a vertex, which is counter productive in the sense of getting good lower bounds, as already demonstrated previously. It is even not always clear that one can always find an intermediate edge with such properties. During my work on this thesis I have not found a proper strategy for this problem. Still I can give some hints from my experience, that may help to develop a good strategy. One should choose an intermediate edge if possible. If there are several possible choices, you should choose the one that induces a graph with smaller number of edges. Try to choose the edges that are closer to the inputs or outputs.

In the following we apply the presented algorithms to some more examples. These results will be used in later chapters. There we will compute lower bounds for arbitrary l-DAGs by reducing the l-DAG to absorption-free subgraphs. Therefore we will discuss the lower bounds of these subgraphs here. You may wish to jump over these examples and come back to this part once we refer to these results. By coincidence for all these examples the lower bound computed by Algorithm 11 is at the same time the solution of the EE (VE) problem.

The following example deals with absorption-free subgraphs of l-DAG from Figure 6.5(a) .

Example 4.20. *Removing the edge (x_4, v_2) from l-DAG from Figure 6.5(a) yield an absorption-free subgraph T_1 as on Figure 4.28(a). Let's apply Algorithm 11 to T_1 . The vertex v_2 satisfies the prerequisites of Lemma 4.17 hence Algorithm 11 will first perform the optimality preserving elimination of v_2 . The resulting l-DAG $G - v_2$ is shown on Figure 4.7(b). v_3 on $G - v_2$ also satisfies the prerequisites of Lemma 4.17 hence Algorithm 11 will then remove vertex v_3 from l-DAG G . The resulting l-DAG $G - [v_2, v_3]$ has only one intermediate vertex v_1 yielding the longest path is of length 2. Hence the call to Algorithm 7 will optimally eliminate this vertex. Hence in this case the lower bound provided by Algorithm 11 is also the optimal solution of the VE problem*

$$1b = \text{VE}(T_1) = \sigma_{T_1}([v_2, v_3, v_1]) = 3 + 2 + 16 = 21 \quad .$$

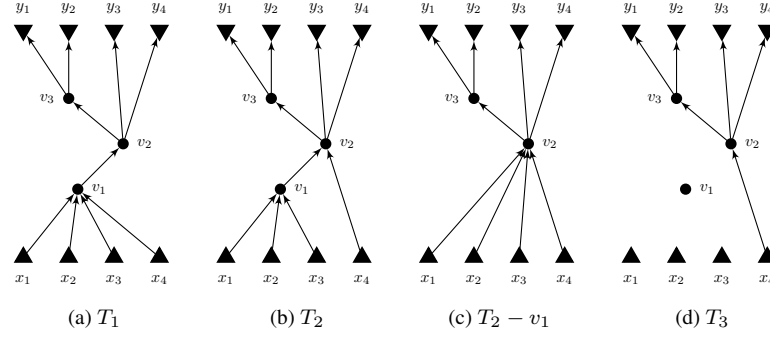


Figure 4.28: Application of Algorithm 11 to various subgraphs of l-DAG from Figure 6.5(a) (illustrating Example 4.20).

Next we apply Algorithm 11 to the absorption-free subgraph T_2 of l-DAG from Figure 6.5(a) that results from removing the edge (x_3, v_2) (Figure 4.28(b)). The vertex v_1 satisfies the prerequisites of Lemma 4.16 hence Algorithm 11 will first perform the optimality preserving elimination of this vertex. The resulting l-DAG $G - v_1$ is shown on Figure 4.28(c). The vertex v_3 satisfies the prerequisites of Lemma 4.17, thus Algorithm 11 will eliminate this vertex next. The resulting l-DAG has only one intermediate vertex, therefore the lower bound provided by Algorithm 11 is also the optimal solution of the VE problem in this case

$$1b = VE(T_2) = \sigma_{T_2}([v_1, v_3, v_2]) = 3 + 2 + 16 = 21 \quad .$$

At the end of this example we apply Algorithm 11 to T_3 as on Figure 4.28(d). T_3 is an absorption-free subgraph of l-DAG from Figure 6.5(a) after the removal of (v_1, v_2) . The vertex v_2 satisfies the prerequisites of Lemma 4.17 hence elimination of v_2 preserves optimality. The resulting l-DAG has only one intermediate meaning that hence the lower bound provided by Algorithm 11 is also the optimal solution of the VE problem in this case

$$1b = VE(T_3) = \sigma_{T_3}([v_2, v_3]) = 3 + 2 = 5 \quad .$$

The following example deals with absorption-free subgraphs of the bat graph.

Example 4.21. Removing the edges (v_3, y_1) and (v_3, y_4) from the bat graph (Figure 2.4) yields an absorption-free l-DAG T_1 as on Figure 4.29(a). If we apply Algorithm 11 to T_1 , the algorithm would estimate the critical degree for the intermediate edges (v_1, v_3) and (v_2, v_3) first and then estimate the critical degree for non-intermediate edges (v_1, y_1) and (v_2, y_4)

$$1b = \Delta_{T_1}((v_1, v_3)) + \Delta_{T_1}((v_2, v_3)) + \Delta_{T_1}((v_1, y_1)) + \Delta_{T_1}((v_2, y_4)) = 6 + 6 + 2 + 2 = 16 \quad .$$

The maximal path length is 3 hence the lower bound computed by the algorithm is also the optimal solution and thus $VE(T_1) = 16$.

Removing the edges (v_1, v_3) and (v_2, v_3) from the bat graph (Figure 2.4) yields an absorption-free l-DAG T_2 as on Figure 4.29(b). If we apply Algorithm 11 to T_2 , the algorithm would eliminate the vertices v_1 and v_2 as elimination of these vertices preserves optimality according to Lemma 4.16. Hence, the lower bound provided by the algorithm is also the optimal solution and

$$1b = VE(T_2) = \sigma_{T_2}([v_1, v_2]) = 2 + 2 = 4 \quad .$$

The last absorption-free subgraph of bat graph I would like to discuss is generated by removing the edges (v_1, y_1) and (v_2, y_4) . The resulting subgraph T_3 is shown on Figure 4.29(c). The vertices v_1 and v_2 satisfy the prerequisites from Lemma 4.16 and hence Algorithm 11 would eliminate these vertices first. The resulting l-DAG has only one intermediate vertex and hence Algorithm 11 computes the lower bound as

$$1b = VE(T_3) = \sigma_{T_3}([v_1, v_2, v_3]) = 2 + 2 + 16 = 20 \quad .$$

Again the lower bound provided by Algorithm 11 is also the solution to the VE problem on T_3 .

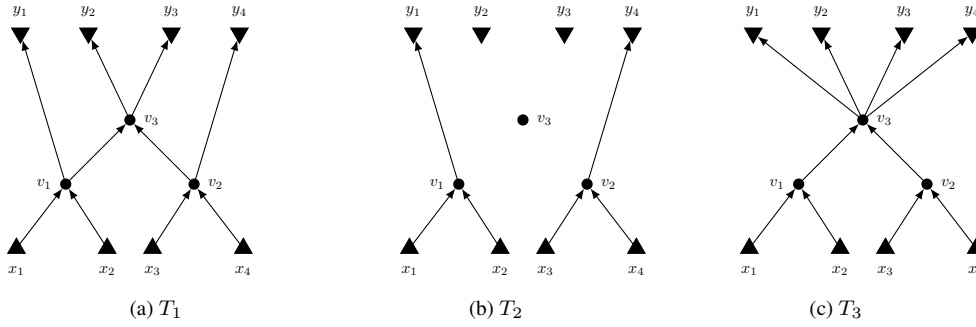


Figure 4.29: Computing lower bound for various absorption-free subgraphs of the bat graph (Figure 2.4). Illustration of Example 4.21.

The next example deals with the absorption-free subgraphs of l-DAG from Figure 6.5(b) .

Example 4.22. Consider absorption-free l-DAG T_1 as on Figure 4.30(a). (resulting from removing (v_2, v_4) , (v_3, v_4) from l-DAG on Figure 6.5(b) If we apply Algorithm 11 to T_1 , it will first eliminate the vertices v_2 , v_3 and v_4 as elimination of these vertices preserves optimality according to Lemma 4.17. The resulting l-DAG has only one intermediate vertex v_1 (compare Figure 4.30(b)) hence Algorithm 11 computes the lower bound as

$$1b = VE(T_1) = \sigma_{T_1}([v_2, v_3, v_4, v_1]) = 2 + 2 + 2 + 4 \cdot 6 = 30 \quad ,$$

which is also the solution of the optimal edge (vertex) elimination problem for T_1 .

If we remove the edges (v_2, v_4) and (v_1, v_4) from the l-DAG on Figure 6.5(b) we get an absorption-free subgraph T_2 as on Figure 4.30(c). If we apply Algorithm 11 to T_2 , it will first eliminate the vertices v_3 and v_2 as elimination of these vertices preserves optimality according to Lemma 4.17. The resulting l-DAG $T_2 - [v_3, v_2]$ is shown on Figure 4.30(d). Then Algorithm 11 eliminates the vertex v_4 as according to Lemma 4.17 the elimination of v_4 on $T_2 - [v_3, v_2]$ preserves optimality. The resulting l-DAG has again only one intermediate vertex. Hence, the lower bound computed by Algorithm 11

$$1b = VE(T_2) = \sigma_{T_1}([v_3, v_2, v_4, v_1]) = 2 + 2 + 2 + 4 \cdot 6 = 30$$

is also the solution of the optimal edge (vertex) elimination problem for T_2 .

The last four examples deal with absorption-free subgraphs of the l-DAGs discussed in [CHMU12]. We will use these results in Chapter 6. For all these examples the lower bound provided by Algorithm 11 is also an optimal solution, meaning that Algorithm 11 only performs optimality preserving eliminations. In the previous examples we often used only the optimality preserving results from Algorithm 5. Therefore in the last four examples we will do it in a more freestyle way and alter the order of application of the optimality preserving eliminations. Doing so we will exploit the critical degree based optimality preserving eliminations more often.

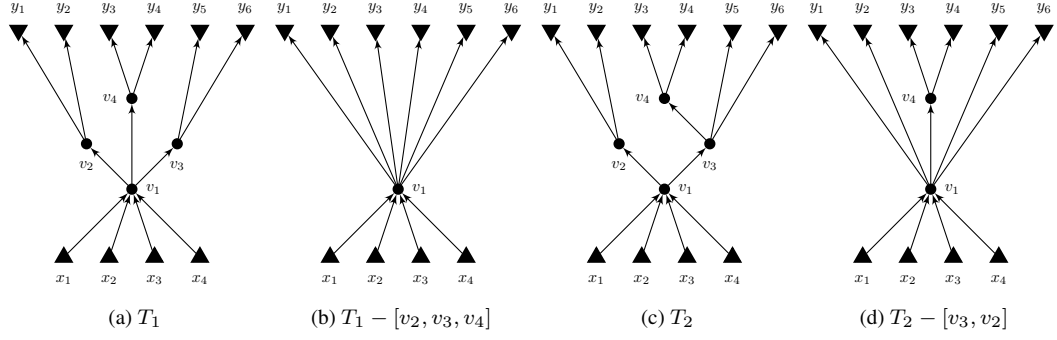


Figure 4.30: Computing lower bound for various absorption-free subgraphs of l-DAG from Figure 6.5(b) (illustrating Example 4.22).

Example 4.23. Consider an absorption-free l-DAG G as on Figure 4.31(a). The edge (v_1, v_2) fulfills the prerequisites of Lemma 4.21 and the edges (x_1, v_2) and (x_4, v_3) fulfill the prerequisites of Lemma 4.22. After the removal of these three edges the l-DAG is completely disconnected and hence

$$1b = VE(G) = \Delta_G((v_1, v_2)) + \Delta_{G_1}((x_1, v_2)) + \Delta_{G_2}((x_4, v_3)) = 2 + 2 \cdot 3 + 3 + 3 = 14 \quad ,$$

where $G_1 = G[E(G) \setminus \{(v_1, v_2)\}]$ and $G_2 = G[E(G_1) \setminus \{(x_1, v_2)\}]$.

Example 4.24. Consider l-DAG G as on Figure 4.31(b). According to Lemma 4.16 the elimination of v_2 and v_3 preserves optimality. The resulting l-DAG $G' = G - [v_2, v_3]$ is isomorphic to the l-DAG from Figure 4.31(a) ($v_2 = v_4$ and $v_3 = v_5$). Hence we can reuse the result from Example 4.23 yielding

$$1b = VE(G) = \sigma_G([v_2, v_3]) + VE(G') = 2 + 1 + 14 = 17 \quad .$$

Example 4.25. Consider l-DAG G as on Figure 4.31(c). First we eliminate the vertices v_3 and v_2 as their elimination preserve optimality according to Lemma 4.16. The resulting l-DAG $G' = G - [v_2, v_3]$ is shown on Figure 4.32(a). According to Lemma 4.16 the elimination of v_4 preserves optimality on G' . The resulting l-DAG $G'' = G' - v_4$ is shown on Figure 4.32(b). According to Lemma 4.16 the elimination of v_1 preserves optimality on G'' . The resulting l-DAG has only one intermediate vertex and hence

$$1b = VE(G) = \sigma_G([v_2, v_3, v_4, v_1, v_5]) = 1 + 1 + 2 + 2 + 8 = 14 \quad .$$

Example 4.26. Consider an absorption-free l-DAG G as on Figure 4.31(d). The l-DAG G has maximal path length 3. All intermediate edges are connected to 2 inputs and 1 output and therefore the critical degree of these edges is $1 + 2 \cdot 1 = 3$. The l-DAG has 8 intermediate edges. After the removal of these edges the outputs are disconnected from the inputs and hence

$$1b = VE(G) = 8 \cdot \Delta_G((v_1, v_5)) = 8 \cdot (1 + 2 \cdot 1) = 24 \quad .$$

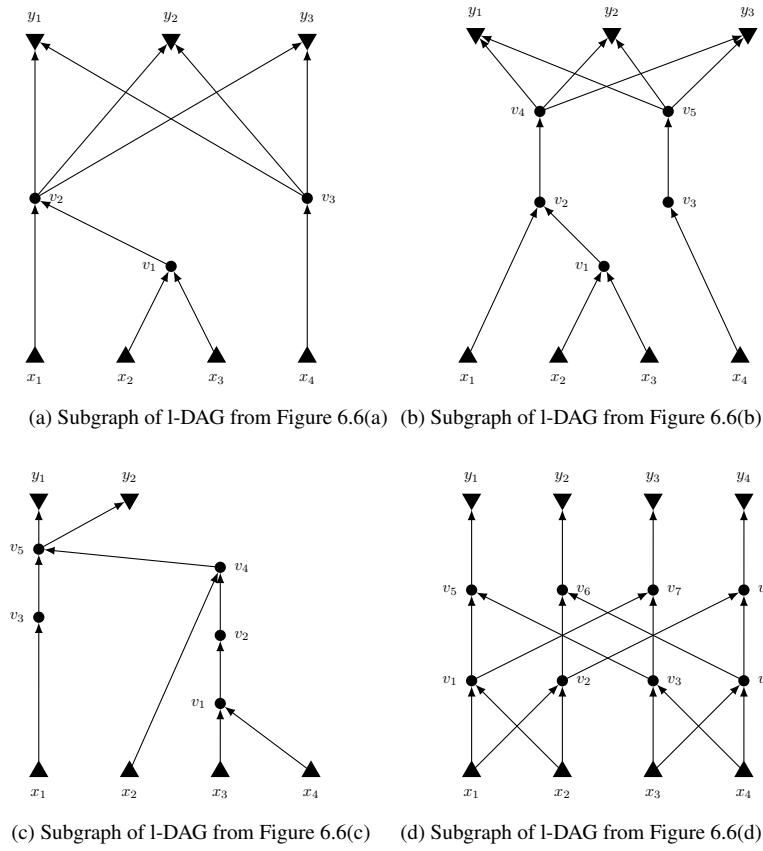


Figure 4.31: Absorption-free subgraphs of the l-DAGs used in [CHMU12]. See Figure 6.6 for original l-DAGs

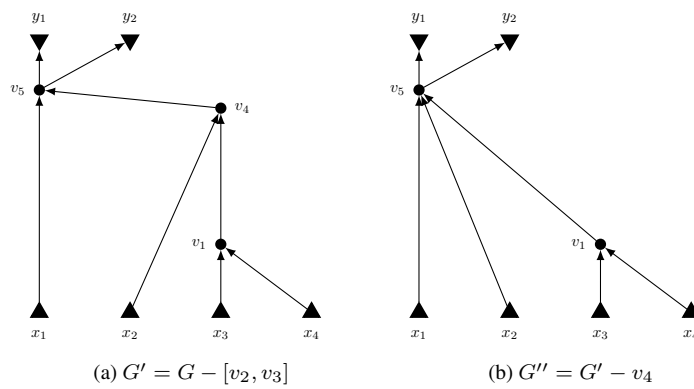


Figure 4.32: Computing a lower bound for l-DAG from Figure 4.31(c) (illustrating Example 4.25).

Chapter 5

Minimal edge count problem on arbitrary l-DAGs

In this chapter we deal with the minimal edge count problem on arbitrary l-DAGs. We present optimality preserving eliminations for both MEC and MECV problems, develop lower bounds for them. We also present the branch and bound algorithm for solving the MEC and MECV problems and show how to use the commutativity of edge and vertex eliminations (proved in Section 3.2) inside this algorithm to reduce the search space.

This chapter is divided into four sections. In the very first section (Section 5.1) the generic branch and bound algorithm (Algorithm 1) is specialized it to solve the MEC and MECV problems. In Section 5.2 we discuss the modifications needed, to employ the commutativity of edge and vertex elimination within the branch and bound algorithm. We apply these modification to two examples and talk about the benefits of this approach. Section 5.3 deals with optimality preserving eliminations for both incarnations of the minimal edge count problem. We present examples showing that optimality preserving eliminations we proved for minimal edge count problem on absorption-free graphs, do not preserve optimality for MECV problem on graphs allowing absorption. We can neither prove nor disprove that these eliminations preserve optimality on arbitrary l-DAGs for MEC problem. Thus we believe that these eliminations do preserve optimality and discuss the reasons for this conjecture. In this section we also prove optimality preserving eliminations based on transformation to a subgraph. These results were already presented in [MN12]. The techniques exploit absorption and are usable for both MEC and MECV problems. The approach for the MEC problem is a generalization of the approach for MECV problem. Thus sometimes we can apply optimality preserving edge eliminations although no optimality preserving vertex eliminations can be applied. Therefore in practice we suggest mixing edge and vertex eliminations instead of solving the pure versions of the problem. This way one can exploit edge elimination to perform more optimality preserving eliminations and then fall back to vertex eliminations once no optimality preserving eliminations can be performed. In the last section of this chapter (Section 5.4) we present lower bounds for MEC and MECV problems. We use the solution of the minimal edge count problem on an absorption-free subgraph as the lower bound. These lower bounds were already published in [MN12]. To the best of my knowledge these are the first lower bounds for these problems presented in the literature.

5.1 Branch and bound algorithm for minimal edge count problem

To modify the generic branch and bound algorithm (Algorithm 1) such that it can solve the minimal edge count problem by vertex and edge eliminations, we need to explain what happens inside the following

routines

- COSTSOFSOLUTION,
- UPDATEUPPERBOUND,
- GETELIMINATIONS,
- COMPUTELOWERBOUND,
- REMOVEPERMUTABLEELIMINATIONS.

In this section we will not discuss the REMOVEPERMUTABLEELIMINATIONS and COMPUTELOWERBOUND routines. Specializations of these routines will be discussed in Section 5.2 and Section 5.4. For the time being we simply assume that REMOVEPERMUTABLEELIMINATIONS leaves the set of possible eliminations M untouched and COMPUTELOWERBOUND routine computes some valid lower bound for the minimal edge count problem on l-DAG G .

The specialization of the first two routines (COSTSOFSOLUTION and UPDATEUPPERBOUND) are the same for both edge and vertex eliminations. To evaluate the costs of a given solution (`solution`) in the context of the minimal edge count problem we obviously need to perform all eliminations contained in `solution` and then count number of edges in the resulting l-DAG. This yields a specialization as in Algorithm 12 we retrieve all eliminations contained in `solution` one by one and apply these eliminations

Algorithm 12 Specialization of COSTSOFSOLUTION for MEC and MECV

```

function COSTSOFSOLUTION( $G$ , solution)
   $H \leftarrow G$ 
  while solution  $\neq \emptyset$  do
    e1  $\leftarrow$  solution.pull()                                // Retrieve elimination
     $H \leftarrow$  PERFORMELIMINATION( $H$ , e1)                    // Apply elimination
  end while
  return  $|E(H)|$ 
end function

```

using the PERFORMELIMINATION routine. The function then returns the number of edges in the resulting graph.

The UPDATEUPPERBOUND routine has to update the upper bound that is used for the recursion in line 16 of Algorithm 1. The new upper bound for minimal edge count problem should be updated according to

$$\min\{\text{ub}, |E(G - \text{e1})|\},$$

where `e1` is either vertex or edge elimination. This yields the specialization as in Algorithm 13.

Algorithm 13 Specialization of UPDATEUPPERBOUND for MEC and MECV

```

function UPDATEUPPERBOUND( $G$ , e1, ub)
   $H \leftarrow$  PERFORMELIMINATION( $G$ , e1)
  return  $\min\{\text{ub}, |E(H)|\}$ 
end function

```

The GETELIMINATIONS routine is the last routine we discuss in this section. Obviously this routine is different for vertex and edge eliminations. As for vertex elimination, the routine should return the set of intermediate vertices of G , yielding specialization as in Algorithm 14. For edge eliminations we need to return all possible front and back edge eliminations, yielding an implementation as in Algorithm 15.

Algorithm 14 Specialization of GETELIMINATIONS for vertex eliminations

```

function GETELIMINATIONS( $G$ )
  return  $Z(G)$ 
end function

```

Algorithm 15 Specialization of GETELIMINATIONS for edge eliminations

```

function GETELIMINATIONS( $G$ )
   $F \leftarrow \{(i, j)^+ \mid i \in X(G) \cup Z(G) \text{ and } j \in Z(G)\}$ 
   $B \leftarrow \{(i, j)^- \mid j \in Y(G) \cup Z(G) \text{ and } i \in Z(G)\}$ 
  return  $F \cup B$ 
end function

```

5.2 Exploit commutativity of edge and vertex eliminations

In this section we discuss how to exploit, within the branch and bound algorithm, the commutativity of edge and vertex eliminations (established in Section 3.2.2) for solving the minimal edge count problem by edge or vertex eliminations. So we will modify the general branch and bound algorithm to use these results. Specifically we show how REMOVEPERMUTABLEELIMINATIONS routine has to be changed. As we already mentioned in Section 3.2.4 the fact that the order of vertex eliminations does not change the resulting l-DAG can be used to reduce the search space dramatically as we do not have to check all possible permutations. Hence we can apply this result not only within branch and bound algorithm, we can even reformulate the original MECV problem and compare only subsets of the intermediate vertices. Nevertheless we will use our original definition of the MECV problem in this section and show the changes to REMOVEPERMUTABLEELIMINATIONS routine required to achieve the same results. The commutativity result for edge eliminations does not have such an impact on the MEC problem as vertex eliminations because in some cases the order of edge eliminations affects the resulting l-DAG. Therefore for the time being I only see its application within some branch and bound algorithm or for proofs.

To exploit commutativity of eliminations we will modify the REMOVEPERMUTABLEELIMINATIONS routine of the branch and bound algorithm (Algorithm 1). First we deal with the MECV problem (compare Algorithm 16). To avoid checking the different permutation we test only those elimination sequences where vertex eliminations are in increasing order. Meaning that $[v_1, \dots, v_n]$, $v_i > v_j$ for all $1 \leq i < j \leq n$. This is ensured by the if statement in the REMOVEPERMUTABLEELIMINATIONS routine in Algorithm 16.

Algorithm 16 Specialization of REMOVEPERMUTABLEELIMINATIONS for MECV problem

```

1: function REMOVEPERMUTABLEELIMINATIONS( $G_p, e1, M$ )
2:   for all  $e \in M$  do
3:     if  $e1 < e$  then                                // avoid elimination sequences that are not in increasing order
4:        $H \leftarrow H \cup \{e\}$ 
5:     end if
6:   end for
7:    $M \leftarrow H$ 
8: end function

```

The REMOVEPERMUTABLEELIMINATIONS routine in Algorithm 16 adds only those candidates to the set H that have a greater index compared to $e1$ (the vertex that was eliminated to transform G_p into the current state). So we will never check an elimination sequence $[a_1, \dots, a_n]$ where $a_i > a_j$ for $1 \leq i < j \leq n$. As already mentioned in Section 3.2.4 using this approach we reduce the search space of

the branch and bound algorithm for the MECV problem from

$$\sum_{k=1}^{|Z(G)|} k! \binom{|Z(G)|}{k}$$

to $2^{|Z(G)|}$.

The next example demonstrates the application of Algorithm 16.

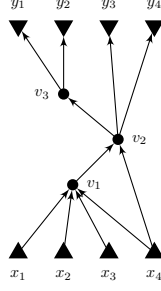


Figure 5.1: L-DAG G used in Example 5.1.

Example 5.1. Consider an l-DAG G as on Figure 5.1. Application of the branch and bound algorithm to solve MECV problem on G , without exploitation of commutativity of vertex elimination yields a search spaces as shown on Figure 5.2(a). Each node in this search tree stands for a possible solution for the MECV problem. The node labels denote the vertices that have been eliminated while the edge labels correspond to the number of edges in the resulting l-DAG. Together with G we have 16 different solutions to check. Let us now see how the Algorithm 16 influences this search space. The reduced search space resulting from application of Algorithm 16 is shown on Figure 5.2(b).

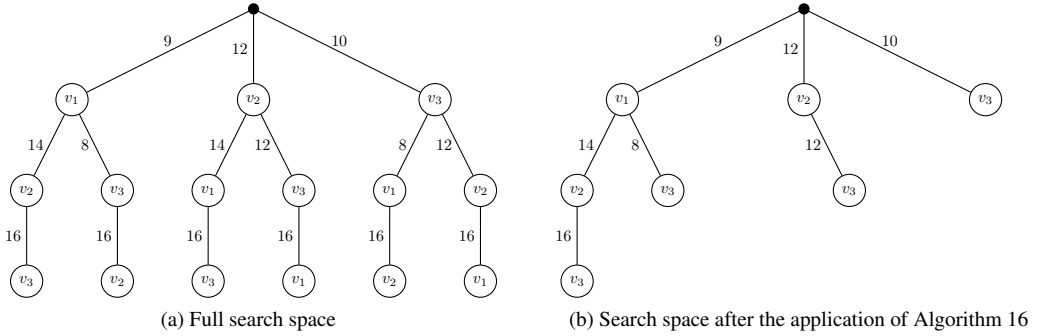


Figure 5.2: Reduced search space of the MECV problem on l-DAG from Figure 5.1 through exploitation of commutativity of vertex eliminations. Illustration of Example 5.1.

On $G - [v_1, v_3]$ the only vertex that can be eliminated is v_2 but $v_3 > v_2$ on $G - v_1$. Hence we do not have to check the elimination of v_2 on $G - v_3$. So we save one solution that has to be checked. $G - [v_2]$ contains two eliminatable vertices v_1 and v_3 but $v_2 > v_1$ on G . Hence we do not need to explore this branch of the search tree. Similarly on $G - [v_2, v_3]$ we do not explore the elimination of v_1 as $v_3 > v_1$ on $G - v_2$. At last we do not perform any eliminations on $G - v_3$ as both vertices v_1 and v_2 has smaller index than v_3 on G . So at the end we need to check only 8 elimination sequences.

So, as you can see, exploiting the commutativity of vertex eliminations reduces the search space significantly.

In order to exploit the commutativity of edge eliminations we must check whether the assumptions of Lemma 3.10, Lemma 3.11 or Lemma 3.12 are fulfilled. If so we know that the order of two edge eliminations does not change the graph remainder and therefore we must test only one order of these edge eliminations. The modified routine REMOVEPERMUTABLEELIMINATIONS is presented in Algorithm 17. The explicit implementation of checking the prerequisites from Lemma 3.10, Lemma 3.11 and Lemma 3.12 is lengthy and does not add anything to the additional understanding. Therefore in Algorithm 17 these checking are done by the call to Lemma 3.10, Lemma 3.11 and Lemma 3.12 routines. We assume that these routines take an l-DAG G and two edge eliminations as input and return **true** if the assumption of the corresponding lemma are valid and **false** if not. If any of the tree routines return **true** we know that the order of eliminating e_{l1} and e does not change the resulting graph and therefore we remove e from the set of candidates H if $e_{l1} < e$.

Algorithm 17 Specialization of REMOVEPERMUTABLEELIMINATIONS for MEC problem

```

function REMOVEPERMUTABLEELIMINATIONS( $G_p, e_{l1}, M$ )
  for all  $e \in M$  do
     $H \leftarrow H \cup \{e\}$ 
    if Lemma 3.10( $G_p, e_{l1}, e$ )  $\vee$  Lemma 3.11( $G_p, e_{l1}, e$ )  $\vee$  Lemma 3.12( $G_p, e_{l1}, e$ ) then
      if  $e_{l1} < e$  then
         $H \leftarrow H \setminus \{e\}$ 
      end if
    end if
  end for
   $M \leftarrow H$ 
end function

```

As already mentioned we cannot quantify the amount of search space reduction that can be achieved exploiting the commutativity of edge eliminations. This depends on the particular l-DAG. In the next example we examine how the usage of Algorithm 17 within the branch and bound algorithm effects the search space for solving the MEC problem for l-DAG from Figure 5.3(a).

Example 5.2. Consider l-DAG G as on Figure 5.3(a). Application of the branch and bound algorithm to solve MEC problem on G , without exploitation of commutativity of edge elimination, yield a search spaces as shown on Figure 5.4(a). Each node in this search tree stands for a possible solution for the MEC problem. The node labels denote the corresponding edge elimination. Together with G we have 36 different solutions to be examined. Let us now see how the application of Algorithm 17 influences this search space. The reduced search space resulting from application of Algorithm 17 is shown on Figure 5.4(b).

On $G - [(v_1, v_2)^+, (x_1, v_2)^+]$ (Figure 5.3(c)) we do not explore the elimination of $(v_1, y_1)^-$ as the order of eliminating $(x_1, v_2)^+$ and $(v_1, y_1)^-$ on $G - (v_1, v_2)^+$ (compare Figure 5.3(b)) does not change the resulting graph according to Lemma 3.12. On $G - [(v_1, v_2)^+, (x_1, v_1)^+]$ (Figure 5.3(d)) the only available edge eliminations are $(x_1, v_2)^+$ and $(v_2, y_1)^-$. According to Lemma 3.10 the order of eliminating $(x_1, v_1)^+$ and $(x_1, v_2)^+$ on $G - (v_1, v_2)^+$ (Figure 5.3(b)) does not change the resulting l-DAG. Similarly according to Lemma 3.12 the order of eliminating $(x_1, v_1)^+$ and $(v_2, y_1)^-$ on $G - (v_1, v_2)^+$ (Figure 5.3(b)) also does not change the resulting l-DAG. Thus we do not have to explore any eliminations on $G - [(v_1, v_2)^+, (x_1, v_1)^+]$ as these cases were already covered in other branches. So together with l-DAG G we compare only 33 different solutions.

As you could see we could not exclude many solutions by exploiting the commutativity of edge elim-

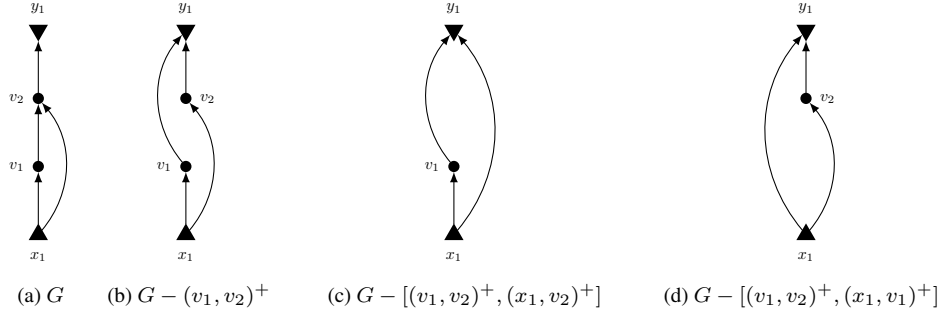


Figure 5.3: Example 5.2.

inations in the previous example. Especially if we compare this performance with results we got from exploiting the commutativity of vertex eliminations. So the question arise whether it makes sense to exploit commutativity of edge eliminations at all. So we need to understand whether this bad performance is inherent to commutativity of edge eliminations or is it just a bad example for exploiting commutativity. If we check the assumptions of Lemma 3.10, Lemma 3.11 and Lemma 3.12 we see that the order of eliminating adjacent edges typically does change the resulting l-DAG, whilst that of not adjacent edges does not. The graph used in the example (Figure 5.3(a)) is small and therefore almost all edges are adjacent. We cannot gain from exploiting the commutativity of edge eliminations on this l-DAG. On bigger l-DAGs we can change the order of edge eliminations more often as more edges are not adjacent. Unfortunately describing the search space for even a slightly bigger example is just not practical. That is why the previous example was chosen. Still, in practice, exploiting the commutativity of edge eliminations provides a significant reduction in search space.

5.3 Optimality preserving eliminations for the minimal edge count problem

In this section we discuss different optimality preserving eliminations that may be used while solving the minimum edge count problem by edge or by vertex elimination. First we discuss optimality preserving eliminations for vertex eliminations (Section 5.3.1) and then those for edge eliminations (Section 5.3.2). For both elimination strategies we prove that elimination sequences transforming an l-DAG on to its subgraph preserve optimality and show some applications of this result.

For vertex elimination we show that optimality preserving eliminations we proved for absorption-free graphs are not applicable on arbitrary l-DAGs. Therefore we present several counter examples where elimination of vertices with one in- or one out-edge do not preserve optimality. For edge eliminations we claim that elimination of vertices with one in- or one out-edge preserve optimality when solving the minimum edge count problem. The proof for this claim still needs to be done.

5.3.1 Vertex elimination

This subsection deals with optimality preserving eliminations for the minimal edge count problem by vertex elimination on arbitrary l-DAGs. In this subsection we prove two results. Firstly we show that the optimality preserving results we developed for absorption-free graphs cannot be applied in the general case. Secondly we develop a different approach for optimality preserving eliminations, based on Theorem 3.3, and prove this approach.

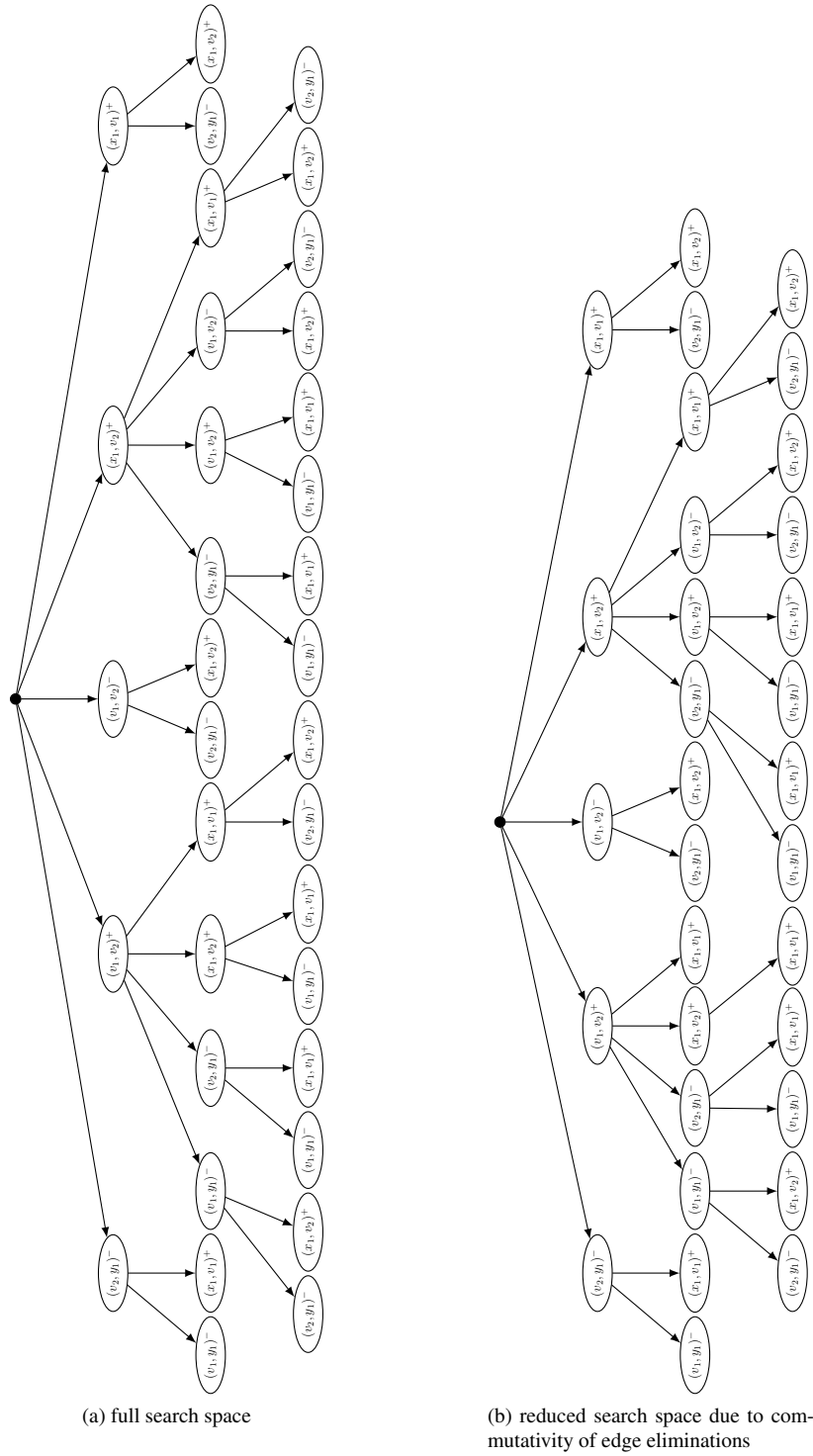


Figure 5.4: Search space for the solving MEC problem on l-DAG from Figure 5.3(a). Illustration of Example 5.2.

For absorption-free l-DAGs we saw that elimination of a vertex with only one in- or one out-edge is an optimality preserving elimination for the MECV problem. On general l-DAGs the elimination of such a vertex does not preserve optimality as the following example shows.

Example 5.3. Consider l-DAG G as on Figure 5.5(a). This l-DAG has 12 edges. The vertex v_1 has only one out edge. Hence if its elimination would preserve optimality for MECV problem then either $[v_1, v_2]$ or $[v_1]$ should solve the MECV problem. $G - v_1$ contains 11 edges (compare Figure 5.5(b)). Obviously $G - [v_1, v_2]$ is a bipartite graph where all five inputs are connected to both output, yielding $|E(G - [v_1, v_2])| = 10$. Hence if elimination of v_1 preserves optimality then $\text{MECV}(G) = 10$. But $|E(G - v_2)| = 9$ (compare Figure 5.5(c)) and therefore the elimination of a vertex with one in- or one out-edge does not preserve optimality for MECV problem on general l-DAGs.

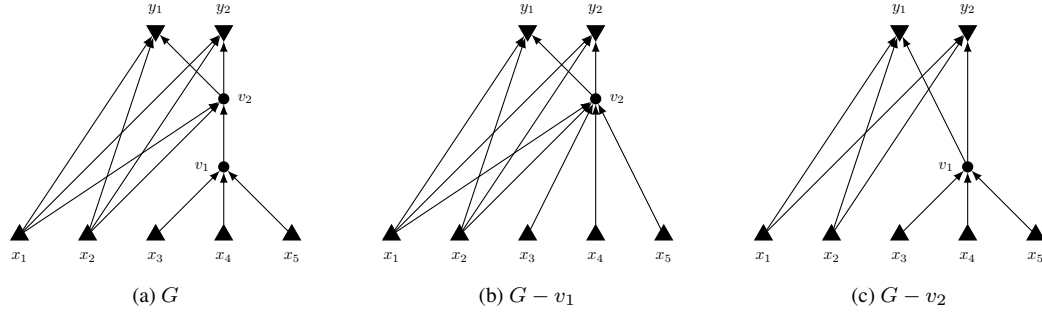


Figure 5.5: Elimination of vertices with one in- or one out-edge does not preserve optimality for MECV problem on arbitrary l-DAGs. Illustration of Example 5.3.

On the previous example we could see why this approach fails. It fails because after the elimination of v_1 we cannot front eliminate the edges (x_1, v_2) and (x_2, v_2) without front elimination of the edges (x_3, v_2) , (x_4, v_2) and (x_5, v_2) . Vertex elimination is simply not fine grain enough at this point. If we could use edge elimination we would simply front eliminate the edges (x_1, v_2) and (x_1, v_2) . The resulting l-DAG $G - [v_1, (x_1, v_2)^+, (x_1, v_2)^+]$ contains also only 9 edges. Moreover as we will see in Example 5.7 elimination sequence $[v_1, (x_1, v_2)^+, (x_1, v_2)^+]$ is also a solution for the MEC problem on l-DAG G . Even elimination of vertices with Markowitz-degree one does not always preserves optimality as the following example demonstrates.

Example 5.4. Consider l-DAG G as on Figure 5.6(a). This l-DAG has 21 edges. The Markowitz-degree of v_2 is one. After elimination of the vertices v_1 and v_3 we get an l-DAG containing 15 edges (compare Figure 5.6(b)). Hence if the elimination of v_2 would preserve optimality there would exist a vertex elimination sequence containing v_2 such that the resulting l-DAG has at most 15 edges. But

- $|E(G - v_2)| = 20$ (compare Figure 5.6(c)),
- $|E(G - [v_2, v_1])| = |E(G - [v_2, v_3])| = 17$ and
- $|E(G - [v_1, v_2, v_3])| = 16$.

Hence elimination of vertices with Markowitz-degree one does not preserve optimality for solving MECV problem in this case.

So as we can see we cannot reuse the optimality preserving results we developed for absorption-free l-DAGs. A completely different approach is needed. Theorem 3.3 ensures that the solution of the minimal

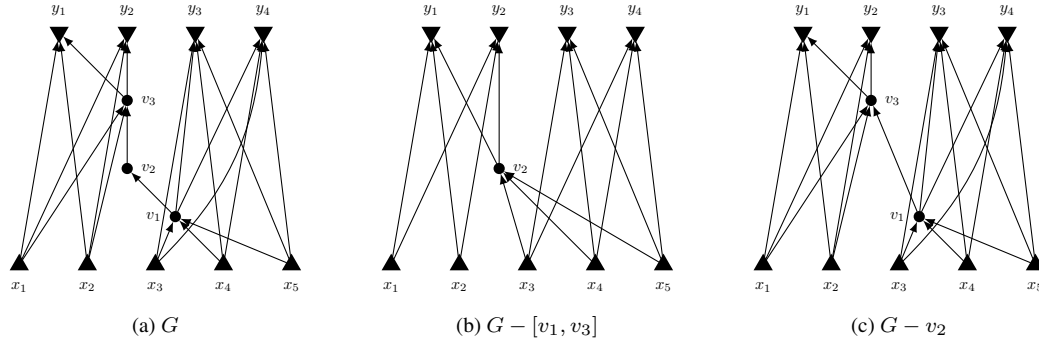


Figure 5.6: Elimination of vertices with Markowitz degree 1 does not preserve optimality for MECV problem on arbitrary I-DAGs. Illustration of Example 5.4.

edge count problem on a subgraph is a lower bound for the number edges of the solution of the minimal edge count problem on the original graph. Hence an elimination sequence transforming an I-DAG on its subgraph preserves optimality. The next theorem is a special case of this idea. There we prove that if elimination of a vertex does not introduce any new edges then elimination of this vertex preserves optimality for solving the minimal edge count problem by vertex eliminations. The proof is based on the result from Theorem 3.3.

Theorem 5.1. *Consider I-DAG G , $v \in Z(G)$, where $(i, j) \in E(G)$ for all $i \in P_G(v)$ and $j \in S_G(v)$ then*

$$\text{MECV}(G) = \text{MECV}(G - v)$$

implying that elimination of the vertex v preserves optimality.

Proof. Obviously

$$\text{MECV}(G) \leq \text{MECV}(G - v) \quad ,$$

as the solution to minimal edge count by vertex elimination on G also allows vertex elimination sequences starting with elimination of vertex v . From $(i, j) \in E(G)$ for all $i \in P_G(v)$ and $j \in S_G(v)$ follows that $G - v \subseteq G$. Hence from Theorem 3.3 follows that $\text{MECV}(G) \geq \text{MECV}(G - v)$ finishing the proof. $\square$

Let us discuss this result on the following example.

Example 5.5. *Consider I-DAG G as on Figure 5.7(a). The elimination of v_2 does not introduce any new edges (compare Figure 5.7(b)) and hence according to Theorem 5.1 its elimination preserves optimality for solving the MECV problem.*

5.3.2 Edge elimination

This subsection deals with optimality preserving eliminations for minimal edge count problem by edge elimination on arbitrary I-DAGs. In this subsection we prove that edge eliminations that do not introduce new edges preserve optimality in the context of minimal edge count problem by edge elimination. This result is a generalization of the result from the previous subsection. We then show that the counter examples, we used to prove that elimination of vertices with one in- or out-edge does not preserve optimality, will not work for edge elimination and discuss the reason. Based on this discussion we formulate the

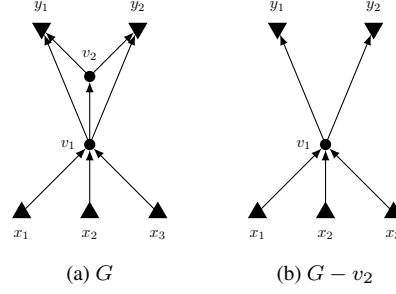


Figure 5.7: Optimality preserving elimination according to Theorem 5.1 (illustrating Example 5.5).

claim that for minimal edge count problem by edge elimination the removal of vertices with one in- or one out-edge preserve optimality. Unfortunately we do not have the proof for this belief so far.

First of all we prove optimality preserving results similar to Theorem 5.1 for edge elimination. These results are based on the fact that if an edge elimination does not generate new edges than such an elimination preserves optimality. First we prove the result for forward edge elimination.

Theorem 5.2. Consider l-DAG G $(i, j) \in E(G)$, where $(i, k) \in E(G)$ for all $k \in S_G(j)$ then

$$\text{MEC}(G) = \text{MEC}(G - (i, j)^+)$$

implying that front elimination of (i, j) preserves optimality.

Proof. Obviously

$$\text{MEC}(G) \leq \text{MECV}(G - (i, j)^+) \quad ,$$

as the solution to minimal edge count by edge elimination on G also allows edge elimination sequences starting with front elimination of (i, j) . From $(i, k) \in E(G)$ for all $k \in S_G(j)$ follows that $G - (i, j)^+ \subseteq G$. Hence from Theorem 3.3 follows that $\text{MEC}(G) \geq \text{MEC}(G - (i, j)^+)$ finishing the proof. $\square$

The similar theorem holds for the back elimination too.

Theorem 5.3. Consider l-DAG G $(i, j) \in E(G)$, where $(k, j) \in E(G)$ for all $k \in P_G(j)$ then

$$\text{MEC}(G) = \text{MEC}(G - (i, j)^-)$$

implying that back elimination of (i, j) preserves optimality.

Proof. The proof is symmetric to the proof of Theorem 5.2. $\square$

In the following example we apply the Theorem 5.2 and Theorem 5.3 to the l-DAG from Example 5.5.

Example 5.6. Consider l-DAG G as on Figure 5.7(a). The front elimination of (v_1, v_2) does not introduce any new edges (compare Figure 5.7(b)). Hence front elimination of (v_1, v_2) preserves optimality for MEC problem according to Theorem 5.2 (compare Figure 5.7(b)). Similarly the back elimination of both edges (v_2, y_1) and (v_2, y_2) on G does not introduce new edges (compare Figure 5.7(b)) and therefore these transformations also preserve optimality for MEC problem according to Theorem 5.3.

As already discussed edge elimination is more generic operation than vertex elimination. The optimality preserving results in Theorem 5.2 and Theorem 5.3 are also more generic versions of Theorem 5.1 as vertex elimination is defined as front elimination of all incoming or back elimination of all outgoing edges. Thus it is more likely that we will find a situation where an edge elimination will not generate new edges and thus preserve optimality. At this point solving an MEC problem might become less complex than solving the MECV problem. Let us discuss this observation on the l-DAG from Example 5.3.

Example 5.7. Consider l-DAG G as on Figure 5.5(a). Neither v_1 nor v_2 satisfy the prerequisites of Theorem 5.1. So we have to try all possible vertex elimination sequences to solve the MECV problem. If we want to solve the MEC problem we can front eliminate the edges (x_1, v_2) and (x_2, v_2) as these eliminations preserve optimality according to Theorem 5.2. The resulting l-DAG is absorption-free (compare Figure 5.8(a)). Hence elimination of v_2 (compare Figure 5.8(c)) or v_1 (compare Figure 5.8(b)) solves the MEC problem in this case. Thus both elimination sequences $[(x_1, v_2)^+, (x_1, v_2)^+, v_2]$ and $[(x_1, v_2)^+, (x_1, v_2)^+, v_1]$ are solutions of the MEC problem on G .

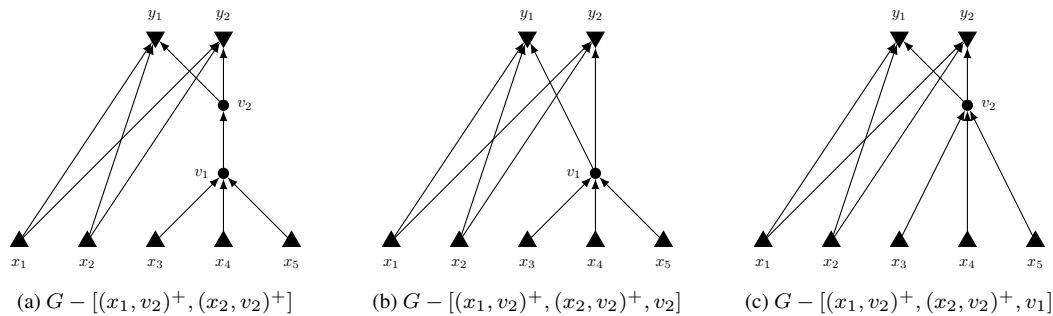


Figure 5.8: Illustration of Example 5.7

So as we can see on the previous example, despite the fact that edge elimination yields a much bigger search space than vertex elimination, sometimes it might be "easier" to solve the minimum edge count problem by edge elimination rather than by vertex. This is especially so if we can use optimality preserving edge eliminations to transform the l-DAG in absorption-free graph where we can fall back to vertex elimination again. The following example demonstrates this observation.

Example 5.8. Consider l-DAG G as on Figure 5.6(a). According to Theorem 5.2 and Theorem 5.3 front elimination of edges (x_1, v_2) and (x_2, v_2) as well as back elimination of the edges (v_1, y_3) and (v_1, y_4) preserve optimality for MEC problem. Performing all these eliminations yield an l-DAG $G' = [(x_1, v_3)^+, (x_2, v_3)^+, (v_1, y_3)^-, (v_1, y_4)^-]$ as on Figure 5.9(a). G' is absorption-free l-DAG and hence elimination of v_1 and v_2 solves the MEC problem on G' (compare Figure 5.9(b)). So we can easily solve the MEC problem, while solving the MECV problem would require testing all possible vertex elimination sequences.

As you might have noticed the examples we used to show that elimination of vertices with one in- or one out-edge do not preserve optimality when solving the MECV problem cannot be applied to the MEC problem. Basically we simply have to apply the same trick we used in the proofs of Lemma 4.16 and Lemma 4.17 to make the examples fail. Edge elimination is flexible enough and allows the required transformation to be performed.

On l-DAG from Figure 5.5(a) the elimination sequence $[v_1, (x_1, v_2)^+, (x_2, v_2)^+]$ solves the MEC problem as it transforms this graph into the l-DAG as on Figure 5.8(c). The most important step is that after

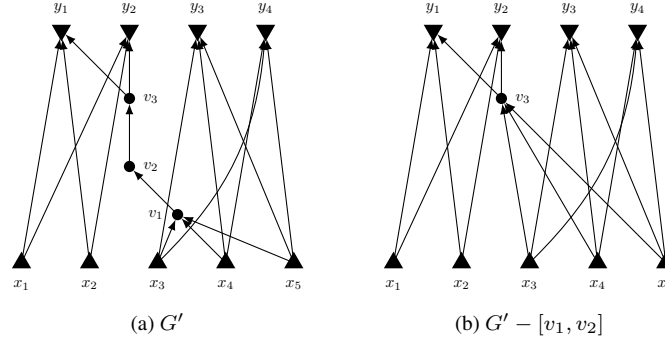


Figure 5.9: Illustration of Example 5.8

the elimination of v_1 (compare Figure 5.5(b)) we can still front eliminate the edges (x_1, v_2) and (x_2, v_2) without front elimination of other incoming edges of v_2 .

The reason for the failure of the second counter example (Example 5.4) is similar. The edge elimination sequence $[v_2, v_1, (x_1, v_3)^+, (x_2, v_3)^+]$ transforms the l-DAG on Figure 5.6(a) onto l-DAG as on Figure 5.9(b) and thus this eliminations sequence also solves the MEC problem for this graph. This counter example fails because the edge elimination allows us to perform the front elimination of the edges (x_1, v_3) and (x_2, v_3) while leaving the other edges arriving at v_3 untouched.

Therefore I believe that unlike MECV problem, elimination of vertices with one in- or one out-edge do preserve optimality for the MEC problem. Unfortunately I do not have a proof for this claim so far.

Conjecture 1. Consider l-DAG G . The following holds for all $a \in \mathcal{M}(G)$

$$\text{MEC}(G - a) = \text{MEC}(G) \quad .$$

Before we switch to lower bounds I would like to summarize the results of this subsection.

I do not think that we can gain a lot by trying to address the MEC problem instead of solving the MECV problem. The much bigger search space of edge elimination is a big issue. Still we should not try and solve the pure MECV problem. Instead we should exploit the optimality preserving results for MEC problem (Theorem 5.2 and Theorem 5.3) whenever we can. This approach can reduce the actual search space for the problem, while producing results that are at least as good as that from solving pure MECV problem. Mixing edge and vertex eliminations this way, would become even more advantageous if we can prove Conjecture 1 and use this result.

5.4 Lower bounds

In this section we develop lower bounds for MEC and MECV problems on arbitrary l-DAGs. Lower bounds are required to improve the performance of various algorithms solving the minimal edge count problem by edge or vertex elimination. For example in [CHMU12] the authors asked for lower bounds for these problems to speed up their solver. Lower bounds can be also useful if we want to prove that the actual solution solves the MEC or MECV problem. To the best of my knowledge these are the first lower bounds for MEC and MECV problems presented in the literature.

This section is structured as follows. First we prove the theoretical basis for the lower bounds (Theorem 5.4). We then use this result to prove that the solution of the minimal edge count problem for l-DAG

on Figure 2.3(a) contains 8 edges (Example 5.9). Finally we develop the algorithm to compute the lower bounds for MEC problem (Algorithm 18) and demonstrate its application.

The idea behind the lower bounds presented here is to exploit Theorem 3.3. This result allows us to reduce the problem to a subgraph of the original l-DAG. Meaning that we can use the solution of MEC (MECV) problem on the subgraph as a lower bound for the solution of MEC (MECV) problem on the original l-DAG. The hope is that solving the problem on a subgraph of the original l-DAG G can be done efficiently. So we need a way to decide what kind of subgraph we should chose. Since we already know how to solve the MEC (MECV) problem on absorption-free l-DAGs in linear time, we should reduce the problem to an absorption-free subgraph.

Theorem 5.4. *Consider an l-DAG G and let G' be an absorption-free subgraph of G . Then*

$$\text{MEC}(G) \geq \text{MEC}(G') \quad \text{and} \quad \text{MECV}(G) \geq \text{MECV}(G')$$

Proof. From Theorem 3.3 follows that $\text{MEC}(G) \geq \text{MEC}(G')$. Therefore we only need to show the inequality for vertex elimination. We already know that $\text{MECV}(G) \geq \text{MEC}(G)$. As G' is absorption-free l-DAG and therefore according to Theorem 4.3 $\text{MEC}(G') = \text{MECV}(G')$. Hence

$$\text{MECV}(G) \geq \text{MEC}(G) \geq \text{MEC}(G') = \text{MECV}(G') \quad ,$$

finishing the proof. □

With the help of Theorem 5.4 we can show that solution of problem of the l-DAG G from Figure 2.3(a) has 8 edges.

Example 5.9. *Consider l-DAG G as on Figure 5.10(a). In Section 2.1 we have already shown that $\text{MEC}(G) \leq 8$. Hence to prove that $\text{MEC}(G) = 8$ it is sufficient to show that the minimum edge count of this DAG cannot become less than 8. The absorption-free l-DAG G' shown in Figure 5.10(b) is a subgraph of G . Moreover $\mathcal{M}(G') = \emptyset$ implying that*

$$\text{MEC}(G) \geq \text{MEC}(G') = |E(G')| = 7 \quad .$$

From Theorem 3.2 follows that if $\text{MEC}(G) = 7$, then there should exist an elimination sequence transforming G into one of the graphs solving the MEC problem for G' , namely G' , $G' - v_1$ (compare Figure 5.10(c)), or $G' - v_2$ (compare Figure 5.10(d)). The elimination of vertex v_1 or v_2 on G introduces a not eliminatable edge $(x_3, y_1) \notin G'$. Hence we cannot transform G to $G' - v_1$ or $G' - v_2$. Transformation of G into G' requires back elimination of the edge (v_2, y_1) on G . However this elimination also generates $(x_3, y_1) \notin G'$. Hence $\text{MEC}(G) > 7$ and therefore $\text{MEC}(G) = 8$.

Next we formulate the algorithm to compute the lower bound for the MEC and MECV problems. As already mentioned the algorithm is based on Theorem 5.4. So to determine the lower bound for MEC or MECV problem for a given l-DAG we first build an absorption-free subgraph of this l-DAG and then use the solution of the MEC problem on this subgraph as the lower bound for MEC or MECV problem on the original l-DAG. This approach is formalized in Algorithm 18. As you can see Algorithm 18 first builds

Algorithm 18 Lower Bound MEC and MECV problem

Require: l-DAG G

Ensure: lower bound for MEC problem

- 1: $G' \leftarrow \text{ALGORITHM 3}(G)$
 - 2: $H \leftarrow \text{ALGORITHM 4}(G')$
 - 3: **return** $|E(H)|$
-

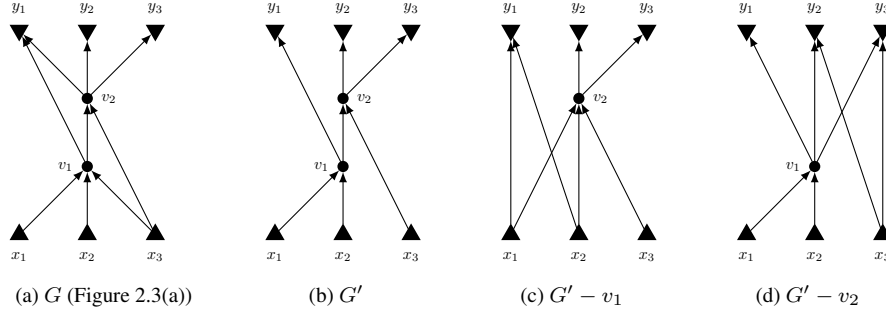


Figure 5.10: The solution of the MEC problem for l-DAG from Figure 2.3(a) contains 8 edges (illustrating Example 5.9).

an absorption-free subgraph of the given l-DAG using the Algorithm 3 and then solves the minimal edge count problem on this subgraph calling Algorithm 4. The complexity of Algorithm 18 is equal to the sum of the cost of Algorithm 3 and Algorithm 4. Complexity of Algorithm 3 is $O((|E(G) + |V(G)||) \cdot |Y(G)| \cdot |E(G)|)$ while the complexity of Algorithm 4 is linear in the number of vertices ($O(|V(G)|)$). Hence the complexity of Algorithm 18 is

$$O((|E(G) + |V(G)||) \cdot |Y(G)| \cdot |E(G)| + |V(G)|) .$$

Let's apply the Algorithm 18 to the lion graph.

Example 5.10. Consider lion graph G . If we apply Algorithm 18 to this l-DAG we first use Algorithm 3 to build an absorption-free subgraph of G . Depending on what weighting we choose we will end up with one of absorption-free subgraphs presented on Figure 5.11. According to Algorithm 4 the elimination of

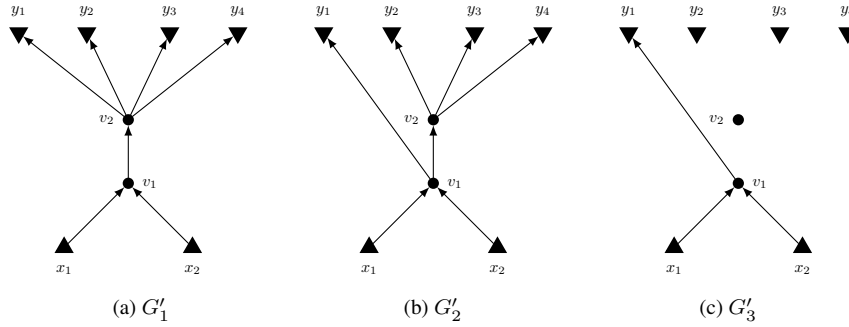


Figure 5.11: Possible absorption-free subgraphs of the lion graph (Example 5.10).

v_2 solves the MEC problem for G'_1 and G'_2 (compare Figure 5.11(a) and Figure 5.11(b)). Hence in this case we will get a lower bound of 6. This result is actually a solution to the MEC problem on the lion graph as the back elimination of (v_2, y_1) transforms the lion graph in to G'_2 .

The choice of the underlying absorption-free subgraph is crucial for the quality of the lower bounds produced by the Algorithm 18 as shown in the next example.

Example 5.11. Consider lion graph G . If we choose a different, more disadvantageous weighting on the l-DAG G Algorithm 3 could also remove the edge (v_1, v_2) yielding the subgraph G'_3 as in Figure 5.11(c). (Note we have to remove edges emanating from vertex v_2 as this vertex has no incoming edges). According to Algorithm 4 the elimination of v_1 solves the MEC problem for G'_3 yielding a lower bound of 2. As we can see the lower bound is worse than the one we saw in Example 5.10.

To summarize the results of this section we developed first lower bounds for the MEC and MECV problems. To get good results we need to find a good heuristic to weight the l-DAG before the application of Algorithm 3.

Chapter 6

Vertex and edge elimination problems on arbitrary l-DAGs

In this chapter we collect results for vertex and edge elimination problems on arbitrary l-DAGs. We discuss optimality preserving eliminations, develop new lower bounds for EE and VE problems and show how to speed up the branch and bound algorithm by exploiting the commutativity of edge and vertex eliminations.

Similar to Chapter 5 we start this chapter discussing ways of exploiting the commutativity of edge and vertex eliminations, when solving the optimal vertex and edge elimination problems. Therefore in the first section of this chapter (Section 6.1) we show how to modify the generic branch and bound algorithm (Algorithm 1) to solve the VE and EE problems. We then show the changes that have to be made to this algorithm to exploit the commutativity of edge and vertex eliminations. These techniques were first presented in the bachelor thesis of Elmar Peise ([Pei09]) that I supervised.

In the next section (Section 6.3) we discuss various algorithms to determine lower bounds for both VE and EE problems. Unlike the minimal edge count problem, various algorithms for computing lower bounds are known for these two problems. Thus in this section we will compare the existing algorithms with the approach based on reduction to absorption-free graphs developed in this thesis. To ensure a fair comparison I will use not only my examples but also examples mentioned in the [CHMU12]. In this publication lower bounds were used to speed up the branch and bound algorithm solving the integer programming representation of the VE problem. This section is split into two subsections. In the first (Section 6.3.1) we talk about so called "simple lower bounds". These are bounds that are computed using only the number of intermediate vertices and eliminatable edges. Therefore they can be computed almost instantly. Although these bounds are not very sharp, they are still very useful to perform cuts in the branch and bound algorithm on small graphs. So we can avoid computing more advanced and expensive lower bounds. The simple bounds were also used in the bachelor thesis of Elmar Peise to perform the cuts in the branch and bound algorithm when solving the VE and EE problems. In the second subsection (Section 6.3.2) we discuss the more advanced lower bounds for VE and EE problems. The lower bounds proposed there are based on reduction to absorption-free subgraphs. These bounds are compared with the lower bounds used in [CHMU12] on various examples. The lower bounds discussed in this subsection were also part of research in the master thesis of Tarek El Habbaj ([EH12]) written under my supervision. In the master thesis the suggested lower bounds were used to speed up the solution of the VE and EE problems by the branch and bound algorithm.

In Section 6.4 we use the software developed in [Pei09] and extended in [EH12] to show the impact of the results from the previous two sections (Section 6.2 and Section 6.3) on the search space of the branch and bound algorithm when solving the VE and EE problems.

The last section (Section 6.5) of this chapter deals with the optimality preserving eliminations on arbitrary l-DAGs. In this section we discuss the existing optimality preserving results for the VE and EE problems. We show the reason why the strategies developed for absorption-free graphs fail on l-DAGs with parallel paths. Then we prove new underestimators for the critical degree of edge and use this results in conjunction with Lemma 4.18 to prove new optimality preserving eliminations for arbitrary l-DAGs. These optimality preserving eliminations were already published in [MN12].

6.1 Branch and bound algorithm to solve the optimal edge and vertex elimination problems

In this section we specialize the Algorithm 1 to use it for solving the optimal edge and vertex elimination problems. To specialize the algorithm we need to explain what happens inside the following routines

- COSTSOFSOLUTION,
- UPDATEUPPERBOUND,
- GETELIMINATIONS,
- COMPUTELOWERBOUND,
- REMOVEPERMUTABLEELIMINATIONS.

Similar to Section 5.1 we will not discuss the changes needed for REMOVEPERMUTABLEELIMINATIONS and COMPUTELOWERBOUND routines in this section. Specializations of these routines will be discussed in Section 6.2. For the time being we simply assume that REMOVEPERMUTABLEELIMINATIONS leaves the set of possible eliminations M untouched and COMPUTELOWERBOUND routine computes some valid lower bound for the optimal edge or vertex elimination problem on l-DAG G .

The specialization of the first two routines from the list (COSTSOFSOLUTION and UPDATEUPPERBOUND) are the same for both edge and vertex eliminations. The costs of a solution in the context of optimal edge (vertex) elimination problem is the sum of the costs of all eliminations contained in solution, yielding a specialization as in Algorithm 19. So in Algorithm 19 we retrieve an elimination

Algorithm 19 Specialization of COSTSOFSOLUTION for EE and VE

```

function COSTSOFSOLUTION( $G$ , solution)
  costs  $\leftarrow$  0
   $H \leftarrow G$ 
  while solution  $\neq \emptyset$  do
     $e1 \leftarrow$  solution.pull()
    costs  $\leftarrow$  costs +  $\sigma_H(e1)$ 
     $H \leftarrow$  PERFORMELIMINATION( $H, e1$ )
  end while
  return costs
end function

```

from the queue as long as the queue is not empty, compute the cost of performing this elimination and then apply this elimination (Note $e1$ can be a vertex or an edge elimination). At the end the function returns the computed costs.

The UPDATEUPPERBOUND routine has to update the upper bound that is used for the recursion in line 16 of Algorithm 1. To get a valid solution for G we need to find an elimination sequence for

$G' = G - e1$ such that the costs of this elimination sequence are not higher then $ub - \sigma_G(e1)$. Hence we have to update the upper bound according to

$$ub - \sigma_G(e1) \quad ,$$

yielding the specialization as in Algorithm 20.

Algorithm 20 Specialization of UPDATEUPPERBOUND for EE and VE

```

function UPDATEUPPERBOUND( $G, e1, ub$ )
    return  $ub - \sigma_G(e1)$ 
end function

```

Similar to the minimal edge count problem the GETELIMINATIONS is different for vertex and edge eliminations. For vertex elimination the routine should return the set of intermediate vertices of G while for edge eliminations we need to return all possible front and back edge eliminations. So for optimal edge and vertex elimination problems we can reuse the specialization of the GETELIMINATIONS routine we developed for the minimal edge count problem by edge and vertex eliminations respectively.

6.2 Exploiting commutativity of eliminations

In the Section 3.2.5 we discussed situations where the order of eliminations does not change the resulting l-DAG. In the same subsection we also proved under which circumstances the order of edge or vertex eliminations does not influence the costs. In Section 4.3 we used these outcomes to proof various claims. In this section we show how to exploit the commutativity of eliminations to reduce the search space of the branch and bound algorithm. Similarly to Section 5.2 we will specialize the REMOVEPERMUTABLEELIMINATIONS routine for optimal vertex and edge elimination problems. The exploitation of the commutativity in the context of these problems is more complicated, as we not only have to ensure that the order of eliminations does not change the resulting l-DAG, we additionally need to take care of the costs of the elimination sequences which can be different as shown in Example 3.7 and Example 3.10. Therefore in addition to the modifications we made for MEC and MECV problems, ensuring that the resulting l-DAG is the same, we add instructions handling the potential changes in the costs. As we will see, similarly to the minimal edge count problem, the exploitation of the commutativity of eliminations in the context of the optimal vertex elimination problem is more effective than that for the optimal edge elimination problem.

This section is split into two subsections. In the first subsection we show how to exploit commutativity within the branch and bound algorithm for the optimal vertex elimination problem. The second subsection deals with commutativity for optimal edge elimination problem. In both subsections we first handle the cases where the order of eliminations does not change the resulting costs. We then extend these routines to exploit the commutativity even if the costs change.

6.2.1 Exploiting commutativity of vertex eliminations

In this subsection we discuss how the commutativity of vertex elimination can be used to reduce the search space of the branch and bound algorithm. We already know that the order of vertex eliminations does not change the resulting l-DAG. We will now use this result to reduce the search space for the branch and bound algorithm (Algorithm 1) specialized for the optimal vertex elimination problem. Although the resulting l-DAG does not depend on the order of vertex eliminations, the corresponding costs do. The costs are not effected if we change the order of two subsequent vertex eliminations where the vertices are not adjacent (compare Lemma 3.13). If we try to change the order of adjacent vertices, the costs

can change (e.g. Example 3.7). First we modify REMOVEPERMUTABLEELIMINATIONS to exploit the commutativity for vertex elimination for not adjacent vertices. We will then extend this routine to deal with the adjacent vertices too.

Not-adjacent Vertices

To exploit the result that the vertex eliminations order of not adjacent vertices does not change the resulting l-DAG nor the costs of elimination, we apply the same idea as in Algorithm 16. We ensure that the vertex eliminations are increasing order in the case that the vertices are not adjacent. This way we ensure that we test only one sequential arrangement of the vertex eliminations for not adjacent vertices. The resulting routine is shown in the Algorithm 21. The only difference to the corresponding routine for

Algorithm 21 Specialization of REMOVEPERMUTABLEELIMINATIONS for VE problem. No handling for adjacent vertices

```

1: function REMOVEPERMUTABLEELIMINATIONS( $G_p, e_l, M$ )
2:   for all  $e \in M$  do
3:     if  $(e, e_l) \in E(G_p) \vee (e_l, e) \in E(G_p)$  then                                // vertices are adjacent
4:        $H \leftarrow H \cup \{e\}$ 
5:     else if  $(e_l < e)$  then // elimination sequence in increasing order for not adjacent vertices
6:        $H \leftarrow H \cup \{e\}$ 
7:     end if
8:   end for
9:    $M \leftarrow H$ 
10: end function

```

the MECV problem (compare Algorithm 16) is the check for adjacent vertices in line 3. If the vertices are adjacent, we add the vertex to H (the set of possible eliminations). Hence in this case we ensure that we test both elimination orders. Lets see how this routine influence the search space of the branch and bound algorithm applied to l-DAG from Figure 5.1.

Example 6.1. The l-DAG from Figure 5.1 has 3 intermediate vertices and therefore we have to test six possible vertex eliminations sequences to solve the optimal vertex elimination problem on this graph. Hence, if we do not use any pruning techniques to reduce the search space in the branch and bound algorithm, we get a search tree as presented on Figure 6.1(a). The edge labels are the costs of eliminating the corresponding vertex. Application of Algorithm 21 yields the reduced search space as shown on Figure 6.1(b). We do not achieve the same search space reduction as for the MECV problem (compare Figure 5.2(b)) due to the restriction that the vertices should not be adjacent. The vertices v_1 and v_3 are not adjacent, hence if we apply Algorithm 21 we will not explore the elimination sequence starting with $[v_3, v_1]$ as we know that it will have the same costs as the elimination sequence starting with $[v_1, v_3]$. Still we do have to check both elimination sequences $[v_2, v_1, v_3]$ and $[v_2, v_3, v_1]$ as v_3 and v_1 are adjacent on $G - v_2$ and hence the costs of these elimination sequences can differ. As the matter of fact the costs of both elimination sequences are:

$$\sigma_G([v_2, v_1, v_3]) = 6 + 12 + 8 = 26 \quad \text{and} \quad \sigma_G([v_2, v_3, v_1]) = 6 + 2 + 16 = 24 \quad .$$

We also have to compare the elimination sequences $[v_1, v_2, v_3]$ and $[v_1, v_3, v_2]$ as v_2 and v_3 are adjacent on G and thus on $G - v_1$. Similarly as v_2 and v_1 are adjacent on G , we also check the elimination sequence starting $[v_2, v_1]$ although $v_2 > v_1$. So using the Algorithm 21 we can reduce the search space by one elimination sequence.

So we need to handle the adjacent vertices somehow to get similar results as for the MECV problem.

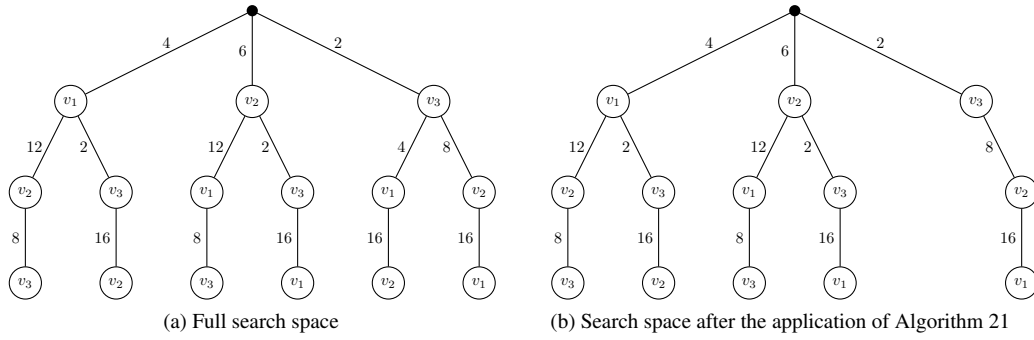


Figure 6.1: Reducing the search space of the optimal vertex elimination problem on l-DAG from Figure 5.1 by exploiting the commutativity of vertex eliminations for not adjacent vertices. Illustration of Example 6.1.

Adjacent Vertices

In the following we employ Lemma 3.14 to improve the Algorithm 21. In this lemma we exploit the fact that the order of vertex eliminations does not change the resulting l-DAG. So if we want to compare the costs of two vertex elimination sequences where the order of two subsequent vertex eliminations was swapped, we only need to compare the costs of these two eliminations. For example we compare the costs of eliminating the vertices v_1 and v_2 from l-DAG as on Figure 5.1 in both orders. The elimination of v_1 followed by v_2 costs 16 fma's while as the elimination of v_2 followed by v_1 needs 18 (compare Figure 6.1(a)). Hence any elimination sequence on this l-DAG starting with $[v_2, v_1]$ cannot be optimal.

In Algorithm 22 we use this idea. Therefore we extend Algorithm 21 to check which order of vertex

Algorithm 22 Specialization of REMOVEPERMUTABLEELIMINATIONS for VE problem with handling of adjacent vertices

```

1: function REMOVEPERMUTABLEELIMINATIONS( $G_p, e1, M$ )
2:   for all  $e \in M$  do
3:     if  $(e, e1) \in E(G_p) \vee (e1, e) \in E(G_p)$  then                                // vertices are adjacent
4:       if  $\sigma_{G_p}([e1, e]) \leq \sigma_{G_p}([e, e1])$  then                            // actual elimination sequence is better
5:          $H \leftarrow H \cup \{e\}$ 
6:       end if
7:       if  $\sigma_{G_p}([e1, e]) = \sigma_{G_p}([e, e1]) \wedge (e1 > e)$  then
8:          $H \leftarrow H \setminus \{e\}$                                                 // avoid double checking if the costs are equal
9:       end if
10:      else if  $(e1 < e)$  then // elimination sequence in increasing order for not adjacent vertices
11:         $H \leftarrow H \cup \{e\}$ 
12:      end if
13:    end for
14:     $M \leftarrow H$ 
15: end function

```

eliminations yields smaller costs for adjacent vertices. This happens in line 4. Thus if $\sigma_{G_p}([e1, e]) \leq \sigma_{G_p}([e, e1])$, we add e to H (the set of possible eliminations). In line 7 we ensure that if the elimination order of adjacent vertices does not change the overall costs, we check only one possible order of eliminations. Therefore we remove e from H if $e1 > e$. For not adjacent vertices we do not need this check as

the costs are always the same. In the next example we discuss this approach.

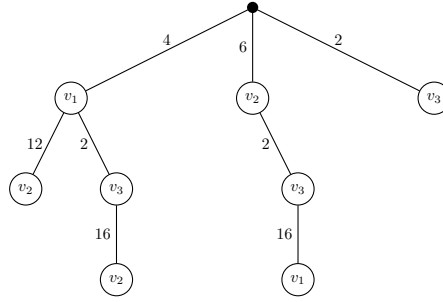
Example 6.2. The l-DAG from Figure 5.1 has 3 intermediate vertices and therefore we have to test six complete vertex eliminations sequences to solve the optimal vertex elimination problem on this graph. Hence, if we do not use any pruning techniques to reduce the search space in the branch and bound algorithm, we get a search tree as presented on Figure 6.1(a). Application of Algorithm 22 yields a search space as on Figure 6.2(a). In addition to the search space reductions presented on Figure 6.1(b) we do not have to check the elimination sequences starting with $[v_3, v_2]$ as

$$\sigma_G([v_3, v_2]) = 2 + 8 = 10 > 8 = 6 + 2 = \sigma_G([v_2, v_3]) \quad .$$

We already discussed that $\sigma_G([v_2, v_1]) > \sigma_G([v_1, v_2])$ implying that we do not need to check sequences starting with $[v_2, v_1]$. We also do not explicitly check the elimination sequence $[v_1, v_2, v_3]$ as

$$\sigma_{G-v_1}([v_2, v_3]) = 12 + 8 = 20 > 18 = 2 + 16 = \sigma_{G-v_1}([v_3, v_2]) \quad .$$

Of course we do compute the costs for both elimination sequences but we do not explore the corresponding branch of the search tree explicitly. Meaning we do not add the vertex v_3 to the set of possible eliminations on $G - [v_1, v_2]$.



(a) Search space after application of Algorithm 22

Figure 6.2: Reducing the search space of the optimal vertex elimination problem on l-DAG from Figure 5.1 by exploiting the commutativity of vertex eliminations with adjacent vertices. Illustration of Example 6.2.

As we can see Algorithm 22 does not allow the same amount of search space reduction as Algorithm 16 when solving the MECV problem. For example let us compare the search trees for solving the optimal vertex elimination problem and the MECV problem on l-DAG from Figure 5.1. The search space tree for the optimal vertex elimination problem has two paths with length 3 (compare Figure 6.2(a)), while that for the MECV problem has only one path of length 3 (compare Figure 5.2(b)).

Another drawback of Algorithm 22 is that, for adjacent vertices the decision of adding a vertex to the set of potential candidates depends on the costs of both elimination orders. Thus we have to compute both eliminations. For example in Example 6.2 for the decision of not pursuing the elimination sequences starting with $[v_1, v_2]$ we computed the costs for both elimination orders $[v_2, v_3]$ and $[v_3, v_2]$ on $G - v_1$. So in this case we have implicitly computed the elimination sequence $[v_1, v_2, v_3]$. Meaning that on our toy example we do not profit too much from exploiting commutativity of adjacent vertices. On graphs with more intermediate nodes the search tree is deeper and therefore the overhead of computing additional eliminations is negligible compared to the search space reduction achieved through the application of Algorithm 22.

6.2.2 Exploiting commutativity of edge eliminations

In this subsection we discuss how the commutativity of edge elimination can be used to reduce the search space of the branch and bound algorithm. As we already mentioned in Section 3.2.4, the resulting l-DAG can change if we change the order of edge eliminations, making it more complicated to exploit the commutativity of the edge elimination compared to vertex eliminations.

To exploit commutativity of edge eliminations within the branch and bound algorithm for optimal edge elimination problem we use the same approach as in the previous subsection. We modify the specialization of the REMOVEPERMUTABLEELIMINATIONS routine for the MEC problem to check not only the prerequisites of the Lemma 3.10, Lemma 3.11 and Lemma 3.12 ensuring that the resulting l-DAG does not change. We also ensure that the corresponding elimination costs are equal. Therefore we exploit Lemma 3.15, Lemma 3.16 and Lemma 3.17. The resulting routine is presented in Algorithm 23.

Algorithm 23 Specialization of REMOVEPERMUTABLEELIMINATIONS for EE problem

```

1: function REMOVEPERMUTABLEELIMINATIONS( $G_p, e1, M$ )
2:   for all  $e \in M$  do
3:      $H \leftarrow H \cup \{e\}$ 
4:     if Lemma 3.10( $G_p, e1, e$ )  $\vee$  Lemma 3.11( $G_p, e1, e$ )  $\vee$  Lemma 3.12( $G_p, e1, e$ ) then
5:       if Lemma 3.15( $G_p, e1, e$ )  $\vee$  Lemma 3.16( $G_p, e1, e$ )  $\vee$  Lemma 3.17( $G_p, e1, e$ ) then
6:         if  $e1 > e$  then
7:            $H \leftarrow H \setminus \{e\}$ 
8:         end if
9:       end if
10:    end if
11:  end for
12:   $M \leftarrow H$ 
13: end function

```

The check for equality of the costs is performed in line 5, by testing the applicability of Lemma 3.15, Lemma 3.16 or Lemma 3.17. Similar to the previous section, if the costs are the same we ensure that the eliminations are performed in increasing order. Otherwise we remove this elimination from the lists of candidates. Let us discuss this routine on the following example.

Example 6.3. We take the results from Example 5.2 as basis for this example. Consider l-DAG G as on Figure 5.3(a). Application of the branch and bound algorithm to solve the EE problem on G , without exploitation of commutativity of edge elimination, yields the same search tree as for solving the MEC problem, shown on Figure 5.4(a). The node labels denote the corresponding edge eliminations. Each path from the root to one of the leaves represents a complete edge elimination sequence.

We have 22 different complete edge elimination sequences to compare. Let us now see how the application of Algorithm 23 influences this search space. Obviously by application of Algorithm 23 we can remove only a subset of the nodes removed by Algorithm 17, as we perform an additional check. So let us see whether we get the same reduced search tree as shown on Figure 5.4(b). Thus we examine the excluded paths in this search tree and check whether we can also exclude this cases for the EE problem.

On $G - [(v_1, v_2)^+, (x_1, v_2)^+]$ (Figure 5.3(c)) we do not explore the elimination of $(v_1, y_1)^-$ as the order of eliminating $(x_1, v_2)^+$ and $(v_1, y_1)^-$ on $G - (v_1, v_2)^+$ (compare Figure 5.3(b)) does not change the resulting graph according to Lemma 3.12. According to Lemma 3.17 the order of these edge eliminations does not change the resulting costs too. On $G - [(v_1, v_2)^+, (x_1, v_1)^+]$ (Figure 5.3(d)) the only available edge eliminations are $(x_1, v_2)^+$ and $(v_2, y_1)^-$. According to Lemma 3.10 the order of eliminating $(x_1, v_1)^+$ and $(x_1, v_2)^+$ on $G - (v_1, v_2)^+$ (Figure 5.3(b)) does not change the resulting l-DAG. The costs also remain unchanged according to Lemma 3.15. Similarly according to Lemma 3.12 and Lemma 3.17

the order of eliminating $(x_1, v_1)^+$ and $(v_2, y_1)^-$ on $G - (v_1, v_2)^+$ (Figure 5.3(b)) also does not change the resulting l-DAG and the corresponding costs. Thus we do not have to explore any eliminations on $G - [(v_1, v_2)^+, (x_1, v_1)^+]$ as these cases were already covered in other branches. Hence we can exclude the same nodes as for the MEC problem. The reduced search space contains 20 different edge elimination sequences.

As we can see, we could reduce the amount of checked elimination sequences only by two. So the advantage of exploiting commutativity when solving optimal edge elimination problem is not as big as when you exploit commutativity for the optimal vertex elimination problem. Still for bigger graphs the gain of using this techniques is more significant. The reason for this is the fact that on bigger graphs less edges are incident. On our toy example almost all edges are incident and therefore the exploitation of commutativity can be applied in very rare cases.

Of course we can apply the same trick as in Algorithm 22 to avoid checking elimination sequences where the residual graph is the same but we have different costs as shown in Algorithm 24. If the order

Algorithm 24 Specialization of REMOVEPERMUTABLEELIMINATIONS for EE problem with handling of costs

```

1: function REMOVEPERMUTABLEELIMINATIONS( $G_p, e1, M$ )
2:   for all  $e \in M$  do
3:      $H \leftarrow H \cup \{e\}$ 
4:     if Lemma 3.10( $G_p, e1, e$ )  $\vee$  Lemma 3.11( $G_p, e1, e$ )  $\vee$  Lemma 3.12( $G_p, e1, e$ ) then
5:       if Lemma 3.15( $G_p, e1, e$ )  $\vee$  Lemma 3.16( $G_p, e1, e$ )  $\vee$  Lemma 3.17( $G_p, e1, e$ ) then
6:         if  $e1 > e$  then
7:            $H \leftarrow H \setminus \{e\}$ 
8:         end if
9:         else if  $\sigma_{G_p}([e1, e]) > \sigma_{G_p}([e, e1])$  then
10:           $H \leftarrow H \setminus \{e\}$ 
11:         else if  $\sigma_{G_p}([e1, e]) = \sigma_{G_p}([e, e1]) \wedge (e1 > e)$  then
12:           $H \leftarrow H \setminus \{e\}$  // avoid double checking if the costs are equal
13:         end if
14:       end if
15:     end for
16:      $M \leftarrow H$ 
17: end function

```

of eliminating the edges e and $e1$ does not change the resulting l-DAG but the Lemma 3.15, Lemma 3.16 or Lemma 3.17 does not hold, we ensure that the actual elimination sequence $[e1, e]$ yields smaller costs, otherwise we remove e from the list of the set of eliminations (compare line 9). In line 11 we ensure that the eliminations are performed in increasing order in case that the costs of both elimination orders are the same. This way we avoid double checking in such a case.

An l-DAG where the condition in line 5 of Algorithm 24 is violated, is too big to discuss its full search tree. Therefore in the next example we will consider only a small part of it.

Example 6.4. Consider l-DAG G is a lion graph (Figure 4.21(a)). In Example 3.10 we saw that both elimination orders $[(x_1, v_1)^+, (v_2, y_1)^-]$ and $[(v_2, y_1)^-, (x_1, v_1)^+]$ yield the same l-DAG when applied to a lion graph. Costs of them are different however

$$\sigma_G([(x_1, v_1)^+, (v_2, y_1)^-]) = 4 \quad , \quad \sigma_G([(v_2, y_1)^-, (x_1, v_1)^+]) = 3 \quad .$$

Hence with Algorithm 23 we need to explore elimination sequences starting with $[(x_1, v_1)^+, (v_2, y_1)^-]$, while with Algorithm 24 we do not do this.

We will not present an example demonstrating the influence of Algorithm 24 on the search space of the branch and bound algorithm. The effectiveness of this algorithm is very similar to that of Algorithm 22 and discussing it on example would not add much to the overall comprehension.

6.3 Lower bounds

In this section we discuss various algorithms to determine the lower bounds for optimal vertex and edge elimination problems on general I-DAGs. As we already saw in Chapter 5 good lower bounds are crucial for efficient application of the branch and bound algorithm. They can also help us to assess our current solution. The main contribution of the section is presented in Section 6.3.2, where we present algorithm for determining lower bound for arbitrary I-DAGs, based on reduction to absorption-free subgraphs.

This section is divided into two subsections. In the first subsection we deal with lower bounds that do not take into account the structure of the I-DAG, meaning that these lower bounds do not consider the fact that the number of predecessors and successors of a vertex can change during a vertex or edge elimination sequence. The computation of these bounds require only static knowledge about the I-DAG, number of intermediate vertices and eliminatable edges. Therefore these bounds are easy (fast) to compute but in typical applications they are not very sharp. We refer to these bounds as simple (lower) bounds. The usage of these lower bounds within the branch and bound algorithm was investigated in [Pei09] under my supervision.

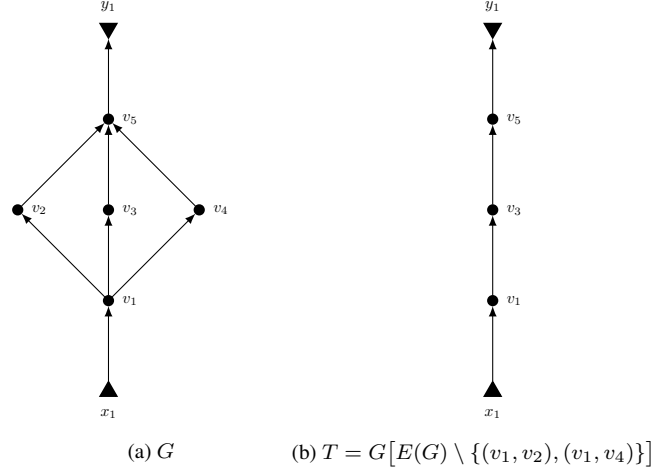
In the second subsection we discuss lower bounds that exploit the structure of the I-DAG to improve the results. The basic idea behind these bounds is to compute the lower bounds on the absorption-free subgraph of the actual graph. The bounds on absorption-free subgraphs are computed using the results from Section 4.3.3. The lower bounds computed this way are in most cases better than that presented in the first subsection but it takes more time to compute these bounds. The comparison of these lower bounds with simple bounds was discussed in [EH12]. Here we also compare these bounds with the lower bounds used in [CHMU12].

6.3.1 Simple bounds

This subsection deals with lower bounds that do not exploit the structure of the I-DAG. The knowledge about the number for (intermediate) vertices and number of eliminatable edges is used to compute these bounds. The bounds can be computed very fast but the quality of the bounds is poor for bigger I-DAGs, only for smaller graphs they produce valuable results. We refer to the (lower) bounds discussed here as simple bounds. These bounds were used in [Pei09] to speed up the branch and bound algorithm for solving the optimal edge and vertex elimination problem. The biggest advantage of these lower bounds is the fact that they can be computed almost instantly. Therefore it still makes sense to use these bounds within a branch and bound algorithm before much time is spent computing more sophisticated lower bounds as presented in Section 6.3.2.

In Section 4.3.3 we presented a lower bound based on the sum of Markowitz degrees of intermediate vertices. We could achieve good results applying this lower bound to absorption-free I-DAGs. Unfortunately we cannot use the sum of Markowitz degrees of intermediate edges as a lower bound for the costs of a complete edge [vertex] elimination sequence on a not absorption-free I-DAGs. This is because the number of predecessors and successors of vertex can decrease during an elimination sequence. This is possible due to absorption of parallel edges which never happens on absorption-free graphs.

Example 6.5. Consider I-DAG as on Figure 6.3(a). The sum of Markowitz-degree is 9 on this I-DAG. While $\sigma([v_2, v_3, v_4, v_1, v_5]) = 1 + 1 + 1 + 1 + 1 = 5$. Hence we cannot use sum of Markowitz-degree as the lower bound on general I-DAGs.

Figure 6.3: L-DAG G from Example 6.5 and its absorption-free subgraph used in Example 6.10.

To ensure the correctness of the lower bound from Lemma 4.24 on general l-DAGs we must replace the Markowitz degree with *minimal* Markowitz degree (on absorption-free l-DAGs the Markowitz degree is equal to minimal Markowitz degree). That is the smallest Markowitz degree that a vertex can have during any vertex or edge elimination sequence (formal definition is presented in Definition 6.1). This idea was described [NH08] when developing an optimal vertex elimination algorithm in single-expression-use graphs. The minimal Markowitz degree is defined as follows.

Definition 6.1. Consider an l-DAG G and $v \in Z(G)$. The minimal Markowitz degree of v in l-DAG G is defined as

$$\text{mark}_G(v) = |\underline{P}(v)| \cdot |\underline{S}(v)| \quad ,$$

where $\underline{P}(v)$ is minimal $X(G)$ - v vertex cut in G and $\underline{S}(v)$ is minimal v - $Y(G)$ vertex cut in G . Minimal $X(G)$ - v vertex cut in G is minimal vertex set $C \subseteq X(G) \cup Z(G) \setminus v$ such that every path from $i \in X(G)$ to $v \in Z(G)$ includes a vertex from C . Similarly v - $Y(G)$ vertex cut in G is minimal vertex set $C \subseteq Y(G) \cup Z(G) \setminus v$ such that every path from v to $i \in Y(G)$ includes a vertex from C .

Let us apply this idea to Example 6.5.

Example 6.6. Consider l-DAG as on Figure 6.3(a). The Markowitz degree of vertices v_2 , v_3 and v_4 is 1 and hence the minimal Markowitz degree of these vertices is also one. The minimal Markowitz degree of v_1 and v_5 is different from their Markowitz degrees as v_1 is connected to only one output and v_5 is connected to only one input and hence

$$|\underline{S}(v_1)| = 1 \quad \text{and} \quad |\underline{P}(v_5)| = 1$$

yielding that the minimal Markowitz degree of these vertices is also one. Hence

$$\sum_{v \in Z(G)} \text{mark}_G(v) = 5 \quad .$$

The algorithm to find the X - v and v - Y sets, needed to compute the minimal Markowitz degree, has at least quadratic complexity (see [NH08] and [CHMU12] for more details). So the computation of these

bounds requires significant amount of time. Hence we will not pursue this idea for lower bound in this subsection as here we want to discuss only those bounds that can be computed almost instantly. We will come back to this bound in Section 6.3.2 where we will compare its results with the lower bounds developed in this thesis.

Instead we simply underestimate the minimal Markowitz degree by 1. The resulting lower bound is proved in the following lemma.

Lemma 6.1. *The following holds for any l-DAG G .*

$$\text{VE}(G) \geq \text{EE}(G) \geq |Z(G)| \quad .$$

Proof. The inequality between VE and EE follows from Lemma 2.1. Let us now prove the remaining second part of the inequality. During the complete edge elimination sequence we need to remove all vertices. The costs of an edge elimination that causes a removal of a vertex is at least one. From the definition of edge elimination follows that we can remove at most one vertex by an edge elimination, hence the claim holds. $\square$

So far we tried to underestimate the costs for removing a vertex from the l-DAG and used this result as a lower bound to the costs of a complete edge or vertex eliminations sequence. We can apply similar idea to the eliminatable edges and underestimate the costs of removing an edge from the l-DAG. In contrast to vertices we can remove an edge either explicitly by front [back] elimination or implicitly if this edge is removed as a result of the removal of one of its incident vertices. Explicit removal of an edge yields at least one fma and we can remove only as many edges implicitly as we have intermediate vertices. This leads to the lower bound presented in the next lemma.

Lemma 6.2. *The following holds for any l-DAG G .*

$$\text{VE}(G) \geq \text{EE}(G) \geq |E^*(G)| - |Z(G)| \quad .$$

Proof. The inequality of VE and EE follows from Lemma 2.1. Let us now prove the remaining second part of the inequality. We prove this by induction of the length n of a complete optimal edge elimination sequence.

$n = 1$: Obvious

$n \mapsto n + 1$: Let (i, j) be the first edge that is eliminated in the complete optimal elimination sequence

Case 1. The edge (i, j) is front eliminated. We set $G' = G - (i, j)^+$. From the fact that $(i, j)^+$ is a first elimination in a complete optimal edge elimination sequence and the induction hypothesis follows

$$\text{EE}(G) = |S(j)| + \text{EE}(G') \geq |S(j)| + |E^*(G')| - |Z(G')| \quad (6.1)$$

Now we have to distinguish between the two cases when $(i, j)^+$ is a pure and a when $(i, j)^+$ is non pure edge elimination.

Case 1.1. $(i, j)^+$ is a pure edge elimination. Hence according to the definition of pure edge elimination $Z(G') = Z(G)$ and $|E^*(G')| \geq |E^*(G)| - 1$. Using these results in Equation (6.1) yields

$$\text{EE}(G) \geq \underbrace{|S(j)|}_{\geq 1} + |E^*(G)| - 1 - |Z(G)| \geq |E^*(G)| - |Z(G)| \quad .$$

Finishing the proof for this case.

Case 1.2. $(i, j)^+$ is a non pure edge elimination. Hence according to the definition of non pure edge elimination $Z(G') = Z(G) \setminus \{j\}$ and $|E^*(G')| \geq |E^*(G)| - (|S(j)| + 1)$. Using these results in Equation (6.1) yields

$$EE(G) \geq |S(j)| + |E^*(G)| - (|S(j)| + 1) - (|Z(G)| - 1) \geq |E^*(G)| - |Z(G)| \quad ,$$

finishing the proof for this case.

Case 2. The edge (i, j) is back eliminated. Symmetrical to **Case 1**.

□

We cannot say that one of the two simple lower bounds we presented is superior compared to the other. It depends on the given l-DAG. If $|Z(G)| > 2|E^*(G)|$ then the bound from Lemma 6.1 provides better result. If $|Z(G)| < 2|E^*(G)|$ the lower bound from Lemma 6.2 should be used. Therefore we should combine the results from Lemma 6.1 and Lemma 6.2 to one lower bound as

$$VE(G) \geq EE(G) \geq \max\{|Z(G)|, |E^*(G)| - |Z(G)|\} \quad .$$

We define this result as our `lower_bound_1` presented in Algorithm 25. In the next example we apply

Algorithm 25 `lower_bound_1`

Require: l-DAG G

Ensure: $lb \leq EE(G)$

$lb = \max\{|Z(G)|, |E^*(G)| - |Z(G)|\}$

these lower bounds to the graph from Figure 6.3 as well as to lion and bat graphs (compare Figure 4.21(a) and Figure 6.4(a)).

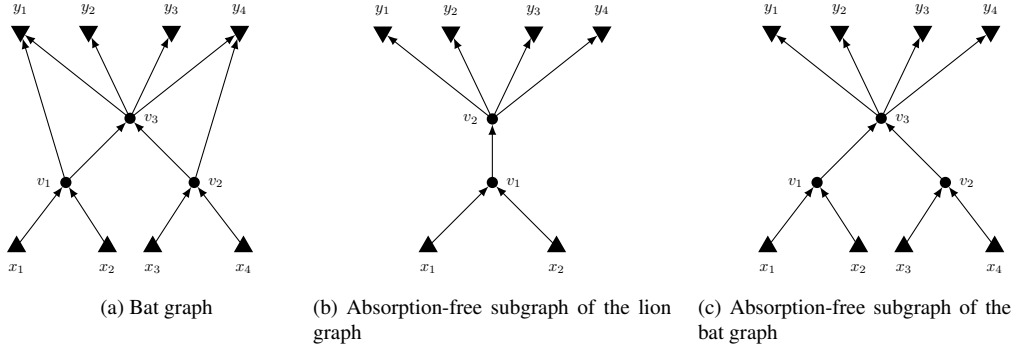


Figure 6.4: Illustration of Example 6.7 and Example 6.8.

Example 6.7. The l-DAG from Figure 6.3 has 5 intermediate vertices and 8 eliminatable edges and hence according to Algorithm 25 we get a lower bound

$$\max\{|Z(G)|, |E^*(G)| - |Z(G)|\} = \max\{5, 3\} = 5 \quad ,$$

which luckily corresponds to the costs of the optimal solution. On bat and lion graphs we do not get such good results.

The lion graph (compare Figure 4.21(a)) contains 2 intermediate vertices and 8 eliminatable edges, hence we get a lower bound of 6. Which is far away from the real solution that is 11.

Similarly, if we apply these techniques to the bat graph (compare Figure 6.4(a)), we get a lower bound of 9 (3 intermediate vertices and 12 eliminatable edges) which is not even half of the costs of the optimal edge elimination solution, 23.

As we can see, the bounds we have so far provide poor results. The reason for this bad performance is that we assume that no new edges are introduced during the elimination sequence. Even on absorption-free graphs we implicitly make this assumption. This of course does not match the reality. To improve the lower bounds we need techniques that can make use of the actual structure of the I-DAG and exploit it for better lower bounds. One possible idea to improve the bounds is to use the results from Theorem 3.2 and compute lower bounds on the absorption-free subgraphs of the actual I-DAG. This idea is discussed in the next subsection.

6.3.2 Lower bounds for edge and vertex elimination problems based on reduction to absorption-free subgraphs

In this subsection we develop more advanced lower bounds for the optimal vertex and edge elimination problem. The biggest advantage of these lower bounds over the ones shown in the previous subsection is that they can exploit the structure of the I-DAG. Specifically we do not assume that all edges are preserved. As first attempts of developing such bounds may be considered the result presented in [NH08]. Sum of minimal Markowitz degrees of intermediate vertices was used as a lower bound for the number of fma's performed during an optimal edge or vertex elimination sequence. We discussed this bound on absorption-free I-DAGs (note that on absorption-free graphs Markowitz degree is equal to minimal Markowitz degree).

The lower bounds developed in this thesis are based on the result from Theorem 3.2. It allows us to use the lower bounds computed on the subgraph of the actual graph as a lower bound for the original I-DAG. So we only have to decide what kind of subgraphs do we want to use for computing the lower bounds. As we already have good algorithms for lower bounds on absorption-free I-DAGs, it is self-evident to build absorption-free subgraphs of the given I-DAG and apply the algorithms from Section 4.1 to compute the lower bound for this subgraph.

This subsection is structured as follows. First we discuss the basic idea of computing the lower bounds based on reduction to absorption-free subgraphs on an example and formulate an algorithm (Algorithm 26). Then we apply this algorithm to several examples. Next we will see that the choice of absorption-free subgraph is crucial to the quality of the lower bounds. Afterwards we apply the algorithm to the I-DAGs discussed in [CHMU12] and compare the results with lower bound based on the sum of minimal Markowitz degrees.

The next example demonstrates the basic idea of computing lower bounds through reduction to an absorption-free subgraph.

Example 6.8. Consider the lion graph as on Figure 4.21(a). As we already know the optimal edge elimination on lion graph requires 11 fma's. We denote the lion graph without the edge (v_1, y_1) (compare Figure 6.4(b)) as G' . G' is obviously an absorption-free I-DAG. Moreover it is a subgraph of the lion graph. Hence $EE(G) \geq EE(G')$ according to Theorem 3.2. Applying the Algorithm 8 to G' we get

$$EE(G') = 2 + 2 \cdot 4 = 10 \quad ,$$

meaning that the optimal edge elimination on the lion graph needs at least 10 fma's. This result is much better than the one we saw in Example 6.7 where we applied the simple lower bounds.

Let us now try to improve the lower bounds for the bat graph (compare Figure 6.4(a)). First we construct an absorption-free subgraph of the bat graph. We can achieve this by removing the edges

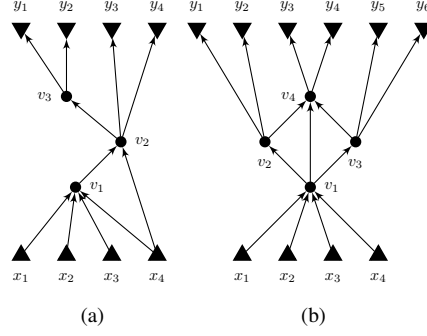


Figure 6.5: Illustration of Example 6.9.

(v_1, y_1) and (v_2, y_4) (compare Figure 6.4(c)). Applying the Algorithm 8 to this subgraph we get a lower bound of 20. Hence an optimal edge or vertex elimination on the bat graph performs at least 20 fma's. This result is only 3 fma's away from the costs of an optimal edge elimination sequence, which is 23 as proved in [Nau04]. It is also much better than the lower bound we get applying Algorithm 25 that was 9 fma's in Example 6.7.

In the Algorithm 26 we formulate the idea of computing lower bounds on the absorption-free subgraph of a given l-DAG. As you can see, Algorithm 26 is a very short algorithm. It first constructs an absorption-

Algorithm 26 Lower bound based on absorption-free subgraphs

Require: l-DAG G , $1b$

// Compute incremental lower bound

Ensure: $1b \leq EE(G)$

ALGORITHM 3(G, T)

ALGORITHM 11($T, 1b$)

free subgraph T of the original l-DAG G using Algorithm 3. Then the lower bound for the subgraph T is computed using the Algorithm 11. Of course we could use any of the algorithms developed in Section 4.3.3 to compute the lower bounds on absorption-free subgraph T . Here we simply chose the best one.

Let us apply this Algorithm 26 to some l-DAGs and see what results we get using this algorithm and compare them with the bounds presented in the previous section.

Example 6.9. Consider the l-DAG G as on Figure 6.5(a). In Example 6.23 we will see that $VE(G) = EE(G) = 22$ for this l-DAG. Let us now see how good the new lower bounds are. Applying the Algorithm 25 to G yields a lower bound of 8 as $|Z(G)| = 3$ and $|E^*(G)| = 11$. Let us now apply the Algorithm 26 to G . We assume that we remove the edge (x_4, v_1) to construct an absorption-free subgraph T of G . Now we have to apply the Algorithm 11 to absorption-free subgraph T . In Example 4.20 we saw that Algorithm 11 provides a lower bound of 21 for T . Hence according to Algorithm 26 $VE(G) \geq EE(G) \geq 21$ which is a very good lower bound.

Let us now look at another example. Consider l-DAG G as on Figure 6.5(b). In Example 6.23 we will prove that $VE(G) = EE(G) = 32$. Applying the Algorithm 25 to G yields a lower bound of 11 as $|Z(G)| = 4$ and $|E^*(G)| = 15$. Now we apply the Algorithm 26. We assume the edges (v_2, v_4) and (v_3, v_4) were removed to construct an absorption-free subgraph T (compare Figure 4.30(a)) of l-DAG G . The application of Algorithm 11 yields a lower bound of 30 fmas (compare Example 4.22). Hence

$VE(G) \geq EE(G) \geq 30$. Which is again a good lower bound as it is only 2 fmas away from the optimal solution.

The examples we discussed up till now may give you a feeling that the bounds based on absorption-free subgraphs are always superior to the bounds we have presented in the previous section. The following example shows that this is not always the case.

Example 6.10. Consider the l-DAG G as on Figure 6.3(a). The lower bound produced by Algorithm 25 is

$$\max \{|Z(G)|, |E^*(G)| - |Z(G)|\} = \max \{5, 8 - 5\} = 5 \quad .$$

Let us now apply the Algorithm 26. First we need an absorption-free subgraph of G . Therefore we need to destroy the two of the three parallel paths connecting v_1 and v_5 . We can do this by removing the edges (v_1, v_2) and (v_1, v_4) . Doing so we must also remove the vertices v_1 and v_4 from this l-DAG as they are no longer connected to the input x_1 (compare Figure 6.3(b)). No matter which edges are removed we end up with a graph which is isomorphic to this one. Obviously the optimal vertex (edge) elimination on this l-DAG requires 3 multiplications hence the lower bound produced by Algorithm 26 cannot exceed 3. Hence, the bounds presented in the previous section do a better job on this example.

Still the previous example is rather an exception. In typical applications the lower bounds based on reduction to absorption-free subgraph provide better results than the simple lower bounds. Nevertheless as already mentioned in the previous subsection, one should always consider using the simple lower bounds, as the computation of these bounds is cheap.

Choice of absorption-free subgraph is crucial. As you might have anticipated the quality of the bounds produced by the Algorithm 26 strongly depends on the choice of the absorption-free subgraph. The next example shows that the choice of the subgraph is often crucial to get good lower bounds. Therefore in the next example we will compute a lower bound for the bat graph and the two l-DAGs from Example 6.9 using not optimal subgraphs. We will then compare the results with the bounds computed previously in this subsection.

Example 6.11. First we discuss the bat graph. Let us assume that we removed the edges (v_3, y_1) and (v_3, y_4) to construct the absorption-free subgraph T (compare Figure 4.29(a)). Hence the application of Algorithm 11 to T yields a lower bound of 16 fma's as shown in Example 4.21. We get a significantly worse lower bound than the one we saw in Example 6.8 where we got a lower bound of 20. We can get even worse result if we construct the absorption-free l-DAG removing the edges (v_1, v_3) and (v_2, v_3) (compare Figure 4.29(b)) yielding a lower bound of 4 as discussed in Example 4.21.

Consider now the l-DAG G as on Figure 6.5(a). We assume that the absorption-free subgraph T is constructed by removing the edge (v_1, v_2) (compare Figure 4.28(d)). Applying the Algorithm 11 to T yields a lower bound of 5 as shown in Example 4.20. The lower bound for this l-DAG we get with this absorption-free l-DAG is far away from the result we got in Example 6.9. There we got a lower bound of 21 fma's.

Removing the edges (v_1, v_2) and (v_1, v_3) from the l-DAG G as on Figure 6.5(b) yields an absorption-free subgraph T that is isomorphic to that on Figure 6.4(b). Hence the application of Algorithm 11 yields a lower bound of 10. This is only a third of the lower bound we got in Example 6.9.

As we can see the choice of the absorption-free subgraph is crucial to get good lower bounds. Algorithm 3 allows us to construct the absorption-free subgraphs we need. We only have to adjust the weights of the edges in a proper way before running the algorithm. First attempts of developing such algorithms were made in the master thesis of Tarek El Habbaj [EH12] under my supervision.

In this thesis I will not discuss the results in any further detail for the following reasons. For the theoretical examples discussed in this thesis I picked the best absorption-free subgraphs to show the

advantages of the suggested approach for computing the lower bounds by reduction to subgraphs. In the long term we do not want to reduce the given l-DAG to an absorption-free subgraph. Rather, we want to stop removing the edges much earlier in order not to lose information about the graph, as discussed in the outlook. So it is not clear whether better algorithms for constructing absorption-free subgraphs can help us in the future.

Before finishing this subsection I would like to present the results of applying the Algorithm 26 to the graphs shown on Figure 6.6. These graphs were used for case study in [CHMU12]. The authors of this paper applied integer programming to solve the vertex elimination problem on these graphs. We will also compare the result of Algorithm 26 with the sum of minimal Markowitz degrees. The latter was used as a lower bound to speed up the branch and bound algorithm employed to solve the integer programming problem.

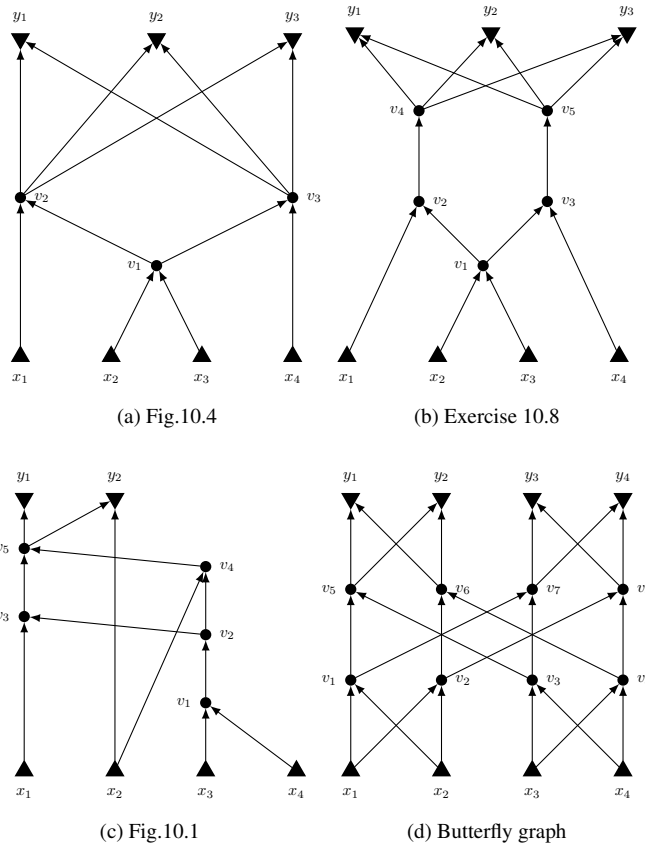


Figure 6.6: L-DAGs used in [CHMU12].

Example 6.12. Consider l-DAG G as on Figure 6.6(a). According to [CHMU12] $\text{VE}(G) = 18$. Let us now apply Algorithm 26 to G . We assume that the edge (v_1, v_3) was removed to build an absorption-free subgraph T of G (compare Figure 4.31(a)). Applying the Algorithm 11 to T yields a lower bound of 15 fma as shown in Example 4.23. We get a sharp lower bound which is only 3 fma 's away from the optimal solution. Still the sum of minimal Markowitz degrees provides a better lower bound of 16 fma 's according

to

$$\sum_{i=1}^3 \text{mark}_G(v_i) = 4 + 6 + 6 = 16 \quad .$$

Example 6.13. Consider l-DAG G as on Figure 6.6(b). According to [CHMU12] $\text{VE}(G) = 22$. Let us now apply Algorithm 26 to G . We assume that the edge (v_1, v_3) was removed to build an absorption-free subgraph T of G (compare Figure 4.31(b)). Applying the Algorithm 11 to T yield a lower bound of 17 fma's as shown in Example 4.24. The lower bound we get is not as sharp as in previous examples as it is 5 fma's away from the optimal solution. Still the sum of minimal Markowitz degrees cannot provide a better bound in this case as

$$\sum_{i=1}^5 \text{mark}_G(v_i) = 4 + 2 + 2 + 3 + 3 = 14 \quad .$$

Example 6.14. Consider l-DAG G as on Figure 6.6(c). According to [CHMU12] $\text{VE}(G) = 15$. Let us now apply Algorithm 26 to G . We assume that the edge (v_2, v_3) was removed to build an absorption-free subgraph T of G (compare Figure 4.31(c)). Applying the Algorithm 11 to T yields a lower bound of 14 fma's as shown in Example 4.25. We get a sharp lower bound which is only 1 fma away from the optimal solution. The sum of minimal Markowitz degrees cannot provide a better lower bound in this case as

$$\sum_{i=1}^5 \text{mark}_G(v_i) = 2 + 1 + 2 + 2 + 4 = 11 \quad .$$

Example 6.15. Consider l-DAG G as on Figure 6.6(d). We refer to G as butterfly graph. According to [CHMU12] $\text{VE}(G) = 48$. Let us now apply Algorithm 26 to G . We assume that the edges $(v_5, y_2), (v_6, y_1), (v_7, y_4), (v_8, y_3)$ were removed to build an absorption-free subgraph T of G (compare Figure 4.31(d)). Applying the Algorithm 11 to T yields a lower bound of 24 fma's as shown in Example 4.26. The bound is not sharp at all as it is only half of the number of fma's performed by an optimal vertex elimination. The sum of minimal Markowitz degrees provides a much better lower bound of 32 fma according to

$$\sum_{i=1}^8 \text{mark}_G(v_i) = 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 = 32 \quad .$$

As we can see, the approach presented in this section is not always better than the sum of minimal Markowitz degrees. It fails especially to provide a good lower bound for the butterfly graph. Still we have not used the full potential of the basic idea of the presented approach. We do not have to reduce the l-DAG to an absorption-free subgraph. We decided to do this because we could compute good lower bounds for absorption-free graphs. If we keep more edges in the graph we can potentially compute a better lower bound, as the more edges we remove the more information about the original graph gets lost. We just need a way of determining good lower bounds on non-absorption-free graphs. First ideas on such algorithms are discussed in the outlook in Section 7.2.2. We also do not use the information contained in the edges we have removed to build a subgraph. For example we could try and estimate the critical degree of the removed edges and improve our lower bound using this information. This idea is also discussed in Section 7.2.2. In the outlook we will apply these ideas to the previous four graphs (Figure 6.6). Exploiting these ideas allows us to compute a lower bound equal to the solution of the optimal vertex (edge) elimination problem for all four graphs and provides a better lower bound than the sum of minimal Markowitz degrees for all four examples.

6.4 Numerical results

In this section we present results of applying the techniques proved in this thesis for solving the EE or VE problems with branch and bound algorithm on several examples. The main objective of presenting the results in this section is to provide a better understanding of the usefulness of the techniques discussed in this thesis. While one can imagine that lower bounds can have a great impact on the performance of the branch and bound algorithm, the impact of exploiting the commutativity of edge and vertex elimination seems to be negligible looking at the examples. Especially the commutativity of edge elimination seems to be not worth considering, if we consider Example 6.3.

To generate results from this section we used the software originally developed as part of a bachelor thesis from Elmar [Pei09] and extended in the master thesis of Tarek El Habbaj [EH12]. It implements a branch and bound algorithm as discussed in Section 6.1 for solving VE and EE problems. It allows the choice of algorithms for lower bounds. The lower bounds can be computed as in Algorithm 25 or based on reduction to absorption-free subgraphs. The lower bounds on absorption-free subgraphs can be evaluated according to Algorithm 10 or Algorithm 9. The software cannot compute the most advanced lower bounds according to Algorithm 11. In addition to lower bounds we can also exploit the commutativity of edge- and vertex elimination (compare Algorithm 23 and Algorithm 21). In case of vertex elimination we can also handle adjacent vertices as in Algorithm 22.

Results of the tests are presented in two tables. Table 6.1 contains results for VE problem while Table 6.2 shows results for EE problem. The first column of the table contains the reference to the figure containing the examined l-DAG. This figure presents the l-DAG and the absorption-free subgraph used to compute the lower bounds (edges removed to build an absorption-free subgraph a marked with dashed lines). The second column shows which algorithms were used to obtain the solution. Each letter in this column indicates an algorithm

r – Algorithm 21 or Algorithm 23 depending on whether VE or EE problem is solved

a – Algorithm 22 only used when solving VE problem

m – Algorithm 9

c – Algorithm 10

The third column contains the number of fma's performed by the solution of the VE (EE) problem. The fourth column contain the overall number of elimination sequences checked by the branch and bound algorithm to compute the solution. The last column shows the time needed to compute the solution. Some of the tests took too long in this case the time column contains the time after which the test was stopped and the fourth column shows the percentage of the whole search space that was tested so far. The test were performed on an machine with Intel(R) Xeon(R) CPU E5-2630 and 128 GB RAM.

The lower bounds from Algorithm 25 are used in all tests even if we run tests using Algorithm 9 or Algorithm 10. Algorithm 9 is only used if Algorithm 25 fails to perform the cut. The lower bounds according to Algorithm 10 are only computed if the lower bounds from Algorithm 9 are not good enough to perform the cut. This way we ensure that the most expensive algorithms (in terms of computational time) are only used if necessary. This helps to reduce the overall run time of the branch and bound algorithm, and is the way the lower bounds should be used for real world problems.

6.4.1 Vertex elimination

In this subsection we present the result for solving the VE problem. With the exception of the butterfly graph (see Figure 6.6(d)) the number of intermediate vertices in all l-DAGs discussed in this thesis is simply too small. Hence the search space induced by the VE problem on these graphs is not big enough

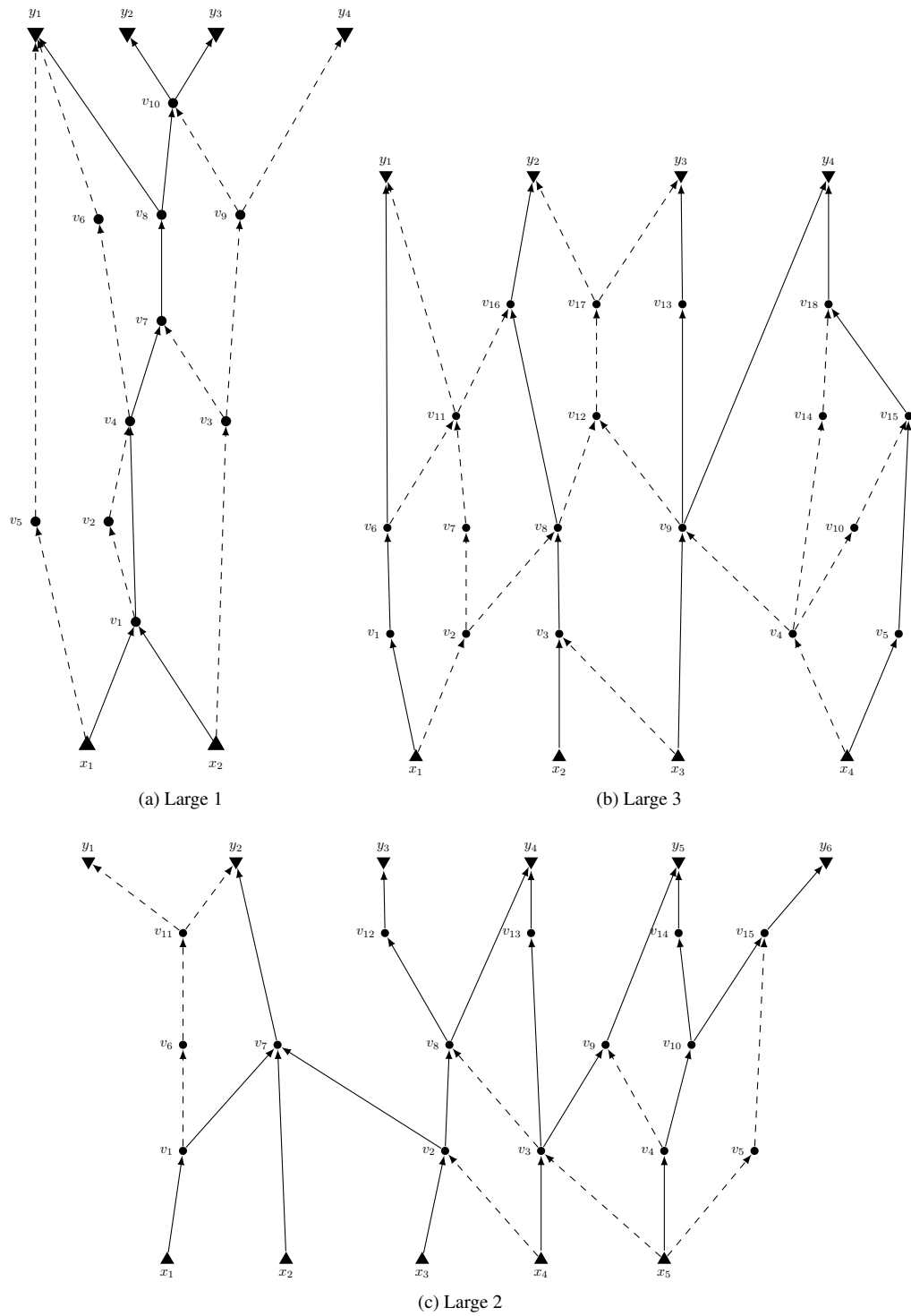


Figure 6.7: L-DAGs with corresponding absorption-free subgraphs used to compute lower bounds. Dashed edges indicate edges that were removed to build the absorption-free subgraph.

and does not show any interesting results. Therefore we will apply the software to the butterfly and three other graphs presented in Figure 6.7.

The results for solving the VE problem on all discussed l-DAGs are shown in Table 6.1. Let us first discuss the results for the butterfly graph. As we can see the lower bounds based reduction to absorption-free subgraph cannot reduce the number of checked elimination sequences, meaning that the lower bounds based on reduction to absorption-free subgraph are not better than the lower bounds from Algorithm 25. Computing the lower bounds based on reduction to absorption-free subgraph almost doubled (Algorithm 9) or tripled (Algorithm 10) the time needed to compute the solution. The reason why the lower bounds based on reduction to absorption-free subgraphs are not good enough is based on the fact that we have to remove too many edges to build an absorption-free subgraph and hence too much information is lost. A solution to this problem is shown in Example 7.6, where we compute a perfect lower bound. Exploitation of commutativity of vertex elimination can significantly reduce the number of checked elimination sequences. Exploiting the commutativity of not adjacent vertices, reduces the number of checked sequences by a factor of 16. Exploitation of the commutativity of adjacent vertices reduces the number of checked elimination sequences by another factor of 20. Overall we can reduce the number of checked elimination sequences by more than 300 times. As we can see we can compute the optimal solution for the butterfly graph in only 0.03 seconds. In [CHMU12] the same problem was solved by an integer programming approach. In best case the solution of this problem took more than 300 seconds.

l-DAG	modes	solution	num. checked sequences	time
Figure 6.8(d)	mc	48	40320	3.7 sec
	m	48	40320	2.7 sec
		48	40320	1.5 sec
	r	48	2468	0.11 sec
	rm	48	2468	0.2 sec
	rmc	48	2468	0.3 sec
	rmca	48	128	0.03 sec
Figure 6.7(a)		23	2386405	38.5 sec
	m	23	669164	20.3 sec
	mc	23	445878	16.3 sec
	r	23	10418	0.28 sec
	rm	23	10334	0.4 sec
	rmc	23	8586	0.5 sec
	rmca	23	2386	0.3 sec
Figure 6.7(b)	mc	–	<0.002%	stopped after 64h
	r	39	995634679	26735 sec
	rm	39	995634679	52339 sec
	rmc	39	995634679	65526 sec
	rmca	39	9483062	1546 sec
Figure 6.7(c)	mc	–	<0.36%	stopped after 27h
	r	35	20430196	602 sec
	rm	35	20418998	1268 sec
	rmc	35	20418998	1795 sec
	rmca	35	409039	92 sec

Table 6.1: Results for solving VE problem.

Next we will discuss the solution of the VE problem on the three l-DAGs presented on Figure 6.7.

The application of the given software to these graphs was extensively discussed in [Pei09] and [EH12]. Here we will only concentrate on some interesting facts. As we can see the commutativity of the vertex elimination is again a very useful tool. For all three l-DAGs it helps to reduce the search space and the run time similar to results we observed for the butterfly graph. For example for the l-DAG on Figure 6.7(b) it makes the differences between computing the result in less than 30 minutes (rmca) or waiting months or even years for the answer (mc). The lower bounds based on reduction to absorption-free subgraphs do not help to reduce the computational time. On the contrary the time needed to compute the solution is increased in all three cases. The number of checked elimination sequences could only be reduced for the l-DAG on Figure 6.7(a). On this l-DAG we can slightly reduce the number of checked elimination sequences by applying Algorithm 10 on top of Algorithm 9. Still the computation of the lower bounds according to Algorithm 10 is too expensive and we cannot decrease the overall run time of the program. For l-DAGs on Figure 6.7(c) and Figure 6.7(b) both Algorithm 9 and Algorithm 10 are beaten by Algorithm 25. Thus the application of both algorithms does not reduce the number of checked elimination sequences but significantly increases the run time.

Let us now investigate why lower bounds based on absorption-free graphs fail to beat the Algorithm 25 on these graphs. If we look at the absorption-free subgraphs of the l-DAGs, we will notice that many edges are removed to build the absorption-free graphs. Hence a lot of information about the original structure of the graphs is lost. So we need strategies of retaining this information to improve the lower bounds. Some ideas on this are presented in Section 7.2. Nevertheless lower bounds based on reduction to absorption-free subgraphs can provide good lower bounds as we will see in the next subsection.

6.4.2 Edge elimination

In this subsection we discuss the results for solving the EE problem on l-DAGs in Figure 6.8. All results are summarized in Table 6.2. As already mentioned the search space of the EE problem is much bigger than the search space of the VE problem. The difference is so big that although we could easily solve the VE problem on butterfly l-DAG, we failed to find the solution for the EE problem on this l-DAG. The program was stopped after 69 hours without making any significant progress. This result is even more disappointing as edge elimination and vertex elimination yield the same costs on the butterfly graph as proved in Example 7.6. It is more astonishing that with the ideas presented in Chapter 7 we could solve the EE problem on this l-DAG in linear time (compare Example 7.6). So we will not even try to solve the EE problem with the given software for any of the l-DAGs in Figure 6.7. For all tests discussed here we first solved the VE problem to get a good upper bound before solving the EE problem.

As we can see exploitation of commutativity of edge elimination (Algorithm 23) allows us to solve the problems much faster. For example for the l-DAG in Figure 6.8(h) it reduces the number of checked sequences and the run time by a factor of 1000 (mc vs rmc). For the l-DAG in Figure 6.8(b) we were not able to compute the solution in acceptable time without exploitation of commutativity of edge elimination. The bigger the l-DAG the more we gain from exploiting the commutativity of the edge eliminations.

For all tests discussed in this section we see that both lower bounds based on reduction to absorption-free subgraphs (Algorithm 9 and Algorithm 10) are better than the lower bounds computed by default. This fact is expressed in significant reduction of run time and number of the checked elimination sequences. We can also see that whenever we use the lower bounds based on Algorithm 10 in addition to lower bounds based on Algorithm 9 we achieve further reduction in the number of checked elimination sequences. The difference between Algorithm 9 and Algorithm 10 depends on the given l-DAG. On graphs with maximal path length 2 (Figure 6.8(e), Figure 6.8(f), Figure 6.8(a)) the lower bound based on Markowitz degree is good compared to results produced by Algorithm 10. On l-DAGs with longer paths like Figure 6.8(b), Figure 6.8(c), Figure 6.8(g) and Figure 6.8(h) Algorithm 10 significantly reduces the number of checked elimination sequences and the run time, especially for l-DAG in Figure 6.8(h), where the number of checked elimination sequences was reduced by a factor of 1000 through application of Al-

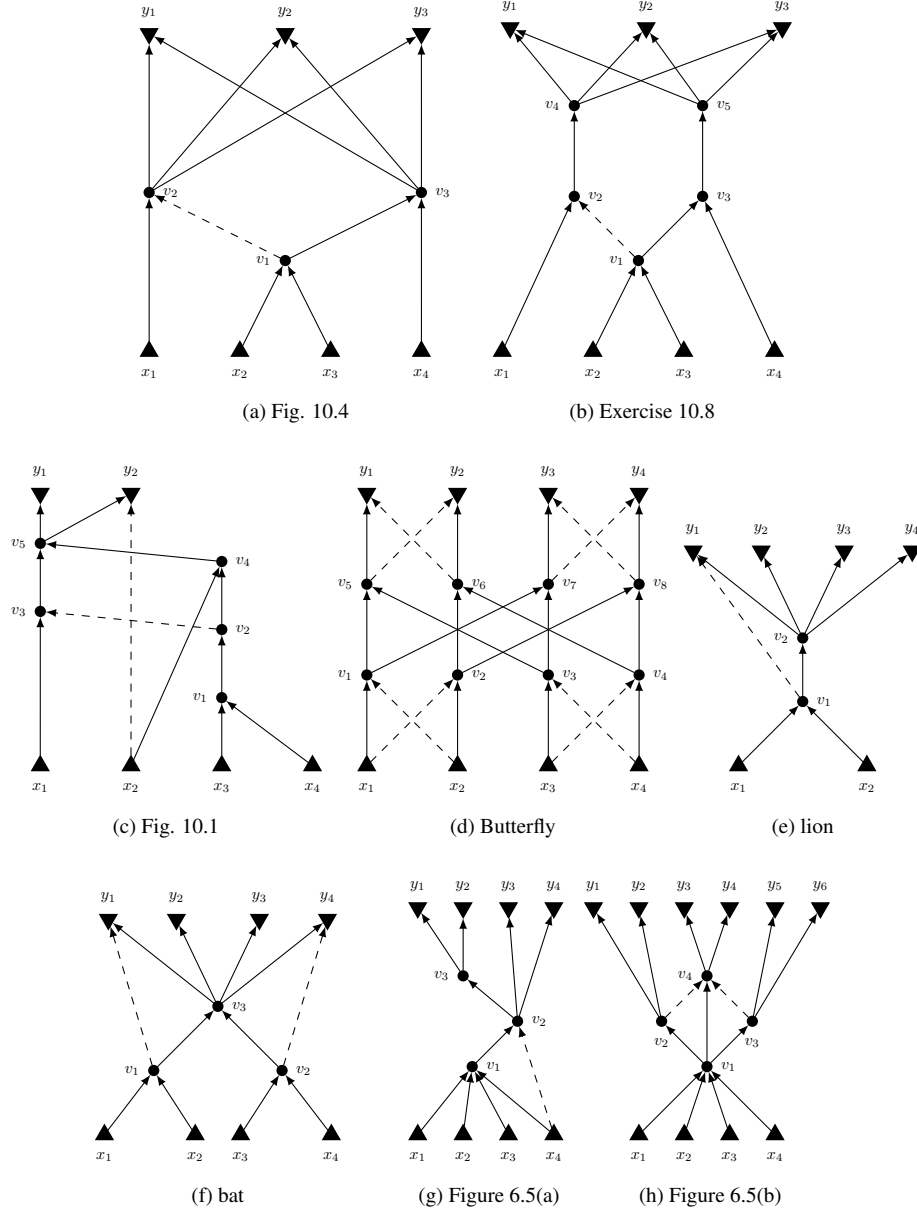


Figure 6.8: L-DAGs with corresponding absorption-free subgraphs used to compute lower bounds. Dashed edges indicate edges that were removed to build the absorption-free subgraph

gorithm 10. The reason why Algorithm 10 is better than Algorithm 9 on l-DAGs with longer paths is that Algorithm 10 can better map the fact that the number of edges can increase during elimination sequence. This can be done even better by the Algorithm 11. Unfortunately we do not have any implementation of this algorithm so far.

l-DAG	modes	solution	num. checked sequences	time
Figure 6.8(e)		11	15883	0.15 sec
	r	11	2594	0.02 sec
	m	11	748	0.01 sec
	rm	11	231	0.004 sec
	mc	11	65	0.002 sec
	rmc	11	57	0.002 sec
Figure 6.8(f)		–	<9%	stopped after 29h
	r	23	208701225	2538 sec
	mc	23	29610868	808 sec
	rm	23	5367293	146 sec
	rmc	23	2149352	68 sec
Figure 6.8(g)		22	1655635947	16612 sec
	r	22	116263115	1218 sec
	rm	22	1172296	31 sec
	mc	22	908	0.06 sec
	rmc	22	533	0.04 sec
Figure 6.8(d)	rmc	–	<0.00001	stopped after 64h
Figure 6.8(h)	r	32	<0.3%	stopped after 27h
	mc	32	321122341	9908 sec
	rm	32	269825048	7857 sec
	rmc	32	256307	9 sec
Figure 6.8(a)		18	38827700	411 sec
	m	18	11462801	264 sec
	mc	18	7515192	167 sec
	r	18	1007483	12 sec
	rm	18	379636	8 sec
	rmc	18	341887	9 sec
Figure 6.8(b)	mc	–	<1.17%	sopped after 36h
	r	22	2726798087	29466 sec
	rm	22	1598082512	30982 sec
	rmc	22	521729712	12712 sec
Figure 6.8(c)		15	396343562	2930 sec
	r	15	17275741	141 sec
	m	15	9262075	161 sec
	rm	15	760053	14 sec
	mc	15	163101	4 sec
	rmc	15	29403	1 sec

Table 6.2: Results for solving EE problem.

6.4.3 Conclusion

As we could see in our tests, exploitation of the commutativity of edge and vertex elimination reduces the search space of the VE and EE problems significantly. Thus these techniques should always be used when trying to solve these problems by the branch and bound algorithm. Sometimes the usage of these techniques makes the difference between being able to solve the problem or not.

The conclusion about the lower bounds based on reduction to absorption-free subgraphs is not that distinct. For l-DAGs discussed in Section 6.4.1 they failed to beat the simple lower bounds from Algorithm 25 and increased the time needed to solve the problems. For l-DAGs discussed in Section 6.4.2 the lower bounds based on reduction to absorption-free subgraphs could significantly decrease the number of checked elimination sequences and speed up the overall computation of the solution. We already discussed the reasons for this behavior. In the rest part of this thesis we will discuss ways to improve the behavior of the lower bounds based on absorption-free subgraphs. This will be achieved in two different ways. Indirectly by proving several optimality preserving eliminations (see Section 6.5). Optimality preserving eliminations allow us to reduce the size of the l-DAG (number of edges and vertices). Therefore the number of edges and vertices that needs to be removed to build an absorption-free subgraph is smaller and hence less information is lost during its construction. We will also show various direct ways to improve the lower bounds in Section 7.2.

6.5 Optimality preserving elimination

In this section we discuss optimality preserving eliminations on general l-DAGs for both edge and vertex elimination problems. Optimality preserving elimination is a very powerful tool that allows significant reduction of the search space. So far the only optimality preserving result for optimal edge and vertex elimination problems is, that elimination of vertices with Markowitz degree 1 preserves optimality. It was proved by Naumann in [Nau99]. In this section we will present two new optimality preserving eliminations (Lemma 6.5 and Lemma 6.6). Using these results we can solve the optimal edge elimination problem on several l-DAGs. For example for lion graph we can prove that the optimal edge elimination sequence involves 11 fmas (Example 6.23). Both results are special cases of Lemma 4.18 which we used to develop optimality preserving eliminations based on critical-degree of an edge on absorption-free graphs. To apply Lemma 4.18 we need to estimate the critical degree of an edge on arbitrary l-DAGs. Therefore we will prove a sufficient condition for critical edges on arbitrary l-DAGs (Lemma 6.3), as the lower bound proved in Theorem 4.5 is not applicable to not absorption-free graphs.

This section is structured as follows. First we discuss why the Lemma 4.16 and Lemma 4.17 are not applicable to general l-DAGs and present a counter example (Example 6.16). Next we discuss how to use the result from Lemma 4.18 for optimality preserving eliminations on not absorption-free graphs. Therefore we show that Theorem 4.5 is not applicable to not absorption-free graphs (Example 6.18). We then develop a sufficient condition for critical edges in Lemma 6.3 and combine this result with Lemma 4.18 to prove the optimality preserving eliminations in Lemma 6.5 and Lemma 6.6. At the end we use these results to obtain optimal solutions for the lion graph and the graphs from Figure 6.5(a) and Figure 6.5(b) (Example 6.23).

Unfortunately we cannot use the results for optimality preserving elimination of semi-bridges (see Lemma 4.16 and Lemma 4.17) on not absorption-free l-DAGs as they exploit the fact that $VE = EE$ on absorption-free l-DAGs. Even the weaker version of this result for bridges (e.g vertices v_2 and v_3 on Figure 6.9(a)) as shown in [MN12] cannot be applied to general l-DAGs, because it exploits the fact that the number of predecessors and successors of a vertex is not decreasing during a vertex elimination sequence. Consider a small example.

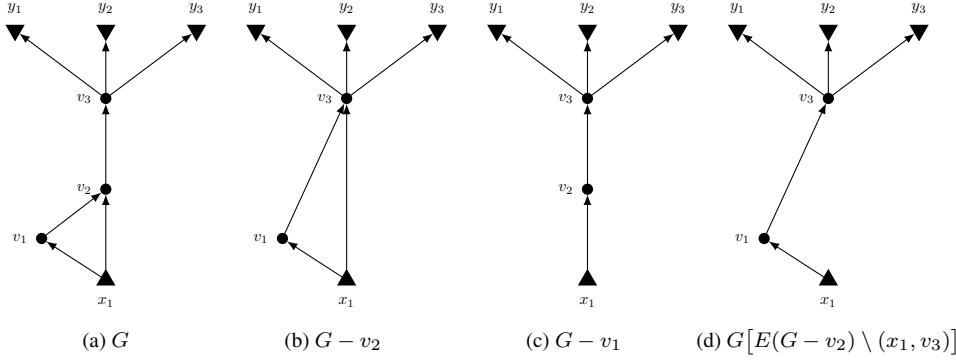


Figure 6.9: Illustration of Example 6.16 and Example 6.17.

Example 6.16. Consider l-DAG G as on Figure 6.9(a). The vertices v_2 and v_3 fulfill the requirements of the weaker version Lemma 4.16 as v_2 is the only predecessor of v_3 , v_3 is the only successor of v_2 and $|P_G(v_2)| < |S_G(v_3)|$. Still the elimination of v_2 does not preserve optimality neither for edge nor for vertex elimination problem. As all elimination sequences starting with elimination of v_2 need at least 6 fma's as $\sigma_G(v_2) = 2$ and $\text{EE}(G - v_2) = 4$ (compare Example 6.17). While the elimination sequence $[v_1, v_2, v_3]$ needs only 5 fmas

$$\sigma_G([v_1, v_2, v_3]) = 1 + 1 + 3 = 5 \quad .$$

The reason for this is the fact that after the elimination of v_1 from G the number of predecessors of v_2 decreases.

So far we can only use the result of Naumann in [Nau99] where he proved that elimination of vertices with Markowitz degree one preserves optimality in the context of the EE and VE problems. Although this result covers much smaller number of cases compared to that from Lemma 4.16 and Lemma 4.17 this result is good enough to help us to solve the optimal edge elimination problem for the l-DAG from Figure 6.9(a) as shown in the next example.

Example 6.17. Consider l-DAG G as on Figure 6.9(a). The Markowitz degree of vertex v_1 is 1. Hence the elimination of this vertex preserves optimality. The resulting l-DAG $G - v_1$ shown on Figure 6.9(c) is an absorption-free l-DAG. So we can apply our optimality preserving results for absorption-free l-DAGs. Hence according to Lemma 4.16 the elimination of v_2 preserves optimality on $G - v_1$ yielding that the vertex elimination sequence $[v_1, v_2, v_3]$ solves the EE and VE problems and

$$\text{EE}(G) = \text{VE}(G) = \sigma_G([v_1, v_2, v_3]) = 1 + 1 + 3 = 5.$$

Similarly the elimination of vertex v_1 on $G - v_2$ (compare Figure 6.9(b)) preserves optimality. Hence the vertex elimination sequence $[v_1, v_2]$ solves the EE and VE problems on $G - v_2$ and

$$\text{EE}(G - v_2) = \text{VE}(G - v_2) = \sigma_G([v_1, v_3]) = 1 + 3 = 4.$$

Not all results from Section 4.3.2 are useless on general l-DAGs. The result from Lemma 4.18 can be applied to arbitrary l-DAGs. To apply Lemma 4.18 we need to be able to determine (estimate) the critical degree of an edge on arbitrary l-DAGs. Unfortunately we cannot use the results from Theorem 4.5 as this theorem is based on Lemma 4.19 and Lemma 4.20. Both lemmas assume that there is no absorption. That is why if we use these results on non-absorption-free l-DAGs we may overestimate the critical degree of an edge as we can see in the following example.

Example 6.18. Consider l-DAG G as on Figure 6.9(b). If we apply the result of Theorem 4.5 to the edge (x_1, v_3) we get

$$\Delta_G((x_1, v_3)) \geq |Y(v_3)| = 3 \quad .$$

In Example 6.17 we have proved that

$$\text{EE}(G) = \text{VE}(G) = 4 \quad .$$

We set $G' = G[E(G) \setminus (x_1, v_3)]$ (compare Figure 6.9(d)). G' is absorption-free l-DAG with maximal path length 3 hence applying Algorithm 8 yields

$$\text{EE}(G') = \text{VE}(G') = 4 \quad .$$

Hence 3 is not a lower bound for the critical degree of the edge (x_1, v_3) neither for edge nor for the vertex elimination as

$$3 \leq \Delta_G^e((x_1, v_3)) \leq \text{EE}(G) - \text{EE}(G') = 4 - 4 = 0$$

and

$$3 \leq \Delta_G^e((x_1, v_3)) \leq \text{VE}(G) - \text{VE}(G') = 4 - 4 = 0 \quad .$$

Moreover (x_1, v_3) is even a not critical edge.

So we need another result to estimate the critical degree of an edge on arbitrary l-DAGs. Absorption makes it more complicated to compute good lower bounds for critical edge degree on arbitrary l-DAGs. The critical degree of edge can be even zero as that happened to the edge (x_1, v_3) in Example 6.18. While on absorption-free l-DAGs the critical degree of every edge is at least one.

Let us try to find a criterion to identify critical edges on arbitrary l-DAGs. As we saw, the edge (x_1, v_3) in Example 6.18 is not critical. The reason for this is that this edge is absorbed during an optimal elimination sequence and therefore it does not contribute to the overall elimination costs. The next example supports this observation.

Example 6.19. Consider l-DAG G as on Figure 6.10(a). We will show that the edge (v_1, y_1) is not critical for both optimal edge and vertex elimination problems as it is absorbed by $[v_2, v_1]$ which is the optimal elimination sequence for G with $\sigma_G([v_2, v_1]) = 12$. Next we will prove that $[v_2, v_1]$ is an optimal vertex elimination sequence for G' (Figure 6.10(b)). G' is an absorption-free subgraph of G which was produced by removing the edge (v_1, y_1) . According to Algorithm 8 $[v_2, v_1]$ is an optimal vertex elimination sequence on G' with $\sigma_{G'}([v_2, v_1]) = 12$. G' is an absorption-free l-DAG and therefore $\text{VE}(G') = \text{EE}(G')$ and hence $\text{VE}(G) = \text{EE}(G)$ as

$$12 = \text{VE}(G) \geq \text{EE}(G) \geq \text{VE}(G') = 12 \quad .$$

This implies that (v_1, y_1) is a not critical edge. Again we see that a not critical edge is absorbed during an optimal edge (vertex) elimination sequence.

Still an edge can be critical even if it is absorbed during an optimal elimination sequence as the following example shows.

Example 6.20. Let us assume that G is a lion graph (compare Figure 4.21(a)). We already know that $\text{EE}(G) = 11$ and $[(v_2, y_1)^-, v_1, v_2]$ is a corresponding optimal elimination sequence. The edge (v_1, y_1) is critical although it is absorbed during this optimal elimination sequence as

$$\Delta_G^e((v_1, y_1)) = \text{EE}(G) - \text{EE}(G') = 11 - 10 \quad ,$$

where G' is the lion graph without (v_1, y_1) (compare Figure 6.4(b)).

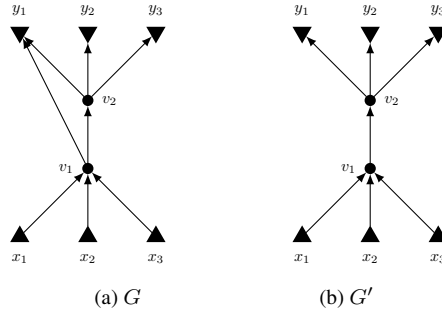


Figure 6.10: Illustration of Example 6.19.

So as we can see, even the knowledge about the optimal elimination sequence not necessarily helps us to decide whether an edge is critical or not. In the following we will develop an easy-to-check criterion for critical edges.

Obviously an edge contributes to the costs (increases the costs) of a given elimination sequence if at least one multiplication with the label of this edge is performed before the edge is absorbed. We can also explain this using bi-coloring. We color an edge with 1 and all other edges with 0. If a given elimination sequence produces at least one 1-colored edge then the 1-colored edge contributes at least one multiplication to this elimination sequence. This idea allows us to formulate the following sufficient condition to decide whether a given edge is critical or not.

Lemma 6.3. *Consider an l-DAG G and $(i, j) \in E^*(G)$. If there exists an optimal edge (vertex) elimination sequence not absorbing (i, j) then (i, j) is critical for edge (vertex) elimination.*

Proof. First we prove the claim for edge elimination. Therefore we investigate the cost of an optimal edge elimination sequence where (i, j) is not absorbed, by applying it to G and $G' = G[E(G) \setminus \{(i, j)\}]$. It is sufficient to show that at least one of the eliminations in this sequence performs at least one additional fma when applied to G , as in this case

$$EE(G') \leq EE(G) - 1 \quad ,$$

implying that (i, j) is critical. Due to the assumption that (i, j) is not absorbed when the elimination sequence is applied to G , there are only three different ways left to eliminate (i, j) from G :

1. elimination of (i, j) explicitly,
2. elimination of j by back elimination of all out-edges,
3. elimination of i by front elimination of all in-edges.

In the first case the elimination of (i, j) will be done only on G , and therefore the elimination sequence will cost at least one fma more when applied to G . Consider the second case. The back elimination of any out-edge of j in G will cause the multiplication with $c_{i,j}$. The claim holds due to the fact that this multiplication is not performed on G' . The third case is similar to the second.

The claim for vertex elimination follows immediately from the fact that vertex elimination is defined as a sequence of edge eliminations. □

Now we have a sufficient condition that allows us to decide whether an edge is critical or not. At first glance it is not quite clear how this result can be used in praxis, as we need to check whether any of the optimal elimination sequences does not absorb the edge. In particular this requires the knowledge about the optimal solution first, making this result not really useful. Still for some edges we know that they cannot be absorbed by any elimination sequence because there exists no alternative path connecting the source of the edge with its sink. The following Lemma formulates this idea.

Lemma 6.4. *Consider an l-DAG G and $(i, j) \in E^*(G)$. If G contains only one directed path from i to j then (i, j) is a critical edge for edge (vertex) elimination.*

Proof. As there exists only one directed path from i to j , the edge (i, j) cannot be absorbed during any edge (vertex) elimination sequence. Hence according to Lemma 6.3 (i, j) is a critical edge. $\square$

Let us discuss this result in the following example.

Example 6.21. *Consider G is a lion graph (compare Figure 4.21(a)). The edge (v_2, y_1) is the only connection of v_2 with y_1 . Hence according to Lemma 6.4 (v_2, y_1) is critical.*

We can use this result together with Lemma 4.18 to prove optimality persevering eliminations on arbitrary l-DAGs.

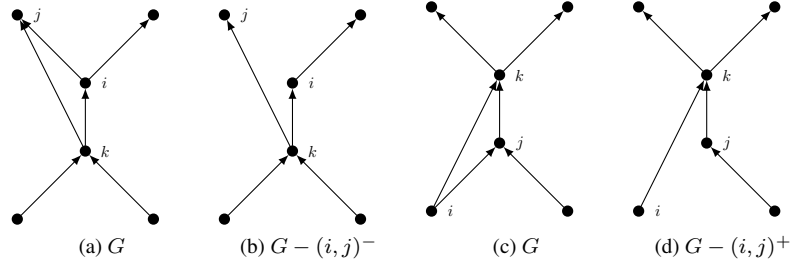


Figure 6.11: Illustration of the assumptions in Lemma 6.5 ((a) and (b)) and Lemma 6.6 ((c) and (d))

Lemma 6.5. *Consider an l-DAG G and $(i, j) \in E(G)$, where $(k, j) \in E(G)$, $P_G(i) = \{k\}$, and there exists only a single path from i to j . Back elimination of (i, j) preserves optimality (compare Figure 6.11(a) and Figure 6.11(b)).*

Proof. From Lemma 6.4 follows that (i, j) is critical. The back elimination of (i, j) transforms G to $G' = G[E(G) \setminus \{(i, j)\}]$ at the cost of one fma. From Definition 4.5 follows that

$$EE(G) \geq EE(G') + 1 = EE(G - (i, j)^-) + 1 \quad ,$$

which completes the proof. $\square$

Lemma 6.6. *Consider an l-DAG G and $(i, j) \in E(G)$, where $(i, k) \in E(G)$, $S_G(j) = \{k\}$, and there exists only one path from i to j . Front elimination of (i, j) preserves optimality (compare Figure 6.11(c) and Figure 6.11(d)).*

Proof. The proof is similar to that of Lemma 6.5 due to symmetry. $\square$

In the following example we apply the results from Lemma 6.5 and Lemma 6.6 to the three l-DAGs we used in examples in Section 6.3.

Example 6.22. Consider G is a lion graph (compare Figure 4.21(a)). The edge (v_2, y_1) fulfills the requirements from Lemma 6.5. Hence back elimination of (v_2, y_1) preserves optimality. The resulting l-DAG is presented on Figure 2.2(b).

Consider l-DAG G as on Figure 4.21(c). The edge (v_2, y_4) fulfills the requirements from Lemma 6.5. Hence back elimination of this edge preserves optimality for the EE problem. As the resulting l-DAG is a lion graph we can apply the same strategy as above.

Consider l-DAG G as on Figure 6.5(a). The edge (x_4, v_1) fulfills the requirements from Lemma 6.6. Hence front elimination of this edge preserves optimality. The resulting l-DAG is presented on Figure 4.28(b).

Consider l-DAG G as on Figure 6.5(b). Both edges (v_2, v_4) and (v_2, v_3) fulfill the requirements from Lemma 6.5. Hence back elimination of (v_2, v_4) and (v_2, v_3) preserves optimality. The resulting l-DAG is presented on Figure 4.30(a).

We already know how to solve the VE (EE) problem on the resulting absorption-free graph remainders from Example 6.22. So the combination of the results of Lemma 6.5 and Lemma 6.6 with the results for optimality preserving eliminations on absorption-free l-DAG (Section 4.3.2) allows us to solve the EE problem for these examples.

Example 6.23. Consider a lion graph G as on Figure 4.21(a). As shown in Example 6.22 back elimination of (v_2, y_1) preserves optimality. In Example 6.8 we proved that $[v_1, v_2]$ solves the edge (vertex) elimination problem on the resulting l-DAG (compare Figure 2.2(b)). Hence the elimination sequence $[(v_2, y_1)^-, v_1, v_2]$ solves the optimal edge elimination problem on the lion graph and

$$\text{EE}(G) = \sigma_G([(v_2, y_1)^-, v_1, v_2]) = 11 \quad .$$

Consider an l-DAG G as on Figure 4.21(c). As shown in Example 6.22, back elimination of (v_2, y_4) preserves optimality. The resulting l-DAG is a lion graph. We already know that an optimal edge elimination sequence on lion graph requires 11 fmas and hence

$$\text{EE}(G) = \sigma_G((v_2, y_4)^-) + \text{EE}(G - (v_2, y_4)^-) = 1 + 11 = 12 \quad .$$

Consider l-DAG G as on Figure 6.5(a). As shown in Example 6.22, front elimination of (x_4, v_1) preserves optimality. In Example 4.20 we proved that $[v_3, v_1, v_2]$ solves the edge (vertex) elimination problem on the resulting l-DAG as on Figure 4.28(b). Hence the elimination sequence $[(x_4, v_1)^+, v_3, v_1, v_2]$ solves the optimal edge elimination problem on this graph and

$$\text{EE}(G) = \sigma_G([(x_4, v_1)^+, v_3, v_1, v_2]) = 22 \quad .$$

Consider l-DAG G as on Figure 6.5(b). As shown in Example 6.22 back elimination of (v_2, v_4) and (v_3, v_4) preserve optimality. In Example 4.22 we proved that $[v_2, v_3, v_4, v_1]$ solves the edge (vertex) elimination problem on the resulting l-DAG as on Figure 4.30(a). Hence the elimination sequence $[(v_2, v_4)^-, (v_3, v_4)^-, v_2, v_3, v_4, v_1]$ solves the optimal edge elimination problem on this graph and

$$\text{EE}(G) = \sigma_G([(v_2, v_4)^-, (v_3, v_4)^-, v_2, v_3, v_4, v_1]) = 32 \quad .$$

So we can solve the optimal edge elimination problem for these examples easily. Still if one asks us to find an optimal vertex elimination sequence for these l-DAGs, we do not have any handy results to help us to solve this problem efficiently as we do not have any optimality preserving eliminations at hand that will allow us to reduce the problem. So it is interesting that although the search space for edge elimination is much bigger than that of vertex elimination, there are cases where it is easier for us to solve the edge elimination problem. This is of course due to the fact that the result presented in Lemma 6.5 and Lemma 6.6 is often applicable only to edge elimination problems.

If we restrict ourselves to vertex elimination, we must ensure that the Markowitz degree of i on Figure 6.11(a) is one, in order to be able to apply the Lemma 6.5; as otherwise we would need more than one fma to perform the elimination of i . Hence in this case the statement of the Lemma 6.5 is equal to the result from Naumann (elimination of vertices with Markowitz degree 1 preserves optimality). Similarly Lemma 6.6 can be replaced by the result from Naumann when solving the optimal vertex elimination problem.

In practice we want to accumulate a Jacobian with minimal number of fma's rather than solve the optimal vertex or edge elimination problem. So if we present a solution that uses a mix of edge and vertex eliminations, this result will still be valuable for the user. I suggest therefore that if we want to solve the optimal vertex elimination problem we should always try to use the results from Lemma 6.5 and Lemma 6.6 because they help to reduce the problem and can make the solution even better (as we saw on example of the lion graph). Still once we have performed all possible optimality preserving eliminations we should switch back to the vertex eliminations to keep the search space small.

Chapter 7

Outlook

In this chapter I would like to highlight some open questions and to give you some of my ideas for improving the results contained in this thesis. The ideas discussed here are only preliminary, meaning that in most cases only an outline of the proof is given. The usage of these results is also part of the ongoing research, thus proper algorithms are still to be formulated. Whenever I have a proof idea, I will present it. In cases of lacking the proper proof idea, I will try to motivate why the result is correct. The ideas discussed in this chapter are important to understand the full potential of the results from this thesis. For example for two of the l-DAGs used in [CHMU12] we can beat the lower bound produced by sum of minimal Markowitz degrees only using the ideas shown here.

This chapter is structured as follows. First we briefly discuss the open questions and claims for the Minimal Edge Count problem by vertex and edge elimination. In the second section we deal with optimal vertex and edge elimination problems. This section itself is divided into two subsections. In the first we look at the absorption-free graphs. There we present ways of improving the lower bounds and show eliminations that can be used in algorithms approximating the optimal solution. In the second section we present ideas for new optimality preserving eliminations and discuss ways of improving lower bounds based on reduction to absorption-free graphs.

7.1 Minimal edge count problem

This section is very short. There are two reasons for this. First reason is that the Minimal Edge Count problem on absorption-free graphs is solved (Algorithm 4). Secondly I have spent most of the time on solving the optimal Jacobian accumulation problem and thus most of the results deal with this problem.

In this thesis we introduced lower bounds for the Minimal Edge Count. These are the first results for lower bounds for this problem. As shown in Example 5.10 and Example 5.11, the quality of the bounds strongly depends on the choice of the absorption-free subgraphs. In this thesis we have not presented any strategies for choosing a proper absorption-free subgraph. Further investigations in this direction are necessary to develop algorithms that compute good lower bounds for the MEC and MECV problems.

Another interesting research point in this area is the proof that elimination of vertices with one in- or one out-edge preserves optimality for the MEC problem (compare Conjecture 1). Proving this claim would provide a strong optimality preserving result for arbitrary l-DAGs. This result can be also used for solving the MECV problem. Even though the elimination of the vertices with one in- or one out-edge does not preserve optimality for the MECV problem I believe that elimination of these vertices is a good strategy for finding near optimal results.

7.2 Optimal Jacobian accumulation

As in this thesis I spent more time dealing with the optimal Jacobian accumulation problem, it is natural that we have raised more open questions on this topic. Especially if we take into account that the optimal vertex and edge elimination problems on absorption-free graphs are still not solved, unlike the minimal edge count problem on absorption-free graphs, where we have a linear time algorithm.

In this section I would like to present enhancements for the lower bounds for both absorption-free and non-absorption-free subgraphs, discuss some strategies to find near optimal solutions and present further ideas for optimality preserving eliminations on arbitrary l-DAGs. The ideas presented here show the so-far undiscovered potential of the ideas presented in this thesis. This is especially so if we can provide better lower bounds than the sum of minimal Markowitz degrees on all four examples discussed in [CHMU12].

This section is divided into two subsections. In the first we deal specifically with absorption-free graphs. In the second we try to improve our results for arbitrary l-DAGs.

7.2.1 Absorption-free l-DAGs

In this subsection we will first discuss some of the still open questions. Then some ideas to improve the bounds on absorption-free subgraphs are presented

One such question on absorption-free l-DAGs is a proof that $FE = EE$ on absorption-free l-DAGs (compare Conjecture 2).

Conjecture 2. For any absorption-free l-DAG G holds:

$$FE(G) = EE(G) \quad .$$

This result would close the gap between the result from Griewank where he proved that face eliminations solves the Optimal Jacobian Accumulation problem on absorption-free l-DAG and Theorem 4.4 ($VE = EE$ on absorption-free l-DAGs). Thus we would have a proof that vertex elimination solves the Optimal Jacobian Accumulation problem on absorption-free l-DAGs, which is an interesting outcome not only from the theoretical point of view. It may also lead to more optimality preserving eliminations as happened with $EE = VE$ (compare Lemma 4.16 and Lemma 4.17).

It is also an open question, whether the optimal vertex elimination on absorption-free l-DAGs is an NP-hard problem or not. I believe that there exists a polynomial time algorithm solving the VE problem for absorption-free l-DAGs.

Let us now look at the ways of improving the lower bounds on absorption-free graphs. First we can try and improve the Algorithm 11 by providing a good approach of choosing the next edge to estimate the critical degree. As already outlined the choice of the edges to estimate the critical degree is crucial for the quality of the lower bounds. First attempts to develop such strategies were done in [EH12] under my supervision. In this masters thesis we tried to improve the results from Algorithm 10. Some of these results can be reused for Algorithm 11.

The last idea I would like to discuss can be used to address two problems: to find near optimal results or for determining lower bounds. The optimality preserving elimination according Lemma 4.16 and Lemma 4.17 are very useful results. Still sometimes we run into the problem that we cannot apply these result because the number of predecessors and successors are not appropriate. For example let us take a look at the lion graph without the edge (v_2, y_1) (Figure 2.2(b)). We cannot apply the Lemma 4.17 to vertex v_2 as it has more successors than v_1 predecessors. If we take a look at the proofs of Lemma 4.16 and Lemma 4.17 we will see that we can easily prove the following generalized version of these two lemmas.

Claim 1. Consider absorption-free l-DAG G . The following holds for every $(a, b) \in E^*(G)$, $a, b \in Z(G)$ with $S(a) = \{b\}$

$$\text{VE}(G - a) + |P(a)| \geq \text{VE}(G) \geq \text{VE}(G - a) + \min\{|P(a)|, |S(b)|\}$$

(compare Figure 4.19(a)).

Proof. At this point I will only sketch the proof idea. The first part of the inequality is trivially true. The proof of the second one is based on the proof of the Lemma 4.16. Similar to the proof of Lemma 4.16 we have to distinguish between two cases:

- a is eliminated before b and
- b is eliminated before a .

The proof of the first case is the same as in proof of Lemma 4.16, as in this case the elimination of a preserves optimality. For the second case we compare the costs of the elimination sequences

$$[v_1, \dots, v_{i-1}, b, v_{i+1}, \dots, v_n] \quad \text{and} \quad [a, v_1, \dots, v_{i-1}, (o_1, b)^+, \dots, (o_{|O|}, b)^+, v_{i+1}, \dots, v_n,]$$

as we know that the resulting graph residuals are isomorphic. In the proof of Lemma 4.16 we have shown that

$$\sigma_G([v_1, \dots, v_n]) - \sigma_G([a, v_1, \dots, v_{i-1}, (o_1, b)^+, \dots, (o_{|O|}, b)^+, v_{i+1}, \dots, v_n,]) = \sigma_G(a) - \sigma_{G_{i-1}}(b),$$

where $G_{i-1} = G - [v_1, \dots, v_{i-1}]$. The claim follows then from $|S_{G_{i-1}}(b)| \geq |S_G(b)|$. $\square$

Claim 2. Consider absorption-free l-DAG G . The following holds for every $(a, b) \in E^*(G)$, $a, b \in Z(G)$ with $P(b) = \{a\}$

$$\text{VE}(G - b) + |S(b)| \geq \text{VE}(G) \geq \text{VE}(G - b) + \min\{|P(a)|, |S(b)|\}$$

(compare Figure 4.19(c)).

Proof. Symmetrical to the proof of Claim 1. $\square$

The generalized version of Lemma 4.17 (Claim 2) is applicable to the vertex v_2 on the lion graph without the edge (v_2, y_1) (Figure 2.2(b)). According to Claim 2 the best elimination sequence on this l-DAG starting with the vertex v_2 costs at most one 1 fma more then the optimal vertex elimination sequence as

$$|S(v_2)| - \min\{|S(v_2)|, |P(v_1)|\} = 3 - \min\{3, 2\} = 1 \quad .$$

We can also use Claim 1 and Claim 2 in conjunction with Algorithm 11. As discussed before, each time we underestimate a critical degree of an edge, we lose information. So to improve the lower bound provided by Algorithm 11 we should try to minimize the number of times we underestimate the critical degree of an edge. So whenever none of the optimality preserving eliminations can be performed, we should check whether Claim 1 or Claim 2 are applicable to any of the vertices. This way we can reduce the number of times we underestimate the critical degree of an edge. In the following example we will improve the lower bound we have computed in the Example 4.19.

Example 7.1. Consider l-DAG G as on Figure 4.25(a). In Example 4.19 we applied Algorithm 11 to this l-DAG and got a lower bound of 44. We cannot perform any of the optimality preserving eliminations on this l-DAG but we can apply Claim 2 to the vertex v_3 . Thus we know that the best elimination sequence

starting with elimination of v_3 requires not more than 1 fma compared to the optimal vertex elimination sequence as

$$|S(v_3)| - \min\{|S(v_3)|, |P(v_2)|\} = 3 - \min\{3, 2\} = 1 \quad .$$

Elimination of v_3 yields an l-DAG as on Figure 7.1(a). According to Lemma 4.16 the elimination of v_1 preserves optimality on $G - v_3$. The resulting l-DAG $G - [v_3, v_1]$ is shown on Figure 7.1(b). According to Lemma 4.17 elimination of v_4 preserves optimality on $G - [v_3, v_1]$. Hence $[v_1, v_4, v_2]$ solves the optimal vertex elimination sequence on $G - v_3$ and therefore according to Claim 2

$$\text{VE}(G) \geq \text{VE}(G - v_3) + \min\{|S(v_3)|, |P(v_2)|\} = \sigma_G([v_1, v_4, v_2]) + 2 = 4 + 4 + 35 + 2 = 45 \quad .$$

So we could improve the previous lower bound. Moreover we know that the costs of elimination sequence $[v_3, v_1, v_4, v_2]$ are at most 1 fma higher than the costs optimal vertex elimination sequence.

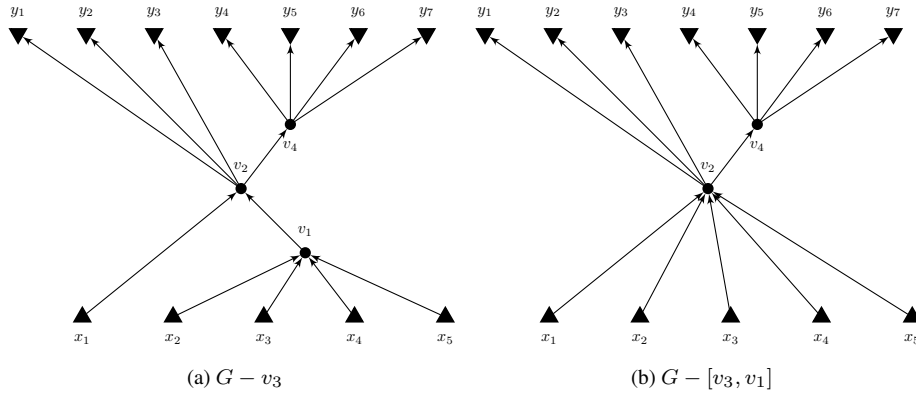


Figure 7.1: Near optimal eliminations on absorption-free l-DAGs. Illustration of Example 7.1.

7.2.2 Arbitrary l-DAGs

For arbitrary l-DAG there are still a lot of open theoretical questions that have not been answered in this thesis. The most famous of them is the question whether any of the following for problems VE, EE, FE or OJA is NP-hard on arbitrary l-DAGs. The other one is the so called "No free refill conjecture" formulated by Naumann. This conjecture states that in an optimal vertex (edge) elimination sequence the edge which was removed from the graph will not be generated again.

A more practical question is to quantify the gain between VE, EE and FE problems. This would help to decide whether it is useful to solve the more complicated problem or should we rather concentrate on solving only the VE one and only exploit edge eliminations to perform optimality preserving eliminations (compare Example 6.23. On absorption-free l-DAGs we know that we can switch to vertex eliminations rather than try to solve the EE problem.

There also emerge questions that are consequences from the results presented in this thesis. Can we transfer the optimality preserving results for absorption-free l-DAGs from Lemma 4.16 and Lemma 4.17 to non-absorption-free graphs? Can we improve the estimators for the critical degree of an edge on arbitrary l-DAG? How can we improve the lower bounds for arbitrary l-DAGs?

In the following I would like to share with you some of my ideas concerning questions from the previous paragraph. First I will show how the results from Lemma 4.16, and Lemma 4.17 can be transferred to arbitrary l-DAGs and sketch the proof. Then ideas on improving the estimators for the critical degree

of an edge are presented. Finally I will discuss the possibilities to improve the lower bounds developed in this thesis. We will then apply these ideas to the l-DAGs from [CHMU12] and show that using these ideas we can provide better lower bounds than the minimal Markowitz degree approach.

Optimality preserving eliminations

If we try to transfer the results from Lemma 4.16 and Lemma 4.17 we will fail to do this. The reason for this is that for both results we rely on the fact that $EE = VE$ on absorption-free l-DAGs. We have only the chance to prove a weaker result, where we assume that for some edge (a, b) , a is the only predecessor of b and b the only successor of a . The proof for this case does not rely on $EE = VE$ as $G - a$ is isomorphic to $G - b$. This result for absorption-free l-DAGs was proved in [MN12]. To transfer this result to arbitrary l-DAGs we have to replace the predecessor and successor sets with minimal predecessor and successor sets.

Claim 3. Consider l-DAG G , $(a, b) \in E^*(G)$ with $|S_G(a)| = 1$ and $|P_G(b)| = 1$.

- If $|P_G(a)| = \min\{\underline{P}_G(a), \underline{S}_G(b)\}$ then elimination of vertex a preserves optimality for the vertex elimination problem.
- If $|S_G(b)| = \min\{\underline{P}_G(a), \underline{S}_G(b)\}$ then elimination of vertex b preserves optimality for the vertex elimination problem.

Proof. The proof idea is similar to that of Lemma 4.16 taking into account that $|\underline{P}_G(a)| \leq |P_{G_i}(a)|$ and $|\underline{S}_G(b)| \leq |S_{G_i}(b)|$ for all i , where $G_i = G - [v_1, \dots, v_i]$. Note that we do not have to perform edge eliminations to get isomorphic graphs. $\square$

We can apply this claim to the l-DAG from Figure 6.6(b).

Example 7.2. Consider l-DAG G as on Figure 6.6(b). The vertex v_2 is the only predecessor of v_3 and v_3 is the only successor of v_2 , moreover

$$2 = |P_G(v_2)| = \min\{|\underline{P}_G(v_2)|, |\underline{S}_G(v_3)|\} = \min\{2, 3\} \quad .$$

Hence elimination of v_2 preserves optimality for the VE problem on this l-DAG. For the reasons of symmetry elimination of v_3 also preserves optimality for the VE problem. The resulting l-DAG $G - [v_2, v_3]$ is isomorphic to the l-DAG on Figure 6.6(a) ($v_2 = v_4$ and $v_3 = v_5$).

Next I will present some ideas to improve the estimators for critical degree of an edge on arbitrary l-DAGs. This would allow us to develop new optimality preserving eliminations according to Lemma 4.18. So far we only have a criterion that tells us whether an edge is critical or not. Possible improvements can be made based on minimal predecessor and successor sets of the source and sink of an edge. Let us discuss this idea on the following example.

Example 7.3. Consider l-DAG G as on Figure 7.2(a). According to Lemma 6.3 the edge (v_1, v_2) is critical as there exists no alternative path from v_1 to v_2 . Obviously it contributes more than one fma to each elimination sequence on this l-DAG as $|\underline{P}_G(v_1)| = 2$ and $|\underline{S}_G(v_2)| = 3$. So if we try to remove this edge explicitly by front or back elimination we will need 3 and 2 fmas respectively. If we do this implicitly by front elimination of all incoming or back elimination of all outgoing edges we will also perform at least 2 multiplications with this edge. Hence the critical degree of this edge is at least two. Back elimination of (v_1, v_2) costs 2 fma's, and the resulting l-DAG $G - (v_1 - v_2)^-$ is a subgraph of G (compare Figure 7.2(b)) hence back elimination of (v_1, v_2) preserves optimality on this l-DAG according to Lemma 4.18.

This idea leads to the following claim.

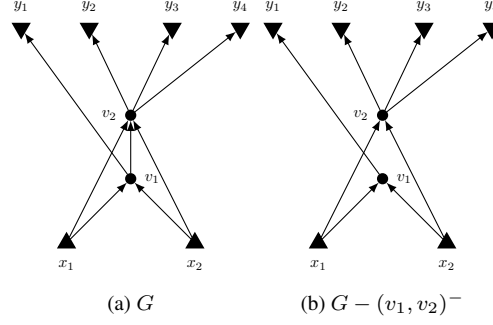


Figure 7.2: Improving estimator for the critical-degree of an edge on arbitrary l-DAGs. Illustration of Example 7.3.

Claim 4. Consider an l-DAG G and $(i, j) \in E^*(G)$. If G contains only one directed path from i to j then (i, j) is critical. Moreover

$$\Delta_G^v(i, j) \geq \min\{|\underline{P}_G(i)|, |\underline{S}_G(j)|\}$$

and

$$\Delta_G^e(i, j) \geq \min\{|\underline{P}_G(i)|, |\underline{S}_G(j)|\} \quad .$$

To my opinion this claim (Claim 4) can be proved using the same proof idea as for Lemma 6.3.

Another idea to improve estimators for the critical edge degree is to relax the requirements from Theorem 4.5. We do not need the whole l-DAG to be absorption-free. We actually need to ensure that the edge is absorption-free (compare Definition 7.1) which is a more local requirement. This idea is discussed in the next part of the outlook (compare Definition 7.1 and Claim 5).

Improving the lower bounds

So far in this thesis we suggested to reduce the given l-DAG to an absorption-free graph and compute the bounds on the absorption-free subgraphs of the original graph. We get good bounds if the l-DAG itself is almost absorption-free meaning that we do not have to remove too many edges to make the graph absorption-free. If a lot of edges are removed during reduction, our bounds get worse as each time we remove an edge, we lose information about the graph. So the optimal edge (vertex) elimination sequence requires less fma on the reduced graph. The only exception is if we remove non-critical edges. So to improve the lower bounds based reduction, there are two general strategies

- retract the information from removed edges or vertices,
- reduce to an arbitrary l-DAG where some of the results of this thesis can be applied

In the following I will first present some ideas on how to retrain the information from removed edges or vertices. Then we will discuss some ideas concerning the second point.

Using the information from removed edges As already mentioned above, by reducing an l-DAG to a subgraph we lose information about the original l-DAG which makes our bounds worse. We could try to get some information from the removed edges. For example by estimating their critical degree using the result from Lemma 6.3 (please read Section 6.5 for more details) or Claim 4. Therefore we would simply apply the same idea as in Algorithm 10 as demonstrated in the following example.

Example 7.4. In this example we will improve the lower bounds computed in Example 6.9. First we improve the lower bound for the l-DAG G on Figure 6.5(a). We remove the edge (x_4, v_1) to build an absorption-free subgraph T . In Example 4.20 we have shown that the lower bound for this subgraph T is 21. Actually we know that this is the optimal solution for this subgraph. Using the Lemma 6.3 we know that the critical degree of (x_4, v_1) is at least one. This means that an optimal edge (vertex) elimination on the absorption-free subgraph needs at least 1 multiplications less, compared to the one on the original l-DAG. Hence we can improve our lower bound by one, yielding an overall lower bound of 22. Recall that, this is also the solution for the optimal edge elimination problem as

$$\sigma_G([v_3, v_1, v_2]) = 2 + 4 + 16 = 22 \quad .$$

In Example 6.9 we computed the lower bound of the l-DAG G on Figure 6.5(b). We removed the edges (v_2, v_4) and (v_3, v_4) to reduce the l-DAG to absorption-free subgraph T . Then we determined the optimal edge (vertex) elimination sequence on the resulting absorption-free subgraph which required 30 fmas. Using the Lemma 6.3 we know that the critical degree of both (v_2, v_4) and (v_3, v_4) is at least one. This means that an optimal edge (vertex) elimination on the absorption-free subgraph needs at least 2 multiplications less. Hence we can improve our lower bound for the l-DAG G by two multiplications, yielding an overall lower bound of 32. We also know that

$$\sigma_G([v_2, v_3, v_4, v_1]) = 3 + 3 + 2 + 24 = 32 \quad ,$$

yielding $32 \geq \text{VE}(G) \geq \text{EE}(G) \geq 32$. Hence $\text{VE}(G) = \text{EE}(G) = 32$.

This approach also allows us to prove that the optimal solution for l-DAG from Figure 6.6(c) requires 15 fmas.

Example 7.5. Consider l-DAG G as on Figure 6.6(c). We remove the edges (v_2, v_3) and (x_2, y_2) to build an absorption-free subgraph T . In Example 4.25 we have shown that the lower bound for this subgraph T is 14, which is also the solution for this subgraph. Using Lemma 6.3 we know that the critical degree of (v_3, v_5) is at least one. This means that an optimal edge (vertex) elimination on the absorption-free subgraph T needs at least 1 multiplications less compared to the optimal elimination sequence on G . Hence we can improve our lower bound by one, yielding an overall lower bound of 15. The edge (x_2, y_2) is not eliminatable, so we cannot improve the bound by retrieving the information from this edge. Luckily

$$\sigma_G([v_4, v_3, v_2, v_1, v_5]) = 2 + 2 + 1 + 2 + 8 = 15$$

and hence $\text{VE}(G) = 15$.

Reduction to arbitrary l-DAG In the following we try to minimize the number of removed edges while constructing a subgraph. Therefore we will reduce the graph to a non-absorption-free subgraph. Theorem 3.8 and Theorem 3.9 allows us to use any subgraphs for lower bounds. So the challenge is to transfer the results we proved for absorption-free l-DAGs to graphs that do not have this property.

One way is to relax the definition of the absorption-free subgraph preserving all proved results. In the definition of absorption-free graphs we demand that there exists at most one directed path between two vertices. Obviously, the existence of parallel paths connecting inputs with outputs does not effect the costs of any vertex (edge) elimination sequence. So we could relax the definition of absorption-free graphs and allow parallel path between inputs and outputs. All results we proved for VE and EE problems on absorption-free graphs should still be valid on this category of graphs. (Note we cannot use this definition in the context of the minimum edge count problem, as the existence of these paths does have influence on the number of edges in the l-DAG. So for the MEC and MECV problems we have to stick with the original definition. For example the butterfly graph (Figure 6.6(d)) satisfies this relaxed definition of absorption-free graphs. All its parallel paths do connect the inputs with the outputs. Let us see what result we get if we assume that Theorem 4.5 holds on the relaxed version of absorption-free graphs.

Example 7.6. Consider l-DAG G is a butterfly graph (Figure 6.6(d)). If we apply Algorithm 10 to this graph, it will estimate the critical degree of all intermediate edges according to Theorem 4.5. The butterfly graph has eight intermediate edges. Each of these edges is connected to two inputs and two outputs. Thus according to Theorem 4.5 the critical degree of the intermediate edge is at least 6. Hence Algorithm 10 would compute a lower bound of $8 \cdot 6 = 48$. By the way the maximum path length in this graph is 3 therefore 48 is even an optimal solution for this problem as we could have also applied the Algorithm 8 that performs optimality preserving eliminations based on critical degree computed in Theorem 4.5.

Another idea is based on the fact that to apply Theorem 4.5 we only need to ensure that eliminations with given edge do not cause absorptions. So we do not need to ensure that the whole l-DAG is absorption-free. So we can perform reduction to graphs that are "locally absorption-free". First of all we need a definition for "locally absorption-free graphs". I suggest the following definition.

Definition 7.1. Consider l-DAG G , $(i, j) \in E^*(G)$. We assume that (i, j) is colored with 1 and all other edges with 0. If all edge elimination sequences applied to the bi-colored version of G never perform absorption with 1 colored edge, then the edge (i, j) is called an absorption-free edge on G .

For example the edge (v_2, v_3) is an absorption-free edge on the bat graph without the edge (v_2, y_4) (compare Figure 7.3(a)). Of course this definition cannot be applied in praxis as it is too difficult to check. So easy to check sufficient conditions for absorption-free edges must be developed in the future.

The proof idea from Theorem 4.5 can easily be extended to prove the following result.

Claim 5. Consider l-DAG G . For any absorption-free edge $(a, b) \in E^*(G)$ holds

$$\Delta_G^e((a, b)) \geq \min\{|X(a) \setminus \{a\}|, |Y(b) \setminus \{b\}|\} + |X(a)||Y(b)|$$

and

$$\Delta_G^v((a, b)) \geq \min\{|X(a) \setminus \{a\}|, |Y(b) \setminus \{b\}|\} + |X(a)||Y(b)| \quad .$$

Proof. The proof idea similar to that from Theorem 4.5. □

Using this result we can improve the lower bound for the bat graph computed in Example 6.8.

Example 7.7. Consider G is a bat graph. We build a subgraph T of G by removing the edge (v_2, y_4) (compare Figure 7.3(a)). Obviously the edge (v_2, v_3) is an absorption-free edge on T . Hence applying Claim 5 to this edge we get

$$\Delta_T^e(v_2, v_3) \geq \min\{2, 4\} + 2 \cdot 4 = 10 \quad .$$

The edge (v_2, v_3) lies only on paths of length three. Hence according to Algorithm 6 back elimination of (v_2, v_3) followed by front elimination of (x_3, v_2) and (x_4, v_2) preserves optimality on T . The resulting l-DAG

$$T' = T - [(v_2, v_3)^-, (x_3, v_2)^+, (x_4, v_2)^+]$$

is shown in Figure 7.3(b). T' is isomorphic to lion graph and hence $\text{EE}(T') = 11$, yielding $\text{EE}(T) = 21$. This means that $\text{EE}(G) \geq 21$. So we could improve the lower bound for the bat graph by one fma.

Of course we should combine this idea with retrieving the information from removed edges. This approach allows us to improve the lower bound for l-DAG on Figure 6.6(a) such that the lower bound is equal to the optimal solution.

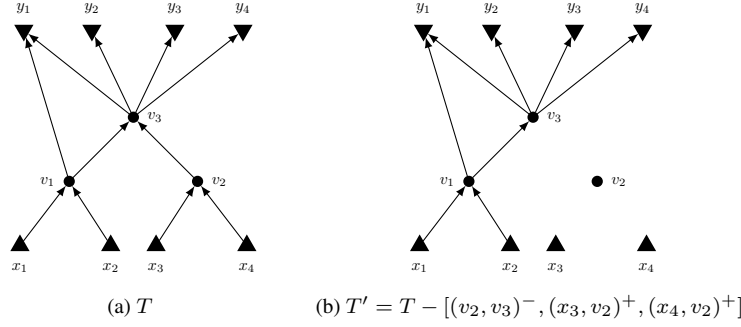


Figure 7.3: Improving lower bound for bat graph (illustrating Example 7.7).

Example 7.8. Consider l -DAG G as on Figure 6.6(a). The edges (x_1, v_2) and (x_4, v_3) are absorption-free edges. Hence according to Claim 5, front elimination of these two edges preserves optimality. The resulting l -DAG $G' = G - [(x_1, v_2)^+, (x_4, v_3)^+]$ is shown on Figure 7.4(a). To estimate the lower bound for EE problem on G' we build an absorption-free subgraph by removing the edges (v_2, y_3) , (v_3, y_1) and (v_3, y_2) . The resulting l -DAG T' is shown on Figure 7.4(b). According to Lemma 6.3 all three removed edges are critical therefore we can increase the lower bound computed on T' by 3. If we apply Algorithm 11 to T' we get a lower bound of 9 as

$$\Delta_{T'}(v_1, v_2) = 2 + 2 \cdot 2 = 6 \quad \text{and} \quad \Delta_{T'}(v_1, v_3) = 1 + 2 \cdot 1 = 3 \quad .$$

Summing up all results we get

$$\text{EE}(G) = 6 + \text{EE}(G') \geq 6 + 3 + \text{EE}(T') = 6 + 3 + 9 = 18 \quad .$$

Hence the computed lower bound corresponds to the optimal solution from [CHMU12]. As

$$\sigma_G([v_2, v_3, v_1]) = 6 + 6 + 6 = 18 \quad ,$$

$$\text{VE}(G) = \text{EE}(G) = 18.$$

The previous example shows us again, the advantage of mixing both edge and vertex eliminations. Especially we see that sometimes we can perform optimality preserving edge eliminations that help us to reduce the search space of the EE problem, such that the actual search space of the EE problem becomes smaller than that of VE problem.

Combining the result from Example 7.8 and Example 7.2 we can easily compute the solution of the VE problem for the l -DAG on Figure 6.6(b).

Example 7.9. Consider l -DAG G as on Figure 6.6(b). In Example 7.2 we proved that elimination of vertices v_2 and v_3 preserve optimality for VE problem on G . The resulting l -DAG $G' = G - [v_2, v_3]$ is isomorphic to the l -DAG on Figure 6.6(a). According to Example 7.8 $\text{VE}(G') = 18$ and hence

$$\text{VE}(G) = \sigma_G([v_2, v_3]) + \text{VE}(G') = 2 + 2 + 18 = 22 \quad .$$

Thus we can confirm the result computed in [CHMU12].

We close this paragraph with an example showing that we do not really need to restrict ourselves to absorption-free property. We can reduce our problem to any subgraph we want as long as we have an idea on how to treat it. We can even choose a subgraph just because we know how to solve the problem for this subgraph.

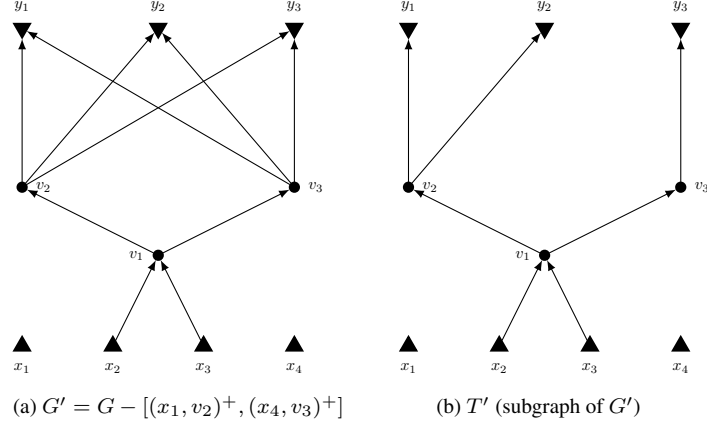


Figure 7.4: Improving lower bound for l-DAG as on Figure 6.6(a) (illustrating Example 7.8).

Example 7.10. Consider the l-DAG G as on Figure 4.21(c). Removing the edge (v_1, y_4) yields a lion graph. Hence lion graph is a subgraph of this graph. We already know that we need 11 multiplications to transform the lion graph in a bipartite graph by edge elimination. Hence 11 is a lower bound for the optimal edge (vertex) elimination on this l-DAG. If we would rely on reducing the l-DAG to absorption-free graph we would have to remove at least one more edge. For example (v_1, y_1) the resulting subgraph is presented on Figure 6.4(b). Giving us a lower bound of 10.

7.3 Solving minimal edge count problem with minimal number of operations

For some applications it is not enough to simply solve the MEC or OJA problem. For example for preaccumulation of basic blocks for the overall forward- or reverse mode of AD, one actually wants to represent the Jacobian of the basic block with minimal number of edges while the accumulation of this minimal representation should use as few operations as possible. So one ends up solving the optimal Jacobian accumulation problem for the minimal edge count problem. Of course we can solve this problems in two steps. First solving the minimal edge count problem and then identify the order in which the eliminations should be performed. This approach works if we solve the minimal edge count problem by vertex eliminations. So in the first step we identify the vertices that must be eliminated to solve the MECV problem. We can then mark the remaining intermediate vertices as independent or dependent ones, to ensure that they are not eliminated, and solve the VE problem on this modified graph. We could use the branch and bound algorithm to solve this modified problem. Still we need to ensure that the results (lower bounds, optimality preserving eliminations) for the VE problem we have proved so far, also hold for this modified problem. So new proofs are required. Another question that has to be answered is, how can the first step of the algorithm be computed without performing the eliminations. Luckily the algorithm for solving the MEC problem on absorption-free graphs allows us to do this. A solution for arbitrary l-DAGs is still outstanding. If we would like to solve the minimal edge count problem for edge eliminations we run into even bigger troubles. As new edges can be generated by an edge elimination sequence. So identifying the set of edges that has to be eliminated is not easy without actually performing the eliminations. Hence getting this two step approach efficient can be a major problem, especially if we want this algorithm, to be not significantly slower than simple solution of the MEC or MECV problems. Therefore I believe that we

need some one pass approach that allows us to find a (approximate) solution for the minimal edge count problem while performing not to many operations. Some of the results of this thesis can be used for such algorithm. Especially such as optimality preserving eliminations but also eliminations of vertices with one in- or out-edge can be a good starting point for such one pass algorithm.

Chapter 8

Conclusion

This thesis contributes theoretical results for solving the MEC and OJA problem by vertex and edge elimination. The intention was to support the development of algorithms for solving these problems. For both problems optimality preserving eliminations and lower bounds based on reduction to absorption-free graphs were presented. It was outlined that although the search space of edge elimination is significantly bigger than that of vertex elimination, in some cases the solution of MEC and EE problems can be obtained more efficiently than the solution of the MECV and VE problems. Therefore mixing edge and vertex eliminations was suggested. The search space of vertex and edge elimination could be reduced significantly exploiting the commutativity of elimination methods proved in Chapter 3. The theoretical base of the existing eliminations methods was established by proving that immediate absorption preserves optimality. A new method of proof using bi-coloring was developed and used to prove the subgraph monotonicity.

Minimum edge count problem For the MEC problem on absorption-free graphs a linear time algorithm was provided. The MEC problem on arbitrary l-DAGs is more complicated. Optimality preserving eliminations and lower bounds based on reduction to absorption-free graphs are presented. These are the first lower bounds published for this problem. Commutativity of vertex elimination is used to show that the search space of the MECV problem is smaller than that of VE problem. This is an interesting fact, as at first glance one tends to believe that it should be exactly the opposite. By exploiting the subgraph monotonicity it is shown that edge eliminations, that do not generate new edges, preserve optimality.

Optimal Jacobian accumulation The discussion of the VE and EE problem takes a bigger part of this thesis. The VE and EE problems on absorption-free build the base for the subsequent development of lower bounds. The most challenging result here and of this thesis in whole, is the proof that $VE = EE$ on absorption-free graphs. Although, this result seems to be only of theoretical interest, practical usage of it was presented in the proof of optimality preserving eliminations. The optimality preserving eliminations based on this result significantly simplified the EE problem on absorption-free DAGs, still a polynomial time algorithm could not be provided. It is even not clear whether the EE problem on absorption-free l-DAGs is NP-hard. Hence, a new approach for lower bounds using the critical degree of an edge is presented. It is shown that these lower bounds are better than the existing ones. Optimality preserving eliminations using critical degree of an edge are presented and used in the algorithm that solves the EE problem on absorption-free graphs with maximal path length 3 in linear time. Using the "subgraph property" the results from absorption-free graphs are reused as lower bounds on arbitrary l-DAGs. It is shown that this approach is not always better than the existing lower bounds. Ways to improve the lower bounds based on reduction were presented in the outlook. With improvements suggested there, the

optimal solution for the EE problem could be derived. The critical degree of an edge is used to develop new optimality preserving eliminations for the EE problem on arbitrary l-DAGs.

Finally I would like to comment on the choice of minimizing the number of fma's as the main objective function in this thesis. This choice seems to be ill-suited for the practical applications, as one would claim that on modern architectures the problems of data locality and sequential data access are more relevant for the overall performance of a program than minimizing the number of operations. So the optimization of the computation of derivatives should happen under these constraints. Nonetheless, I believe that the results of this thesis can be used in optimization of derivative code generation. In particular representing the computational graph with minimal number of edges should be very useful, as it allows to represent the same code with smaller amount of memory. This can address the problem of data locality. The optimal Jacobian accumulation problem can then be used to find an optimal way of accumulating the minimal edge count representation as outlined in Section 7.3.

Appendix A

Appendix

A.1 Code

To generate the results in Section 6.4 the implementation of the generic branch and bound algorithm originally developed in [Pei09] and then extended in [EH12] is used. The package with the software can be retrieved from the author by email. Please extract the archive and see the `README` file for installation details. The code was tested under Linux with GNU C++ compiler version 4.8. The software uses *ANGEL*¹ to represent the l-DAGs. The *ANGEL* Library itself is based on *C++ Boost Libraries*². Both libraries are included in the package.

A.2 Running the branch and bound algorithm

To run the software you need to execute `BAC.exe`. This file is located in the root directory after installation. `BAC.exe` expects the following three command-line parameters:

1. options specifying the elimination method, lower bounds etc. to be used,
2. path to the input file containing the l-DAG to be examined in EliAD-format (see Section A.3),
3. path to the output file (see Section A.4).

The first command-line argument can contain the following options:

- `v` - solve VE problem (cannot be used in conjunction with `e`)
- `e` - solve EE problem (cannot be used in conjunction with `v`)
- `l` - use lower bounds according to Algorithm 25
- `m` - use lower bounds based on Algorithm 9 (can only be used in conjunction with option `l`)
- `c` - use lower bounds based on Algorithm 10 (can only be used in conjunction with `l`)
- `r` - exploit commutativity according to Algorithm 21 or Algorithm 23 depending on whether VE or EE problem is solved

¹ANGEL - Automatic differentiation Nested Graph Elimination Library <http://angellib.sourceforge.net>

²Boost C++ Libraries <http://www.boost.org>

- a - exploit commutativity of adjacent vertices (Algorithm 22) only for VE problem (can only be used in conjunction with option `bb—r—`)

The first command-line argument must contain either `v` or `e`. All other settings are optional. The settings can be provided in arbitrary order. For example the call

```
BAC.ex vlmra input_file output_file
```

would solve the VE problem on the l-DAG provided in the input file, using the both lower bounds according Algorithm 25 and based on Markowitz degree (Algorithm 9) and exploiting commutativity as in Algorithm 22.

Independently from the specified options following information will be printed to the standard output (stdout) during the execution of the branch and bound algorithm:

- the currently best solution (vertex or edge elimination sequence) and the corresponding costs
- a progress display showing the percentage of all elimination sequences that have been checked so far. (Note this is only a rough estimator based on the assumption that the underlying search tree is balanced.)

The elimination sequence is represented as follows:

- eliminations inside the elimination sequence are separated with brackets
- vertex elimination is denoted with a single number corresponding to the index of this vertex
- edge elimination is represented with three numbers. The first two represent the sink and the source of the edge respectively. The last number stands for front (1) or back elimination (0).

After each execution the software prints the optimal vertex (edge) elimination sequence and the corresponding costs to the stdout. The optimal elimination sequence is written in the same notation from above.

A.3 EliAD-format

To specify the l-DAG in EliAD-format we first need to set the overall number of vertices through

- $n = |V(G)|$

and then specify all edges of the l-DAG in the following format

- "[sink] [source] [weight]" .

The edges should be separated by line breaks and enclosed by

```
CG = [
]
```

In Figure A.1(a) you can see the lion graph in EliAD-format. The edge weights are used to build an absorption-free subgraph for computing lower bounds. If all weights are greater than zero then Algorithm 3 is used build the absorption-free subgraph. If any of the weights is zero, then only zero weighted edges are removed. This way the user has the opportunity to specify a specific absorption-free subgraph. The software will not check whether the resulting l-DAG is absorption-free or not.

```

n = 8
CG = [
  3  1  2
  3  2  2
  4  3  2
  8  3  2
  5  4  2
  6  4  2
  7  4  2
  8  4  2
]

```

(a) EliAD-format

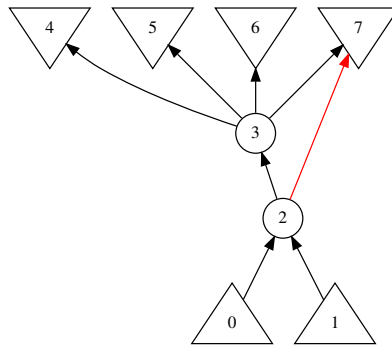
(b) Output file of the processed with *Graphviz*

Figure A.1: Lion graph

A.4 Output file

The output file contains the initial l-DAG in `.dot`-format. The edges that were removed to build an absorption-free l-DAG are marked with red color. `.dot`-format can be processed by the tool *Graphviz*³ (compare Figure A.1(b)). Please refer to the *Graphviz* website for documentation and explanation of the `.dot`-format. The output file is meant to be used as sanity check to ensure that the correct l-DAG and subgraph were used.

³Graphviz <http://www.graphviz.org>

Bibliography

- [AGN03] A. Albrecht, P. Gottschling, and U. Naumann. Markowitz-type heuristics for computing Jacobian Matrices efficiently. In P. S. et al., editor, *Computational Science – ICCS 2003*, volume 2658 of *LNCSE*, pages 575–584. Springer, 2003.
- [AKHL⁺08] H. Abdel-Khalik, P. Hovland, A. Lyons, T. Stover, and J. Utke. A low rank approach to automatic differentiation. In *Advances in Automatic Differentiation*, pages 55–65. Springer, 2008.
- [Bau74] F. L. Bauer. Computational graphs and rounding errors. *SIAM Journal on Numerical Analysis*, 11:87–96, 1974.
- [BBH⁺08] C. Bischof, M. Bücker, P. Hovland, U. Naumann, and J. Utke, editors. *Advances in Automatic Differentiation*, number 64 in *Lecture Notes in Computational Science and Engineering (LNCSE)*. Springer, 2008.
- [BCH⁺05] M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Implementations*, volume 50 of *Lecture Notes in Computational Science and Engineering*. Springer, New York, NY, 2005.
- [BCH⁺06] M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Tools*, number 50 in *Lecture Notes in Computational Science and Engineering (LNCSE)*. Springer, 2006.
- [BMMN11] M. Beckers, V. Mosenkis, M. Maier, and U. Naumann. Adjoint subgradient calculation for McCormick relaxations. Technical Report AIB-2011-10, RWTH Aachen, October 2011.
- [BMN12] M. Beckers, V. Mosenkis, and U. Naumann. Adjoint mode computations of subgradients for McCormick relaxations. In *Recent Advances in Algorithmic Differentiation*, pages 103–113. Springer, 2012.
- [BN10] M. Beckers and U. Naumann. Propagation of Uncertainties using the Method of Moments. In *Proceedings of The Tenth International Conference on Computational Structures Technology*. Civil-Comp Press, 2010.
- [BN11] M. Beckers and U. Naumann. Uncertainty Quantification for First-Order Nonlinear Optimization Algorithms. In *CFD & Optimization*. ECCOMAS Thematic Conference, 2011.
- [CFG⁺02] G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors. *Automatic Differentiation of Algorithms – From Simulation to Optimization*, New York, 2002. Springer.
- [CHMU12] J. Chen, P. Hovland, T. Munson, and J. Utke. An integer programming approach to optimal derivative accumulation. In S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, editors,

- Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*, pages 221–231. Springer Berlin Heidelberg, 2012.
- [CMB⁺11] C. Corbett, M. Maier, M. Beckers, U. Naumann, A. Ghobeity, and A. Mitsos. Compiler-generated subgradient code for McCormick relaxations. Technical Report AIB-2011-25, RWTH Aachen, November 2011.
- [CN11] C. Corbett and U. Naumann. Demonstration of a branch-and-bound algorithm for global optimization using McCormick relaxations. Technical Report AIB-2011-24, RWTH Aachen, November 2011.
- [EH12] T. El Habbaj. Branch and Bound auf Meta-Berechnungsgraphen. Master’s thesis, RWTH Aachen University, January 2012.
- [FHP⁺12] S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, editors. *Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, 2012.
- [Gil80] J. Gilbert. A note on the NP-completeness of vertex elimination on directed graphs. *J. Alg. Disc. Meth.*, 1(3):292–294, September 1980.
- [GMP05] A. H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, 2005.
- [GN03] A. Griewank and U. Naumann. Accumulating Jacobians as chained sparse matrix products. *Mathematical Programming*, 95(3):555–571, 2003.
- [GR91] A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markowitz rule. In A. Griewank and G. F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. SIAM, Philadelphia, PA, 1991.
- [Gri03] A. Griewank. A mathematical view of automatic differentiation. *Acta Numerica*, 12:321–398, 5 2003.
- [GV04] A. Griewank and O. Vogel. Analysis and exploitation of Jacobian scarcity. In H. Bock, E. Kostina, H. Phu, and R. Rannacher, editors, *Modelling, Simulation and Optimization of Complex Processes*, pages 149–164, New York, NY, 2004. Springer.
- [GW08] A. Griewank and A. Walther. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation. Second Edition*. SIAM, 2008.
- [HS05a] S. Hossain and T. Steihaug. Efficient computation of sparse Jacobian matrices using patterns. In Bücker et al. [BCH⁺05].
- [HS05b] S. Hossain and T. Steihaug. Computing sparse Jacobian matrices optimally. In Bücker et al. [BCH⁺05], pages 77–87.
- [Jab14] R. Jaber. On testing uni-connectivity in directed graphs and some related problems. *CoRR*, abs/1412.1639, 2014.
- [JP98] R. Jessani and M. Putrino. Comparison of single- and dual-pass multiply-add fused floating-point units. *Computers, IEEE Transactions on*, 47(9):927–937, Sep 1998.
- [Kru56] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

- [LLN11] J. Lotz, K. Leppkes, and U. Naumann. dco/c++ - derivative code by overloading in C++. Technical Report AIB-2011-06, RWTH Aachen, May 2011.
- [LMSK63] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Oper. Res.*, 11(6):972–989, December 1963.
- [LU08] A. Lyons and J. Utke. On the practical exploitation of scarcity. In *Advances in Automatic Differentiation*, pages 103–114. Springer, 2008.
- [LW66] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Oper. Res.*, 14(4):699–719, August 1966.
- [MN12] V. Mosenkis and U. Naumann. On optimality preserving eliminations for the minimum edge count and optimal Jacobian accumulation problems in linearized dags. *Optimization Methods and Software*, 27(2):337–358, 2012.
- [Nau99] U. Naumann. *Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs*. PhD thesis, Technical University Dresden, Feb. 1999.
- [Nau00] U. Naumann. An Enhanced Markovitz Rule for Accumulating Jacobians Efficiently. In A. H. et al., editor, *Proceedings of 15th Conference on Scientific Computing (ALGO-RITHMY 2000)*, pages 320–329. Slovak University of Technology, 2000.
- [Nau02] U. Naumann. Cheaper Jacobians by simulated annealing. *SIAM Journal on Optimization*, 13(3):660–674, 2002.
- [Nau04] U. Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Mathematical Programming*, 99(3):399–421, 2004.
- [Nau08] U. Naumann. Optimal Jacobian accumulation is NP-complete. *Mathematical Programming*, 112(2):427–441, 2008.
- [NG01] U. Naumann and P. Gottschling. Prospects for simulated annealing in Automatic Differentiation. In K. Steinhöfel, editor, *Stochastic Algorithms: Foundations and Applications*, volume 2264 of *LNCS*, pages 131–144. Springer, March 2001.
- [NH08] U. Naumann and Y. Hu. Optimal vertex elimination in single-expression-use graphs. *ACM Transactions on Mathematical Software*, 35(1):2:1–2:20, July 2008.
- [NSST09] U. Naumann, O. Schenk, H. Simon, and S. Toledo, editors. *Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings. Leibnitz-Zentrum für Informatik, 2009.
- [NU05] U. Naumann and J. Utke. Optimality-preserving elimination of linearities in Jacobian accumulation. *Electronic Transactions on Numerical Analysis*, 21:134–150, 2005. Special Volume on Combinatorial Scientific Computing.
- [Pei09] E. Peise. Branch and Cut auf Meta-Berechnungsgraphen. Bachelor thesis, RWTH Aachen University and University of Hagen, September 2009.
- [Ros70] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597 – 609, 1970.

- [Ros72] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Academic Press, New York, 1972.
- [RT78] D. Rose and R. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *J. Appl. Math.*, 34(1):176–197, January 1978.
- [TBPF05] M. Tadjouddine, F. Bodman, J. D. Pryce, and S. A. Forth. Improving the performance of the vertex elimination algorithm for derivative calculation. In Bücker et al. [BCH⁺06], pages 111–120.
- [TFPR02] M. Tadjouddine, S. A. Forth, J. D. Pryce, and J. K. Reid. *Performance Issues for Vertex Elimination Methods in Computing Jacobians using Automatic Differentiation*, pages 1077–1086. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.
- [UN12] J. Utke and U. Naumann. Combinatorial problems in Open/AD. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*, Computational Science. Chapman and Hall/CRC, January 2012. chap. 6.
- [UNF⁺08] J. Utke, U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, and C. Wunsch. OpenAD/F: A modular open-source tool for Automatic Differentiation of Fortran codes. *ACM Transactions on Mathematical Software*, 34(4):18:1–18:36, July 2008.
- [Var11] E. Varnik. *Exploitation of Structural Sparsity in Algorithmic Differentiation*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2011.
- [VN09] E. Varnik and U. Naumann. What Color is the Non-Constant Part of Your Jacobian? In [NSST09], 2009. Extended Abstract.

Index

Symbols

$(i, j)^+$	front elimination of edge (i, j)	20
$(i, j)^-$	back elimination of edge (i, j)	20
$E^*(G)$	set of eliminatable edges of G	28
$E^+(G)$	set of front eliminatable edges of G	28
$E^-(G)$	set of front eliminatable edges of G	28
$P_G(v)$	predecessor set of vertex v on l-DAG G	20
$S_G(v)$	successor set of vertex v on l-DAG G	20
$X(G)$	$\equiv \{v \in V(G) \mid P_G(v) = \emptyset\}$	28
$Y(G)$	$\equiv \{v \in V(G) \mid S_G(v) = \emptyset\}$	28
$Z(G)$	set of intermediate vertices of G	28
$X_G(v)$	set of inputs of G connected to v	28
$Y_G(v)$	set of outputs of G connected to v	28
$X_G((i, j))$	set of inputs of G connected to (i, j)	28
$Y_G((i, j))$	set of outputs of G connected to (i, j)	28
σ_G	cost of vertex (edge) elimination [sequence]	21
$\text{tgt}(i, j)$	sink of edge (i, j)	20
$\text{src}(i, j)$	source of edge (i, j)	20
$\text{EE}(G)$	cost of the optimal edge elimination sequence on G	22
$\text{VE}(G)$	cost of the optimal vertex elimination sequence on G	22
$\text{MEC}(G)$	number of edges contained in the solution of the MEC problem on G	22
$\text{MECV}(G)$	number of edges contained in the solution of the MECV problem on G	22
EE	optimal edge elimination problem	22
VE	optimal vertex elimination problem	22
MEC	minimal edge count problem by edge elimination	22
MECV	minimal edge count problem by vertex elimination	22
$\text{mark}_G(v)$	$\equiv P_G(v) \cdot S_G(v) $	28
$\mathcal{M}(G)$	$\equiv \{v \in Z(G) \mid S_G(v) = 1 \vee P_G(v) = 1\}$	29
T_{k_1, k_2}	transformation operator. See definition on page	37
$I_{i, j}^G$	bridge degree of i and j on G . See definition on page	71
$[w_1, \dots, w_k]$	vertex (edge) elimination sequence	21
$\Delta_G^v((i, j))$	critical degree of (i, j) in G for vertex elimination	88
$\Delta_G^e((i, j))$	critical degree of (i, j) in G for edge elimination	88
$\Delta_G((i, j))$	critical degree of (i, j) in G on absorption-free l-DAGs	88
$H[S]$	l-DAG H induced by the set S of edges	29

$\underline{P}(v)$	minimal $X(G) - v$ vertex cut in G	136
$\underline{S}(v)$	minimal $v - Y(G)$ vertex cut in G	136
$\text{mark}_G(v)$	minimal Markowitz degree of v in G	136

Definitions

absorption-free	56
bi-coloring	32
bridge	85
bridge degree	71
critical degree	88
complete elimination sequence	22
edge elimination (front/back)	20
edge elimination sequence	21
forward connected vertex elimination sequence	69
l-DAG	11
Markowitz degree	28
optimality preserving eliminations	22
pure edge elimination	20
semi-bridge	87
subgraph	32
subgraph monotonicity	32
underlying undirected graph	68
vertex elimination	21
vertex elimination sequence	21