Optimal Vertex Elimination in Single-Expression-Use Graphs

Uwe Naumann, Yuxiao Hu

Arne Lorenz

Lehrstuhl B für Mathematik
RWTH Aachen

GK-Sommerschule 2006

Gliederung

Voraussetzungen

- Eine vektorwertige Funktion:

$$F : \mathbb{R}^n \to \mathbb{R}^m : x \mapsto y = F(x)$$

Voraussetzungen

- Eine vektorwertige Funktion:

$$F : \mathbb{R}^n \to \mathbb{R}^m : x \mapsto y = F(x)$$

- Implementation – elementare Zuweisungen:

$$v_j := \varphi_j(v_i)_{i \prec j} \qquad j = 1, \ldots, p + m$$

Voraussetzungen

- Eine vektorwertige Funktion:

$$F : \mathbb{R}^n \to \mathbb{R}^m : x \mapsto y = F(x)$$

- Implementation – elementare Zuweisungen:

$$v_j := \varphi_j(v_i)_{i \prec j} \qquad j = 1, \ldots, p+m$$

- Partialordnung:

$$v_i \prec v_j \quad :\Longleftrightarrow \quad \frac{\partial \varphi_j}{\partial v_i} \neq 0$$

  (v_j hängt direkt von v_i ab)
- transitiver Abschluß: $\prec^*$

Computational Graph

- Computational Graph: gerichteter, azyklischer Graph $G = (V, E)$

$$\begin{array}{lcll} V & = & X \dot{\cup} Z \dot{\cup} Y & \text{Ecken} \\ E & = & \{(v_i, v_j) | \, v_i \prec v_j\} & \text{Kanten} \end{array}$$

Computational Graph

- Computational Graph: gerichteter, azyklischer Graph $G = (V, E)$

$$\begin{array}{lcll} V & = & X \dot\cup Z \dot\cup Y & \text{Ecken} \\ E & = & \{(v_i, v_j) | \, v_i \prec v_j\} & \text{Kanten} \end{array}$$

  - $X = \{v_{1-n} = x_1, \ldots, v_0 = x_n\}$,
  - $Z = \{v_1, \ldots, v_p\}$,
  - $Y = \{v_{p+1} = y_1, \ldots, v_{p+m} = y_m\}$.

Computational Graph

- Computational Graph: gerichteter, azyklischer Graph $G = (V, E)$

$$\begin{array}{lcll} V & = & X \dot\cup Z \dot\cup Y & \text{Ecken} \\ E & = & \{(v_i, v_j) | \, v_i \prec v_j\} & \text{Kanten} \end{array}$$

  - $X = \{v_{1-n} = x_1, \ldots, v_0 = x_n\}$,
  - $Z = \{v_1, \ldots, v_p\}$,
  - $Y = \{v_{p+1} = y_1, \ldots, v_{p+m} = y_m\}$.

- Linearisierter Graph:
  Versehe die Kante (v_i, v_j) mit der *lokalen partiellen Ableitung*

$$c_{j,i} = \frac{\partial \varphi_j}{\partial v_i}(v_k).$$

Beispiel

Die Funktion:

$$y = F(x_1, x_2) = x_1 \sin(x_1)/x_2$$

Die Implementation:

$$\begin{aligned} v_{-1} &= x_1 \\ v_0 &= x_2 \\ v_1 &:= \sin(v_{-1}) \\ v_2 &:= v_{-1}\, v_1 \\ y = v_3 &:= v_2/v_0 \end{aligned}$$

Beispiel

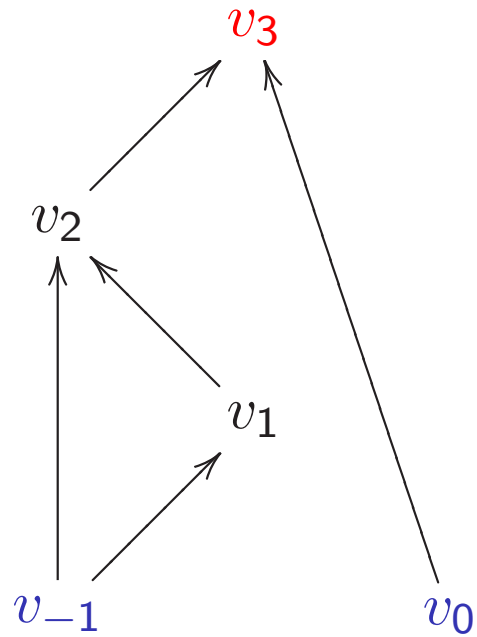
Die Funktion:

$$y = F(x_1, x_2) = x_1 \sin(x_1)/x_2$$

Die Implementation:

$$\begin{aligned} v_{-1} &= x_1 \\ v_0 &= x_2 \\ v_1 &:= \sin(v_{-1}) \\ v_2 &:= v_{-1}\, v_1 \\ y = v_3 &:= v_2/v_0 \end{aligned}$$

Computational Graph:



Beispiel

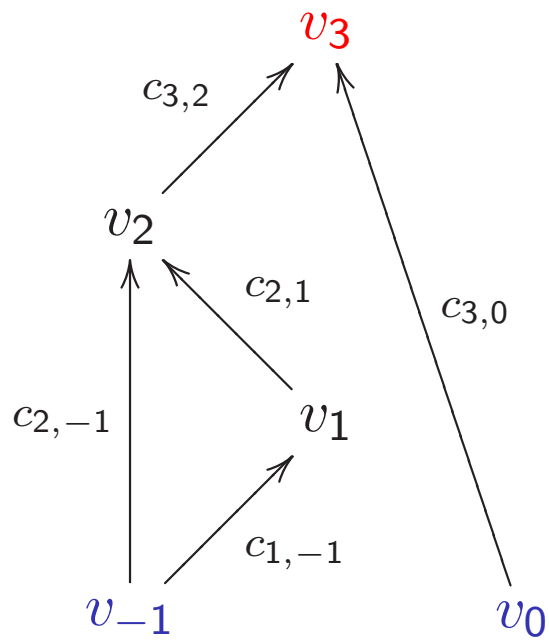
Die Funktion:

$$y = F(x_1, x_2) = x_1 \sin(x_1)/x_2$$

lokale Ableitungen:

$$\begin{aligned} c_{1,-1} &= \cos(v_{-1}) \\ c_{2,-1} &= v_1 \\ c_{2,1} &= v_{-1} \\ c_{3,0} &= -v_2/v_0^2 \\ c_{3,2} &= 1/v_0 \end{aligned}$$

Linearisierter Graph:



Ziel: Jacobimatrix

- Berechne die Jacobimatrix

$$F'(i,j) = \frac{\partial y_j}{\partial x_i}(x) = \sum_{\text{Pfade } x_i \to y_j} \prod_{(k,l) \in \text{Pfad}} c_{k,l}.$$

Ziel: Jacobimatrix

- Berechne die Jacobimatrix

$$F'(i,j) = \frac{\partial y_j}{\partial x_i}(x) = \sum_{\text{Pfade } x_i \to y_j} \prod_{(k,l) \in \text{Pfad}} c_{k,l}.$$

- Transformiere G_{lin} in einen bipartiten Graphen

$$G' = (X \dot\cup \emptyset \dot\cup Y, E')$$

- Eliminiere Ecken/Kanten

Ziel: Jacobimatrix

- Berechne die Jacobimatrix

$$F'(i,j) = \frac{\partial y_j}{\partial x_i}(x) = \sum_{\text{Pfade } x_i \to y_j} \prod_{(k,l) \in \text{Pfad}} c_{k,l}.$$

- Transformiere G_{lin} in einen bipartiten Graphen

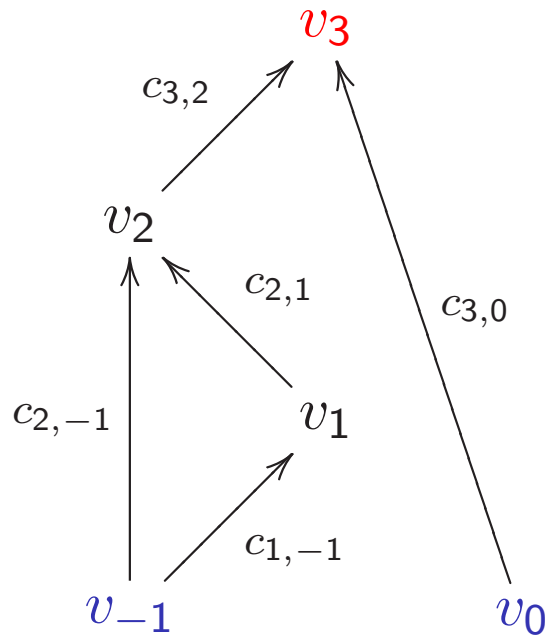
$$G' = (X \dot\cup \emptyset \dot\cup Y, E')$$

- Eliminiere Ecken/Kanten
- Kosten:
  - # Multiplikationen
  - # Additionen

Beispiel

Berechne die Jacobimatrix:

- Optimale Lösung:

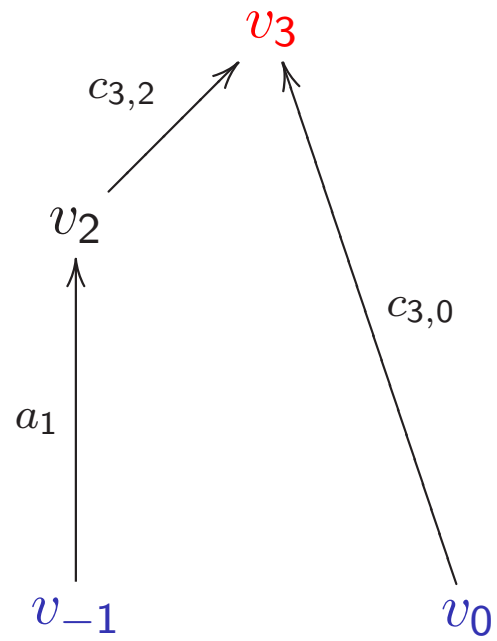


Beispiel

Berechne die Jacobimatrix:

- Optimale Lösung:

$$a_1 \quad = \quad c_{2,1} \cdot c_{1,-1} + c_{2,-1}$$

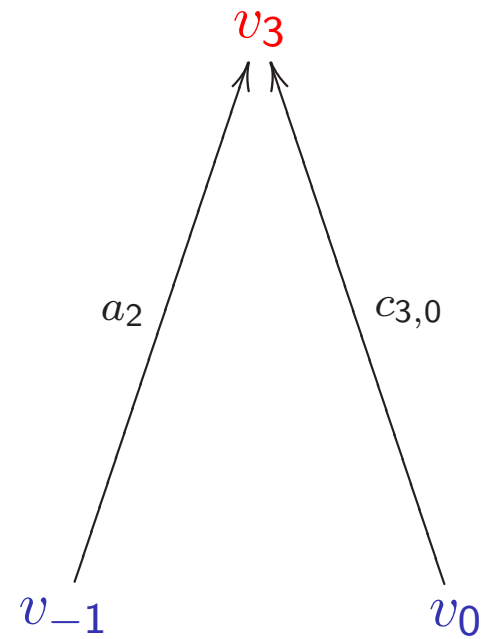


Beispiel

Berechne die Jacobimatrix:

- Optimale Lösung:

$$
\begin{aligned}
a_1 &= c_{2,1} \cdot c_{1,-1} + c_{2,-1} \\
a_2 &= c_{3,2} \cdot a_1
\end{aligned}
$$

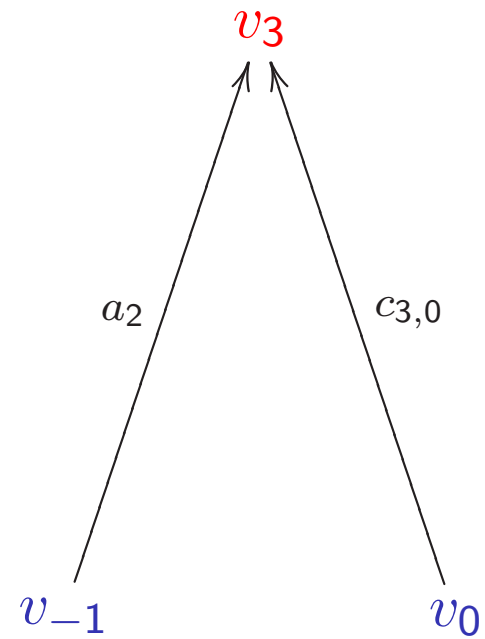


Beispiel

Berechne die Jacobimatrix:

- Optimale Lösung:

$$
\begin{aligned}
a_1 &= c_{2,1} \cdot c_{1,-1} + c_{2,-1} \\
a_2 &= c_{3,2} \cdot a_1 \\
(a_3 &= c_{3,0})
\end{aligned}
$$



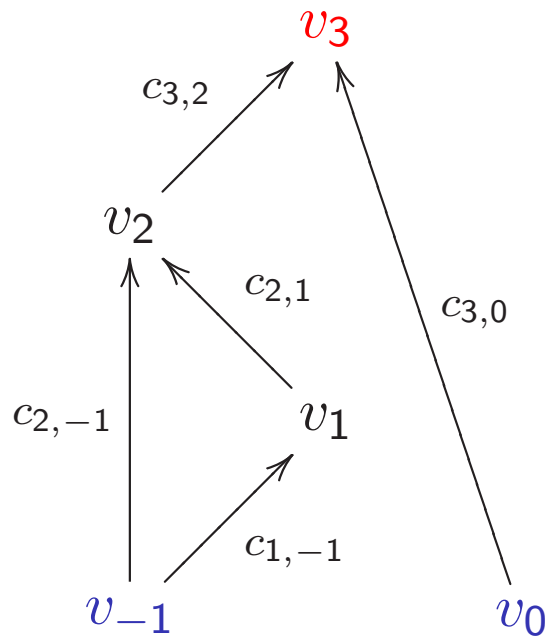
Beispiel

Berechne die Jacobimatrix:

- Optimale Lösung:

$$\begin{aligned} a_1 &= c_{2,1} \cdot c_{1,-1} + c_{2,-1} \\ a_2 &= c_{3,2} \cdot a_1 \\ (a_3 &= c_{3,0}) \end{aligned}$$

- schlechtere Lösung:



Beispiel

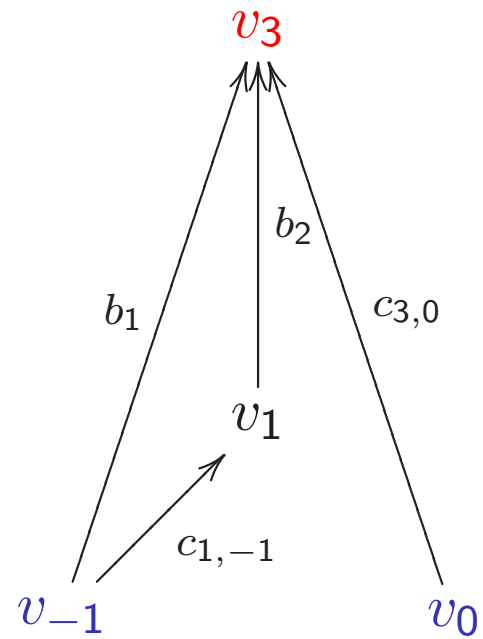
Berechne die Jacobimatrix:

- Optimale Lösung:

$$\begin{aligned} a_1 &= c_{2,1} \cdot c_{1,-1} + c_{2,-1} \\ a_2 &= c_{3,2} \cdot a_1 \\ (a_3 &= c_{3,0}) \end{aligned}$$

- schlechtere Lösung:

$$\begin{aligned} b_1 &= c_{3,2} \cdot c_{2,-1} \\ b_2 &= c_{3,2} \cdot c_{2,1} \end{aligned}$$



Beispiel

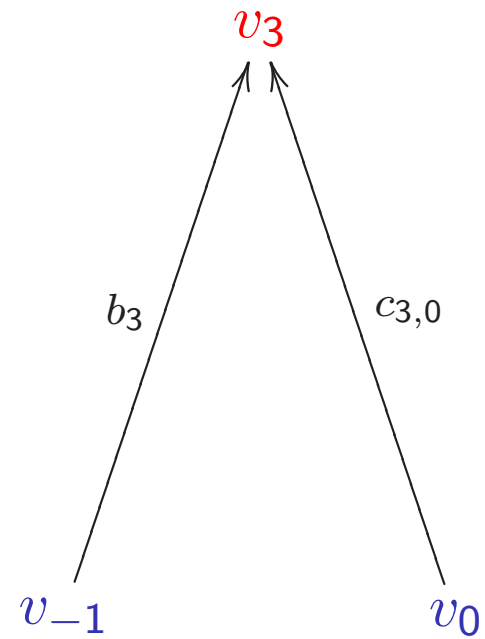
Berechne die Jacobimatrix:

- Optimale Lösung:

$$\begin{aligned} a_1 &= c_{2,1} \cdot c_{1,-1} + c_{2,-1} \\ a_2 &= c_{3,2} \cdot a_1 \\ (a_3 &= c_{3,0}) \end{aligned}$$

- schlechtere Lösung:

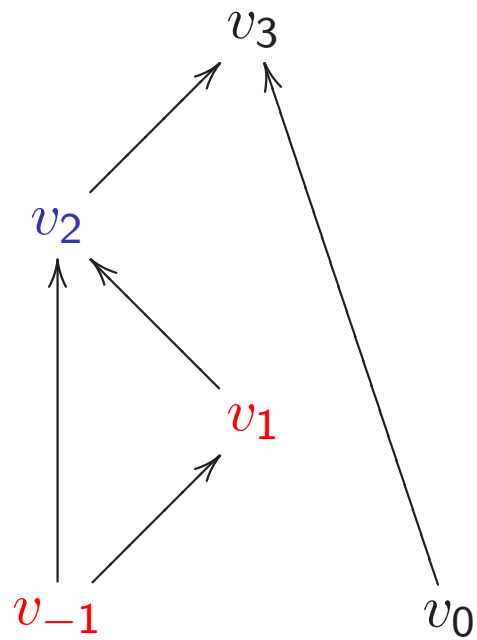
$$\begin{aligned} b_1 &= c_{3,2} \cdot c_{2,-1} \\ b_2 &= c_{3,2} \cdot c_{2,1} \\ b_3 &= b_2 \cdot c_{1,-1} + b_1 \end{aligned}$$



Graphentheorie

Sei $v_j \in V$ ein Vertex.

- P_j: direkte Vorgänger von v_j
- $|P_j|$: Innengrad

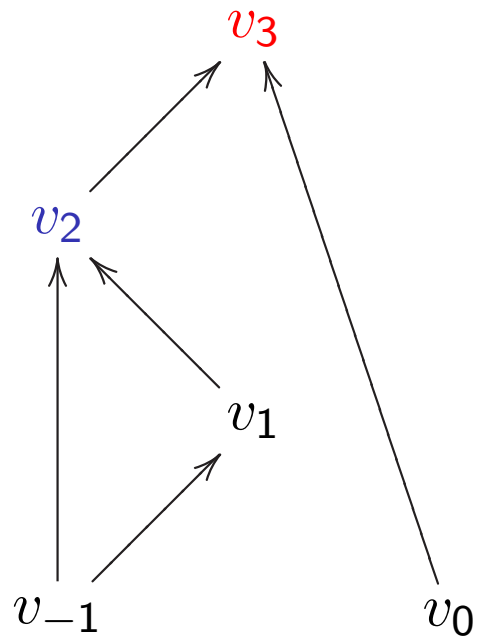


$$|P_2| = 2$$

Graphentheorie

Sei $v_j \in V$ ein Vertex.

- P_j: direkte Vorgänger von v_j
- $|P_j|$: Innengrad
- S_j: direkte Nachfolger
- $|S_j|$: Außengrad

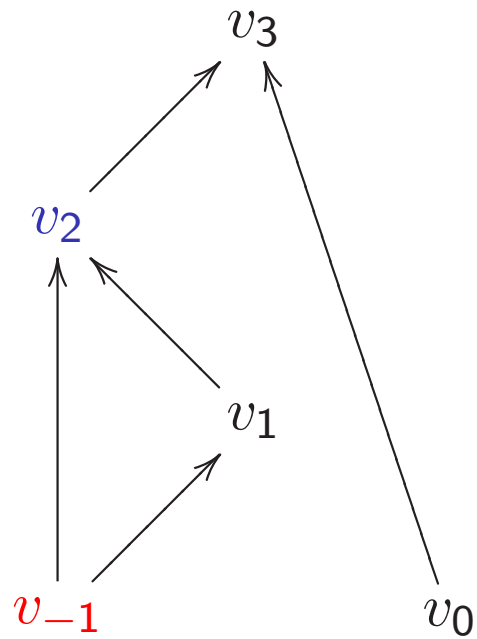


$$|S_2| = 1$$

Graphentheorie

Sei $v_j \in V$ ein Vertex.

- P_j: direkte Vorgänger von v_j
- $|P_j|$: Innengrad
- S_j: direkte Nachfolger
- $|S_j|$: Außengrad
- $\underline{P}_j$, minimale Vorgänger über alle Vertexeliminationen

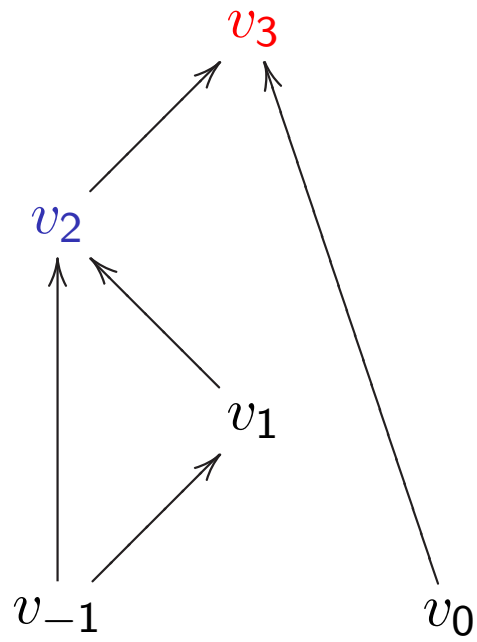


$|\underline{P}_2| = 2$

Graphentheorie

Sei $v_j \in V$ ein Vertex.

- P_j: direkte Vorgänger von v_j
- $|P_j|$: Innengrad
- S_j: direkte Nachfolger
- $|S_j|$: Außengrad
- $\underline{P}_j$, minimale Vorgänger über alle Vertexeliminationen
- $\underline{S}_j$, minimale Nachfolger über alle Vertexeliminationen



$|\underline{S}_2| = 1$

SEU-Graphen

Definition

Ein Single-Expression-Use-Graph (SEU-Graph) ist ein linearisierter Graph $G_{lin} = (X \dot\cup Z \dot\cup Y, E)$ mit:

- $|S_z| = 1 \; \forall z \in Z$,
- $c_{i,j}(x)$ algebraisch unabhängig.

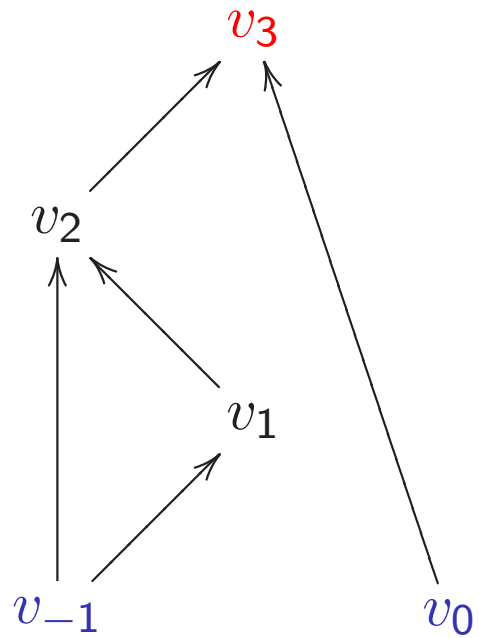
SEU-Graphen

Definition

Ein Single-Expression-Use-Graph (SEU-Graph) ist ein linearisierter Graph $G_{lin} = (X \dot\cup Z \dot\cup Y, E)$ mit:

- $|S_z| = 1 \; \forall z \in Z$,
- $c_{i,j}(x)$ algebraisch unabhängig.

Beispiel



Eigenschaften

SEU-Graph: $G_{lin} = (X \dot{\cup} Z \dot{\cup} Y, E)$, $|S_z| = 1 \forall z \in Z$

- Bijektion:

$$\text{Pfade } X \to Y \longleftrightarrow S_X$$

Eigenschaften

SEU-Graph: $G_{lin} = (X \dot\cup Z \dot\cup Y, E)$, $|S_z| = 1 \forall z \in Z$

- Bijektion:

  Pfade $X \to Y \longleftrightarrow S_X$

- Wenn vorhanden, ist $(z \to j)$ eindeutig.

Eigenschaften

SEU-Graph: $G_{lin} = (X \dot\cup Z \dot\cup Y, E)$, $|S_z| = 1 \forall z \in Z$

- Bijektion:

  Pfade $X \to Y \longleftrightarrow S_X$

- Wenn vorhanden, ist $(z \to j)$ eindeutig.
- X-j-Schnitt:
  - $S \subseteq V \backslash \{j\}$
  - Jeder Weg $(x \to j)$ enthält ein $s \in S$.

Eigenschaften

SEU-Graph: $G_{lin} = (X \dot\cup Z \dot\cup Y, E)$, $|S_z| = 1 \forall z \in Z$

- Bijektion:

  Pfade $X \to Y \longleftrightarrow S_X$

- Wenn vorhanden, ist $(z \to j)$ eindeutig.
- X-j-Schnitt:
  - $S \subseteq V \backslash \{j\}$
  - Jeder Weg $(x \to j)$ enthält ein $s \in S$.
- $\underline{P_j}$ ist ein minimaler X-j-Schnitt.

Eigenschaften

SEU-Graph: $G_{lin} = (X \dot\cup Z \dot\cup Y, E)$, $|S_z| = 1 \forall z \in Z$

- Bijektion:

  Pfade $X \to Y \longleftrightarrow S_X$

- Wenn vorhanden, ist $(z \to j)$ eindeutig.
- X-j-Schnitt:
  - $S \subseteq V \backslash \{j\}$
  - Jeder Weg $(x \to j)$ enthält ein $s \in S$.
- $\underline{P_j}$ ist ein minimaler X-j-Schnitt.
- Es gibt immer minimale X-j-Schnitte

$$C_v \subseteq X \cup P_j$$

Minimale Kosten – Additionen

Lemma

Alle Eliminationssequenzen für SEU-Graphen benötigen $\sum_{i \in X} |S_i| - \mu$ *Additionen.*

$\mu =$ # Jacobimatrixeinträge $\neq 0$

Minimale Kosten – Additionen

Lemma

Alle Eliminationssequenzen für SEU-Graphen benötigen $\sum_{i \in X} |S_i| - \mu$ *Additionen.*

$\mu =$ # Jacobimatrixeinträge $\neq 0$

Beweis.

- $\partial F_j / \partial x_i \neq 0$: # (Pfade $i \rightarrow j$) -1 Additionen

Minimale Kosten – Additionen

Lemma

Alle Eliminationssequenzen für SEU-Graphen benötigen $\sum_{i \in X} |S_i| - \mu$ *Additionen.*

$\mu =$ # Jacobimatrixeinträge $\neq 0$

Beweis.

- $\partial F_j / \partial x_i \neq 0$: # (Pfade $i \to j$) $- 1$ Additionen
- Eine Summe kommt in zwei Matrixeinträgen vor:

$$s = \prod_{(i_1,j_1) \in (i \to j)_1} c_{j_1,i_1} + \prod_{(i_2,j_2) \in (i \to j)_2} c_{j_2,i_2}$$

$\Longrightarrow i \in X$

Minimale Kosten – Additionen

Lemma

Alle Eliminationssequenzen für SEU-Graphen benötigen $\sum_{i \in X} |S_i| - \mu$ *Additionen.*

$\mu =$ # Jacobimatrixeinträge $\neq 0$

Beweis.

- $\partial F_j / \partial x_i \neq 0$: # (Pfade $i \to j$) $- 1$ Additionen
- Eine Summe kommt in zwei Matrixeinträgen vor:

$$s = \prod_{(i_1,j_1) \in (i \to j)_1} c_{j_1,i_1} + \prod_{(i_2,j_2) \in (i \to j)_2} c_{j_2,i_2}$$

$\Longrightarrow i \in X$

- j oder $k \prec^* j$ hat zwei Nachfolger. $\lightning$

Minimale Kosten – Multiplikationen

Lemma

Die Elimination von v_j in G durch Kanteneliminination kostet minimal $|\underline{P}_j|\,|\underline{S}_j|$ Multiplikationen.

Minimale Kosten – Multiplikationen

Lemma

Die Elimination von v_j in G durch Kanteneliminination kostet minimal $|\underline{P}_j|\,|\underline{S}_j|$ Multiplikationen.

Beweis.

- $|\underline{P}_j| =$ Anzahl disjunkter Pfade $X \to v_j$
- $|\underline{S}_j| =$ Anzahl disjunkter Pfade $v_j \to Y$

□

Minimale Kosten – Multiplikationen

Lemma

Die Elimination von v_j in G durch Kanteneliminination kostet minimal $|\underline{P}_j|\,|\underline{S}_j|$ Multiplikationen.

Beweis.

- $|\underline{P}_j| =$ Anzahl disjunkter Pfade $X \to v_j$
- $|\underline{S}_j| =$ Anzahl disjunkter Pfade $v_j \to Y$

□

Folgerung

Die Transformation $G \to G'$ kostet minimal $\sum_{j \in Z} |\underline{P}_j|\,|\underline{S}_j|$ Multiplikationen.

Rückwärtselimination

Lemma

Für SEU-Graphen mit minimalem Innengrad, d. h.

$$\forall j \in Z : |P_j| = |\underline{P}_j|$$

ist die Rückwärtselimination optimal:

Eliminiere j vor i, falls $i \prec^* j$.

Rückwärtselimination

Lemma

Für SEU-Graphen mit minimalem Innengrad, d. h.

$$\forall j \in Z : |P_j| = |\underline{P}_j|$$

ist die Rückwärtselimination optimal:

Eliminiere j vor i, falls $i \prec^* j$.

Beweis.

Rückwärtselimination läßt $|P_i|$ und $|S_i|$ der Vorgänger konstant. □

Optimale Vertexelimination

Theorem

Der folgende Algorithmus ist optimal für SEU-Graphen:

for $j = 1, \ldots p$ **do**
  if $|\underline{P}_j| < |P_j|$ **then**
    eliminiere alle $\underline{P}_j \prec^* v_k \prec^* v_j$ *rückwärts;*
  end
end
Rückwärtselimination;

Optimale Vertexelimination

Theorem

Der folgende Algorithmus ist optimal für SEU-Graphen:

for $j = 1, \ldots p$ **do**
  if $|\underline{P}_j| < |P_j|$ **then**
    eliminiere alle $\underline{P}_j \prec^* v_k \prec^* v_j$ *rückwärts;*
  end
end
Rückwärtselimination;

Beweis.

Optimale Vertexelimination

Theorem

Der folgende Algorithmus ist optimal für SEU-Graphen:

for $j = 1, \ldots p$ **do**
  if $|\underline{P}_j| < |P_j|$ **then**
    eliminiere alle $\underline{P}_j \prec^* v_k \prec^* v_j$ *rückwärts;*
  end
end
Rückwärtselimination;

Beweis.

- Rückwärtselimination für X-j ist optimal.

Optimale Vertexelimination

Theorem

Der folgende Algorithmus ist optimal für SEU-Graphen:

for $j = 1, \ldots p$ **do**
  if $|\underline{P}_j| < |P_j|$ **then**
    eliminiere alle $\underline{P}_j \prec^* v_k \prec^* v_j$ *rückwärts;*
  end
end
Rückwärtselimination;

Beweis.

- Rückwärtselimination für X-j ist optimal.
- $\Longrightarrow |\underline{P}_j| = |P_j|$

Optimale Vertexelimination

Theorem

Der folgende Algorithmus ist optimal für SEU-Graphen:

for $j = 1, \ldots p$ **do**
  if $|\underline{P}_j| < |P_j|$ **then**
    eliminiere alle $\underline{P}_j \prec^* v_k \prec^* v_j$ *rückwärts;*
  end
end
Rückwärtselimination;

Beweis.

- Rückwärtselimination für X-j ist optimal.
- $\Longrightarrow |\underline{P}_j| = |P_j| \forall j$

Optimale Vertexelimination

Theorem

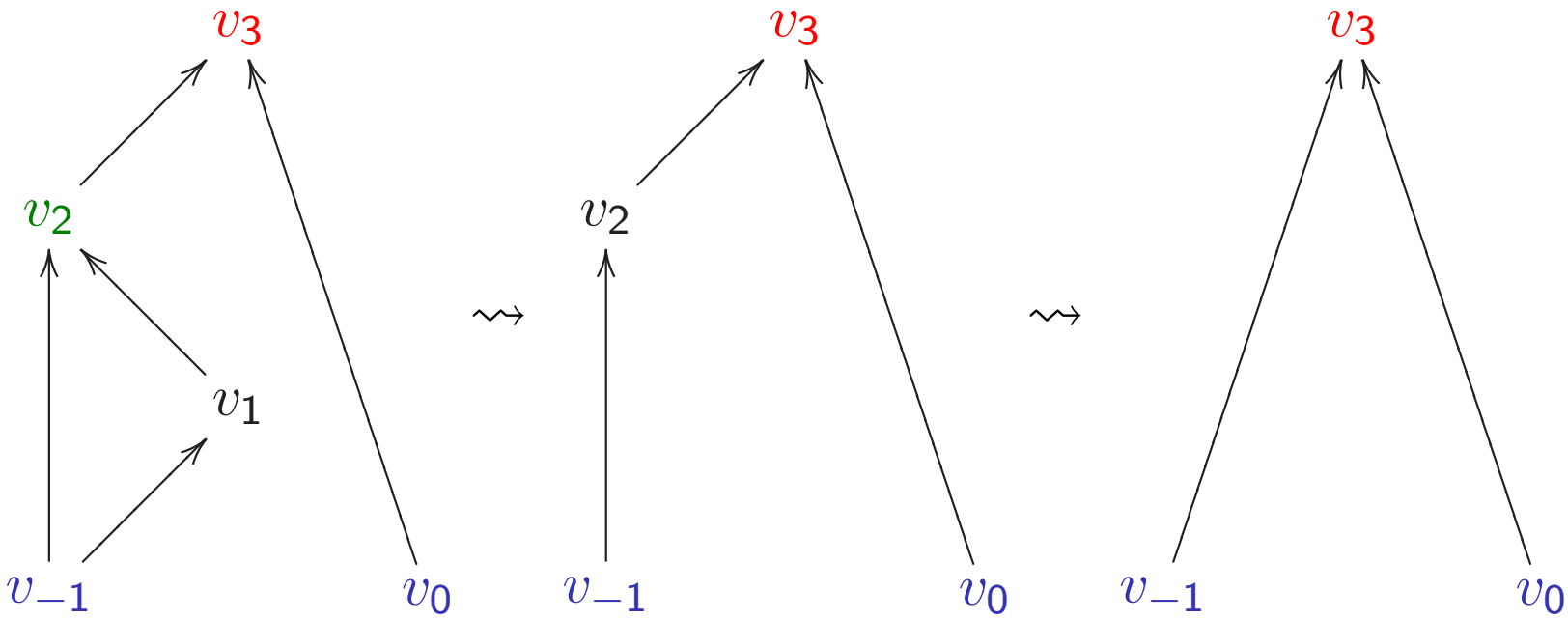
Der folgende Algorithmus ist optimal für SEU-Graphen:

for $j = 1, \ldots p$ **do**
  if $|\underline{P}_j| < |P_j|$ **then**
    eliminiere alle $\underline{P}_j \prec^* v_k \prec^* v_j$ *rückwärts;*
  end
end
Rückwärtselimination;

Beweis.

- Rückwärtselimination für X-j ist optimal.
- $\Longrightarrow |\underline{P}_j| = |P_j| \forall j$
- Rückwärtselimination ist optimal.

Beispiel

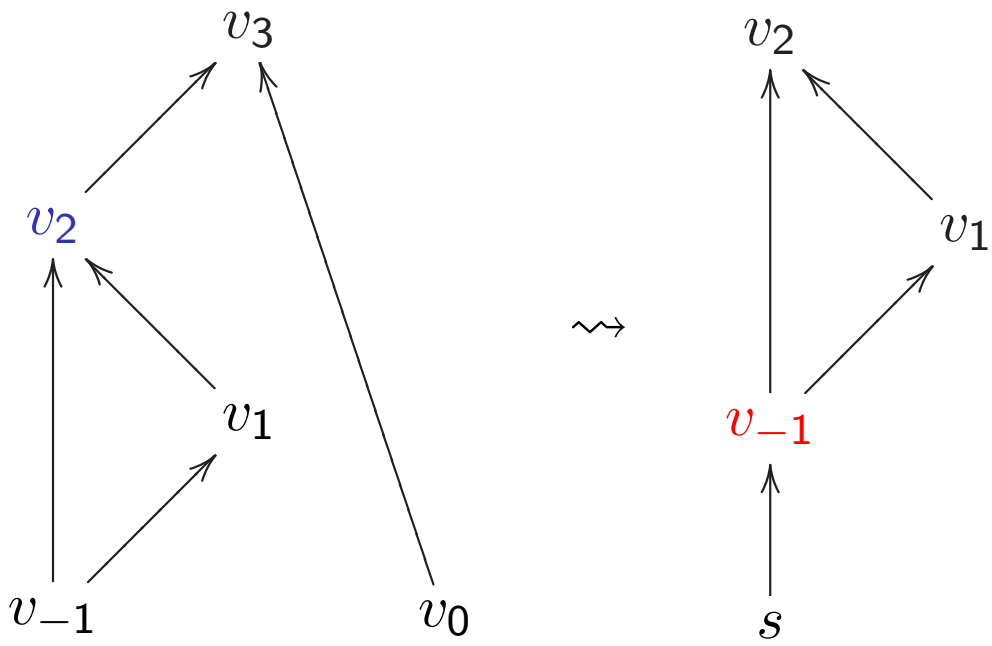


Minimale Schnitte

Berechne $|\underline{P}_j|$ und ein $\underline{P}_j$:

- Betrachte Netzwerke:

$$\begin{aligned} \dot{V}_j &= V_{\prec^* j} \cup \{s\} \\ \dot{E}_j &= E_{\prec^* j} \cup \{(s, x_i)\, \forall x_i \in \dot{V}_j\} \\ c &= 1 \quad \text{Kapazität} \end{aligned}$$



Minimale Schnitte

Berechne $|\underline{P}_j|$ und ein $\underline{P}_j$:

- Betrachte Netzwerke:

$$\begin{aligned} \dot{V}_j &= V_{\prec^* j} \cup \{s\} \\ \dot{E}_j &= E_{\prec^* j} \cup \{(s, x_i) \forall x_i \in \dot{V}_j\} \\ c &= 1 \quad \text{Kapazität} \end{aligned}$$

- minimaler Schnitt $\hat{=}$ maximaler Fluß $\bar{\Phi}$
- Ford-Fulkerson: $O(|\dot{E}_j| \cdot \bar{\Phi})$

Minimale Schnitte

Berechne $|\underline{P}_j|$ und ein $\underline{P}_j$:

- Betrachte Netzwerke:

$$\begin{aligned} \dot{V}_j &= V_{\prec^* j} \cup \{s\} \\ \dot{E}_j &= E_{\prec^* j} \cup \{(s, x_i) \forall x_i \in \dot{V}_j\} \\ c &= 1 \quad \text{Kapazität} \end{aligned}$$

- minimaler Schnitt $\hat{=}$ maximaler Fluß $\bar{\Phi}$
- Ford-Fulkerson: $O(|\dot{E}_j| \cdot \bar{\Phi})$
- Verbesserung: Edmonds & Karp: $O(|\dot{E}_j|\,|\dot{V}_j|^2)$

Minimale Schnitte

Berechne $|\underline{P}_j|$ und ein $\underline{P}_j$:

- Betrachte Netzwerke:

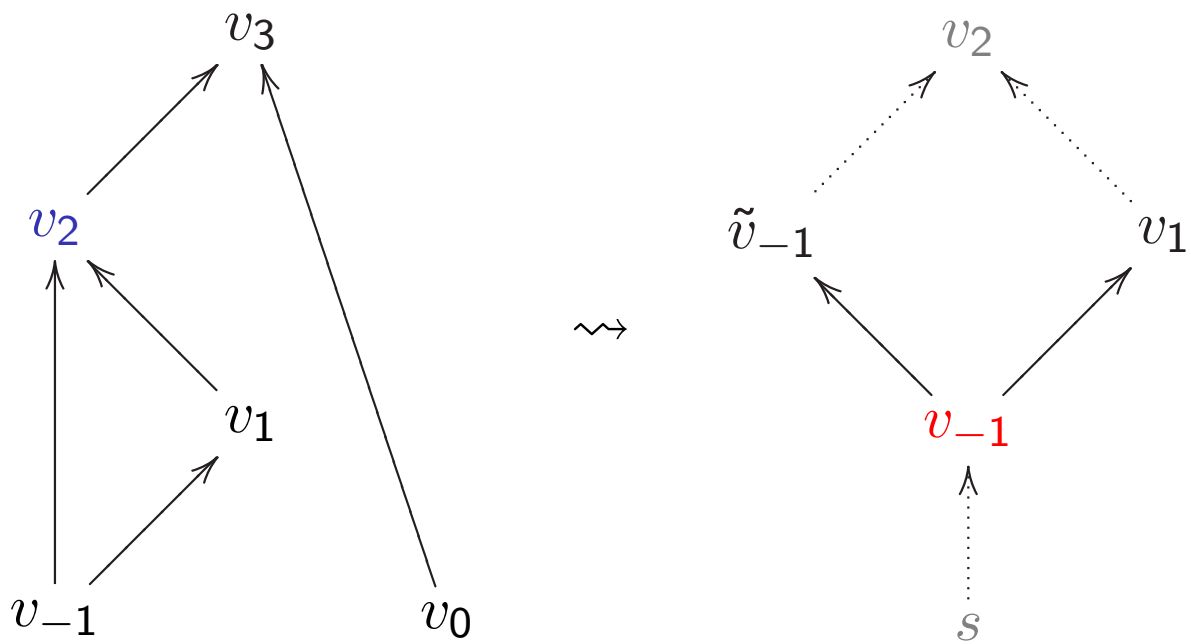
$$\begin{aligned} \dot{V}_j &= V_{\prec^* j} \cup \{s\} \\ \dot{E}_j &= E_{\prec^* j} \cup \{(s, x_i) \forall x_i \in \dot{V}_j\} \\ c &= 1 \quad \text{Kapazität} \end{aligned}$$

- minimaler Schnitt $\hat{=}$ maximaler Fluß $\bar{\Phi}$
- Ford-Fulkerson: $O(|\dot{E}_j| \cdot \bar{\Phi})$
- Verbesserung: Edmonds & Karp: $O(|\dot{E}_j|\,|\dot{V}_j|^2)$
- SEU-Graphen: $\bar{\Phi} \leq |P_j| \leq |\dot{V}_j|$
- $\Longrightarrow$ Ford-Fulkerson: $O(|\dot{E}_j||\dot{V}_j|)$

Minimale Schnitte II

- Es gibt immer minimale X-j-Schnitte $C_v \subseteq X \cup P_j$
- Betrachte bipartiten Graphen:

$$
\begin{aligned}
V_j^b &= P_j \cup \tilde{P}_j \qquad \tilde{P}_j = \widetilde{P_j \cap X_j} \\
E_j^b &= \{(i,k) | \, i \prec^* k\} \cup \{(x_i, \tilde{x}_i) | x_i \in P_j \cap X_j\}
\end{aligned}
$$

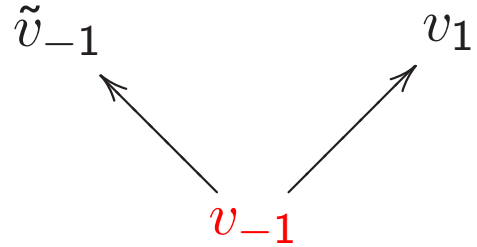


Minimale Schnitte II

- Es gibt immer minimale X-j-Schnitte $C_v \subseteq X \cup P_j$
- Betrachte bipartiten Graphen:

$$\begin{aligned} V_j^b &= P_j \cup \tilde{P}_j \qquad \tilde{P}_j = \widetilde{P_j \cap X_j} \\ E_j^b &= \{(i,k) | \, i \prec^* k\} \cup \{(x_i, \tilde{x}_i) | x_i \in P_j \cap X_j\} \end{aligned}$$

- C_v Schnitt $\Longleftrightarrow$ alle Kanten sind mit C_v verbunden



Minimale Schnitte II

- Es gibt immer minimale X-j-Schnitte $C_v \subseteq X \cup P_j$
- Betrachte bipartiten Graphen:

$$\begin{aligned} V_j^b &= P_j \cup \tilde{P}_j \qquad \tilde{P}_j = \widetilde{P_j \cap X_j} \\ E_j^b &= \{(i,k) | \, i \prec^* k\} \cup \{(x_i, \tilde{x}_i) | x_i \in P_j \cap X_j\} \end{aligned}$$

- C_v Schnitt $\Longleftrightarrow$ alle Kanten sind mit C_v verbunden
- minimaler Schnitt $\hat{=}$ minimale Überdeckung

Minimale Schnitte II

- Es gibt immer minimale X-j-Schnitte $C_v \subseteq X \cup P_j$
- Betrachte bipartiten Graphen:

$$\begin{aligned} V_j^b &= P_j \cup \tilde{P}_j \qquad \tilde{P}_j = \widetilde{P_j \cap X_j} \\ E_j^b &= \{(i,k) | \, i \prec^* k\} \cup \{(x_i, \tilde{x}_i) | x_i \in P_j \cap X_j\} \end{aligned}$$

- C_v Schnitt $\Longleftrightarrow$ alle Kanten sind mit C_v verbunden
- minimaler Schnitt $\hat{=}$ minimale Überdeckung
- König:
  minimale Überdeckung $\hat{=}$ maximales Matching

Minimale Schnitte II

- Es gibt immer minimale X-j-Schnitte $C_v \subseteq X \cup P_j$
- Betrachte bipartiten Graphen:

$$\begin{aligned} V_j^b &= P_j \cup \tilde{P}_j \qquad \tilde{P}_j = \widetilde{P_j \cap X_j} \\ E_j^b &= \{(i,k) | \, i \prec^* k\} \cup \{(x_i, \tilde{x}_i) | x_i \in P_j \cap X_j\} \end{aligned}$$

- C_v Schnitt $\Longleftrightarrow$ alle Kanten sind mit C_v verbunden
- minimaler Schnitt $\hat{=}$ minimale Überdeckung
- König:
  minimale Überdeckung $\hat{=}$ maximales Matching
- Hopcroft & Karp: $O(\sqrt{|V_j^b|}\,|E_j^b|)$

Implementation

- Graphengenerator:

```
>./seu_gen 30 3 1 myGraph.dot
```

Implementation

- Graphengenerator:

```
>./seu_gen 30 3 1 myGraph.dot
```

- Optimale Elimination:

```
>./seu_elim -m myGraph.dot
Number of Multiplications: 69
Number of Additions:       43
```

Implementation

- Graphengenerator:

```
>./seu_gen 30 3 1 myGraph.dot
```

- Optimale Elimination:

```
>./seu_elim -m myGraph.dot
Number of Multiplications: 69
Number of Additions:       43
```

- Vorwärtselimination:

```
>./seu_elim -f myGraph.dot
Number of Multiplications: 76
Number of Additions:       43
```

Implementation

- Graphengenerator:

```
>./seu_gen 30 3 1 myGraph.dot
```

- Optimale Elimination:

```
>./seu_elim -m myGraph.dot
Number of Multiplications: 69
Number of Additions:       43
```

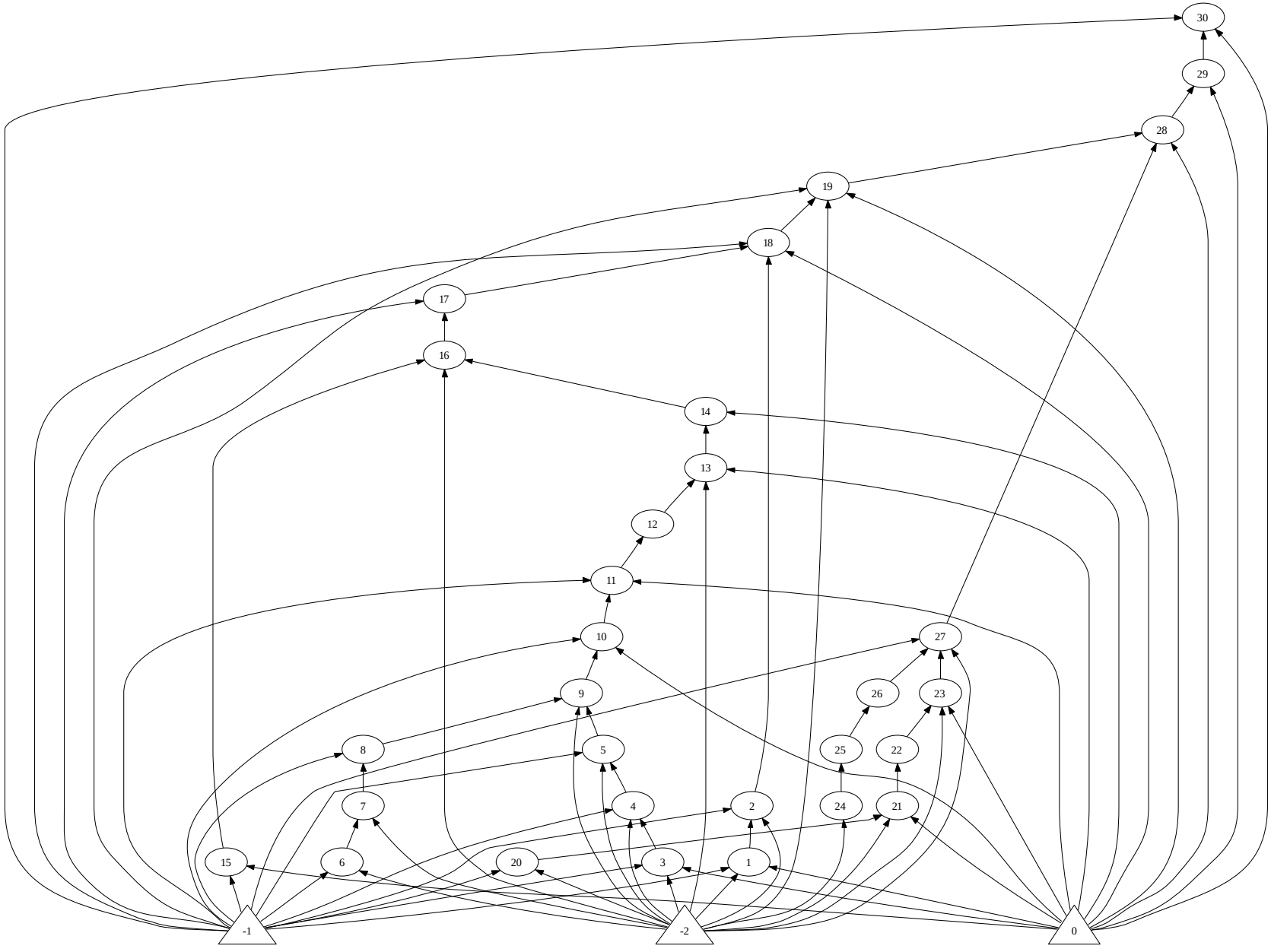
- Vorwärtselimination:

```
>./seu_elim -f myGraph.dot
Number of Multiplications: 76
Number of Additions:       43
```

- Rückwärtselimination:

```
>./seu_elim -r myGraph.dot
Number of Multiplications: 72
Number of Additions:       43
```

Der Graph


30
29
28
19
18
17
16
14
13
12
11
10
27
9
26
23
8
5
25
22
7
4
2
24
21
15
6
20
3
1
-1
-2
0

Ende

Danke für die Aufmerksamkeit!