Calculation of Inside and Outside Weights of Weighted Hypergraphs by Newton's Method

Tobias Denkinger
tobias.denkinger@mailbox.tu-dresden.de

Motivation

- Calculation of the corpus likelihood.
- Inside weight is needed in the E-step of the EM-algorithm
- Normalization, transformation of pCFGs [2]

Motivation

Task

Preliminaries

- Hypergraphs
- Inside and outside weights

Approximation

- Fixed-point iteration
- Newton's method
- Decomposed Newton's method

Performance

Task

- Calculation of **Inside and Outside weights** in weighted hypergraphs

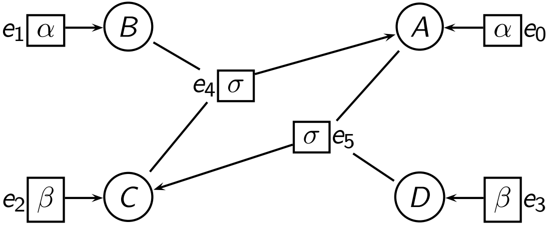
Task

- Calculation of **Inside and Outside weights** in weighted hypergraphs
- by **Newton's method**

Task

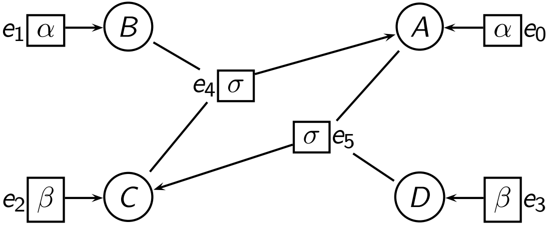
- Calculation of **Inside and Outside weights** in weighted hypergraphs
- by **Newton's method**
- embedded in **Vanda**

Hypergraphs



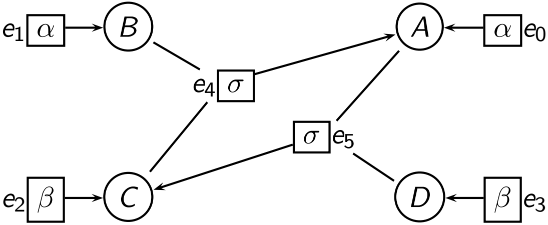
- weighted Σ-hypergraph $H = (V, E, \mu)$
- vertices V
- edges $E \subseteq V^* \times \Sigma \times V$
- weights $\mu \colon E \to \mathbb{R}_{\geq 0}$

Hypergraphs

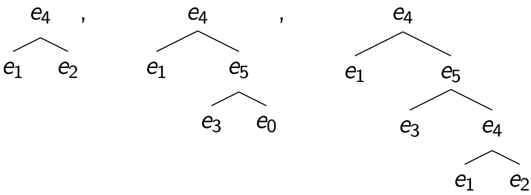


hyperpath **tree** of **edges**, connected according to the hypergraph, leaves have a *rank* of 0

Hypergraphs

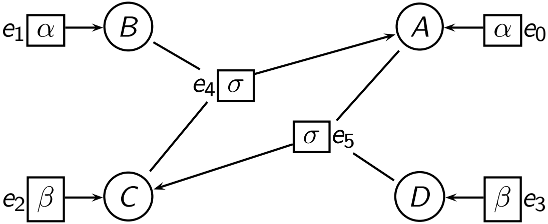


hyperpath **tree** of **edges**, connected according to the hypergraph, leaves have a *rank* of 0



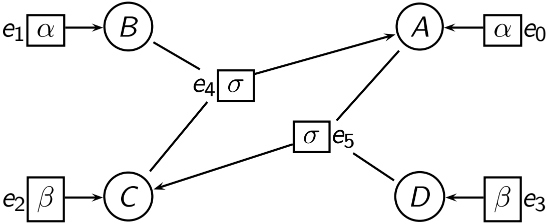
$\in \mathrm{HP}(H, A)$.

Hypergraphs

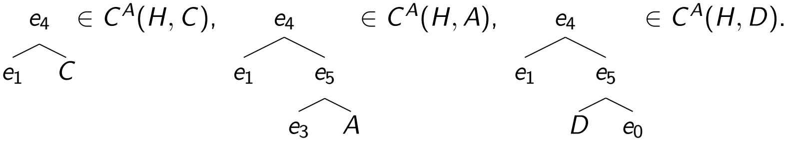


context **tree** of **edges** and **one vertex**, connected according to the hypergraph, leaves have a *rank* of 0, **exactly** one leave is a vertex

Hypergraphs



context **tree** of **edges** and **one vertex**, connected according to the hypergraph, leaves have a *rank* of 0, **exactly** one leave is a vertex



Inside and outside weights

inside weight for a vertex v: sum of the weights of all hyperpaths in H leading to v

$$\mathrm{inside}(v) = \sum_{t \in \mathrm{HP}(H,v)} \mathrm{wt}(t)$$

Inside and outside weights

inside weight for a vertex v: sum of the weights of all hyperpaths in H leading to v

$$\text{inside}(v) = \sum_{t \in \text{HP}(H,v)} \text{wt}(t)$$

outside weight for a vertex v' and a target vertex v: sum of the weights of all v-rooted v'-contexts in H

$$\text{outside}^{v}(v') = \sum_{c \in \text{C}^{v}(H,v')} \text{wt}(c)$$

Fixed-point iteration

- Recursive system

$$\text{inner}(v) = \sum_{\substack{e \in E \\ e=(w,\sigma,v)}} \mu(e) \cdot \prod_{i \in [|w|]} \text{inner}(w_i)$$

Fixed-point iteration

- Recursive system

$$\text{inner}(v) = \sum_{\substack{e \in E \\ e=(w,\sigma,v)}} \mu(e) \cdot \prod_{i \in [|w|]} \text{inner}(w_i)$$

$$\text{outer}^{v'}(v) = s(v) + \sum_{\substack{e \in E \\ e=(w,\sigma,\hat{v}) \\ i \in \mathbb{N}:\ w_i = v}} \text{outer}^{v'}(\hat{v}) \cdot \mu(e) \cdot \prod_{\substack{j \in [|w|] \\ j \neq i}} \text{inner}(w_j)$$

$$s(v) = \begin{cases} 1 & \text{if } v = v' \\ 0 & \text{otherwise} \end{cases}$$

Newton's method

- inside and outside weights as root

$$0 = -i_v + \sum_{\substack{e \in E \\ e=(w,\sigma,v)}} \mu(e) \cdot \prod_{i \in [|w|]} i_{w_i}$$

- uses multivariate Newton's method

Newton's method

- inside and outside weights as root

$$0 = -i_v + \sum_{\substack{e \in E \\ e = (w, \sigma, v)}} \mu(e) \cdot \prod_{i \in [|w|]} i_{w_i}$$

$$0 = -o_v + s(v) + \sum_{\substack{e \in E \\ e = (w, \sigma, \hat{v}) \\ i \in \mathbb{N}:\ w_i = v}} o_{\hat{v}} \cdot \mu(e) \cdot \prod_{\substack{j \in [|w|] \\ j \neq i}} \mathrm{inner}(w_j)$$

- uses multivariate Newton's method

Univariate Newton's method (1 unknown)

Input a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$

Univariate Newton's method (1 unknown)

Input a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$

Output a root of f

Univariate Newton's method (1 unknown)

Input	a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$
Output	a root of f
For every	$n \in \{0, 1, ...\}$

Univariate Newton's method (1 unknown)

Input a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$

Output a root of f

For every $n \in \{0, 1, \dots\}$

1. Construct a **tangent** t at x_n to f.

Univariate Newton's method (1 unknown)

Input a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$

Output a root of f

For every $n \in \{0, 1, ...\}$

1. Construct a **tangent** t at x_n to f.
2. Calculate the **root** of t.

Univariate Newton's method (1 unknown)

Input a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$

Output a root of f

For every $n \in \{0, 1, ...\}$

1. Construct a **tangent** t at x_n to f.
2. Calculate the **root** of t.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Univariate Newton's method (1 unknown)

Input a function $f : \mathbb{R} \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}$

Output a root of f

For every $n \in \{0, 1, ...\}$

1. Construct a **tangent** t at x_n to f.
2. Calculate the **root** of t.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3. If $x_{n+1} = x_n$, output x_n.

Multivariate Newton's method (m unknowns) [1]

Input a function $f : \mathbb{R}^m \to \mathbb{R}^m$, a starting value $x_0 \in \mathbb{R}^m$

Multivariate Newton's method (m unknowns) [1]

Input a function $f\colon \mathbb{R}^m \to \mathbb{R}^m$, a starting value $x_0 \in \mathbb{R}^m$

Output a root of f

Multivariate Newton's method (m unknowns) [1]

Input a function $f : \mathbb{R}^m \to \mathbb{R}^m$, a starting value $x_0 \in \mathbb{R}^m$

Output a root of f

- **Jacobian matrix** corresponds to the derivative

Multivariate Newton's method (m unknowns) [1]

Input a function $f : \mathbb{R}^m \to \mathbb{R}^m$, a starting value $x_0 \in \mathbb{R}^m$

Output a root of f

- **Jacobian matrix** corresponds to the derivative
- multiplication with **inverse Jacobian matrix** replaces division

Multivariate Newton's method (m unknowns) [1]

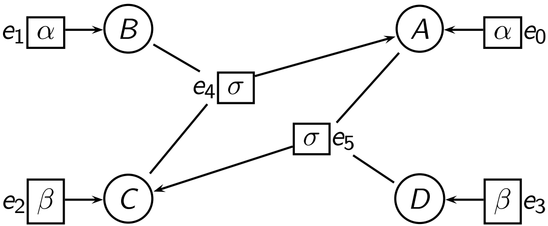
Input a function $f : \mathbb{R}^m \to \mathbb{R}^m$, a starting value $x_0 \in \mathbb{R}^m$

Output a root of f

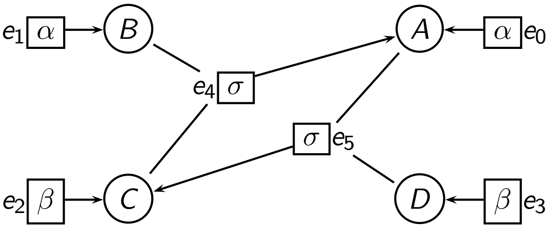
- **Jacobian matrix** corresponds to the derivative
- multiplication with **inverse Jacobian matrix** replaces division

$$x_{n+1} = x_n - (Jf(x_n))^{-1} \cdot f(x_n)$$

Inside weights as root of a function

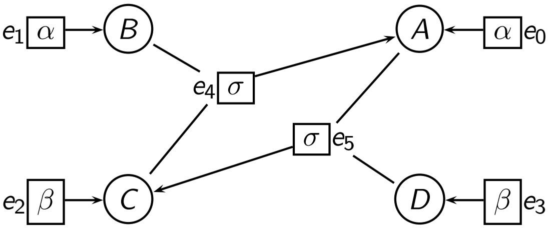


Inside weights as root of a function



$$in'(i)_v = -i_v + \sum_{\substack{e \in E \\ e=(w,\sigma,v)}} \mu(e) \cdot \prod_{i \in [|w|]} i_{w_i}$$

Inside weights as root of a function



$$in'(i)_v = -i_v + \sum_{\substack{e \in E \\ e=(w,\sigma,v)}} \mu(e) \cdot \prod_{i \in [|w|]} i_{w_i}$$

$$in'(i) = \begin{pmatrix} -i_A + \mu(e_0) + \mu(e_4) \cdot i_C \cdot i_B \\ -i_B + \mu(e_1) \\ -i_C + \mu(e_2) + \mu(e_5) \cdot i_D \cdot i_A \\ -i_D + \mu(e_3) \end{pmatrix}$$

Decomposed Newton's method

Idea Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Decomposed Newton's method

Idea Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps

Decomposed Newton's method

Idea: Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps

1. Find SCCs of the hypergraph.

Decomposed Newton's method

Idea: Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps

1. Find SCCs of the hypergraph.
2. Sort them topologically.

Decomposed Newton's method

Idea Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps

1. Find SCCs of the hypergraph.
2. Sort them topologically.
3. Intersect the hypergraph with the SCCs.

Decomposed Newton's method

Idea Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps

1. Find SCCs of the hypergraph.
2. Sort them topologically.
3. Intersect the hypergraph with the SCCs.
4. (Compute polynomials.)

Decomposed Newton's method

Idea: Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps:

1. Find SCCs of the hypergraph.
2. Sort them topologically.
3. Intersect the hypergraph with the SCCs.
4. (Compute polynomials.)
5. Substitute known inside weights in the polynomials with values.

Decomposed Newton's method

Idea Decompose hypergraph into **strongly connected components**, then apply **Newton's method** to each component, memorize and **reuse** already computed inside weights.

Steps

1. Find SCCs of the hypergraph.
2. Sort them topologically.
3. Intersect the hypergraph with the SCCs.
4. (Compute polynomials.)
5. Substitute known inside weights in the polynomials with values.
6. (Apply Newton's method.)

Decomposed Newton's method

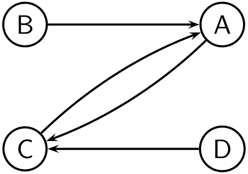
1. Find SCCs of the hypergraph H.

Decomposed Newton's method

1. Find SCCs of the hypergraph H.
   - Construct a graph G s.t. $\mathrm{SCCs}(G) = \mathrm{SCCs}(H)$

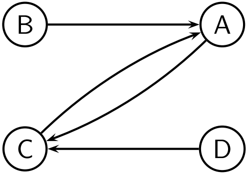
Decomposed Newton's method

1. Find SCCs of the hypergraph H.
   - Construct a graph G s.t. $\text{SCCs}(G) = \text{SCCs}(H)$



Decomposed Newton's method

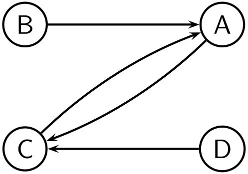
1. Find SCCs of the hypergraph H.
   - Construct a graph G s.t. $\text{SCCs}(G) = \text{SCCs}(H)$



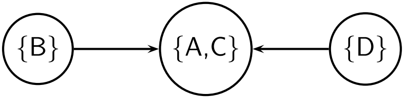
   - Collapse the SCCs [3]

Decomposed Newton's method

1. Find SCCs of the hypergraph H.
   - Construct a graph G s.t. SCCs(G) = SCCs(H)



   - Collapse the SCCs [3]

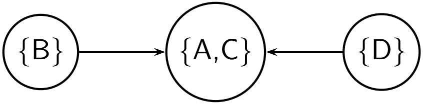


Decomposed Newton's method

2. Sort the SCCs topologically.

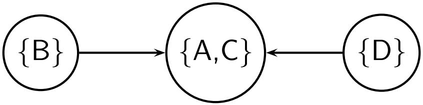
Decomposed Newton's method

2. Sort the SCCs topologically.



Decomposed Newton's method

2. Sort the SCCs topologically.
   SCCs $= \langle \{B\}, \{D\}, \{A, C\} \rangle$

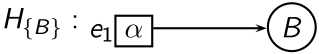


Decomposed Newton's method

2. Sort the SCCs topologically.
   SCCs $= \langle \{B\}, \{D\}, \{A, C\} \rangle$
3. Intersect H with $SCCs$.

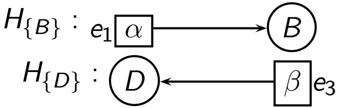
Decomposed Newton's method

2. Sort the SCCs topologically.
   SCCs $= \langle\{B\}, \{D\}, \{A, C\}\rangle$
3. Intersect H with $SCCs$.



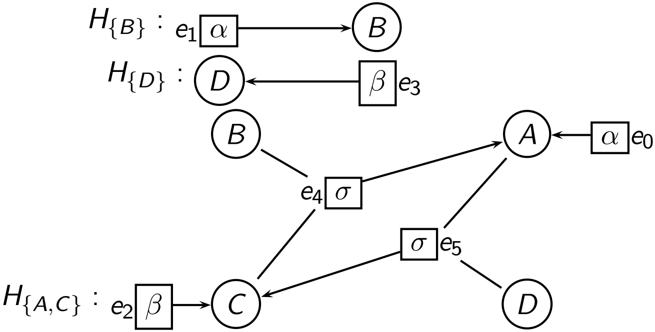
Decomposed Newton's method

2. Sort the SCCs topologically.
   SCCs $= \langle \{B\}, \{D\}, \{A, C\} \rangle$
3. Intersect H with $SCCs$.



Decomposed Newton's method

2. Sort the SCCs topologically.
   SCCs $= \langle \{B\}, \{D\}, \{A, C\} \rangle$
3. Intersect H with $SCCs$.



Decomposed Newton's method

4. (Compute polynomials.)
5. Substitute known inside weights in the polynomials with values.
6. (Apply Newton's method.)

For the hypergraph $H_{\{B\}}$ we get

$$\begin{aligned} in'_{\{B\}} &= \left(-i_B + \mu(e_1)\right) \\ &= \left(1 - i_B\right) \\ (Jin'_{\{B\}})^{-1} &= \left(-1\right). \end{aligned}$$

Then by Newton's method in:

$$i_B = 1.$$

Decomposed Newton's method

4. (Compute polynomials.)
5. Substitute known inside weights in the polynomials with values.
6. (Apply Newton's method.)

For the hypergraph $H_{\{D\}}$ we get

$$\begin{aligned} in'_{\{D\}} &= \left(-i_D + \mu(e_3)\right) \\ &= \left(1 - i_D\right) \\ (Jin'_{\{D\}})^{-1} &= \left(-1\right). \end{aligned}$$

Then by Newton's method:

$$i_D = 1.$$

Decomposed Newton's method

4. (Compute polynomials.)
5. Substitute known inside weights in the polynomials with values.
6. (Apply Newton's method.)

For the hypergraph $H_{\{A,C\}}$ we get

$$in'_{\{A,C\}} = \begin{pmatrix} -i_A + \mu(e_0) + \mu(e_4) \cdot i_C \cdot i_B \\ -i_C + \mu(e_2) + \mu(e_5) \cdot i_D \cdot i_A \end{pmatrix}$$

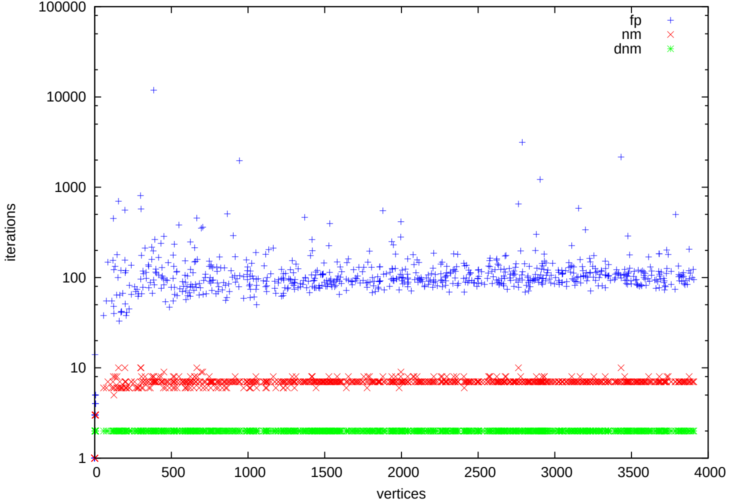
$$= \begin{pmatrix} -i_A + 0.25 + 0.25 \cdot i_C \\ -i_C + 0.25 + 0.25 \cdot i_A \end{pmatrix}$$

$$(Jin'_{\{B\}})^{-1} = -\frac{4}{15} \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}.$$

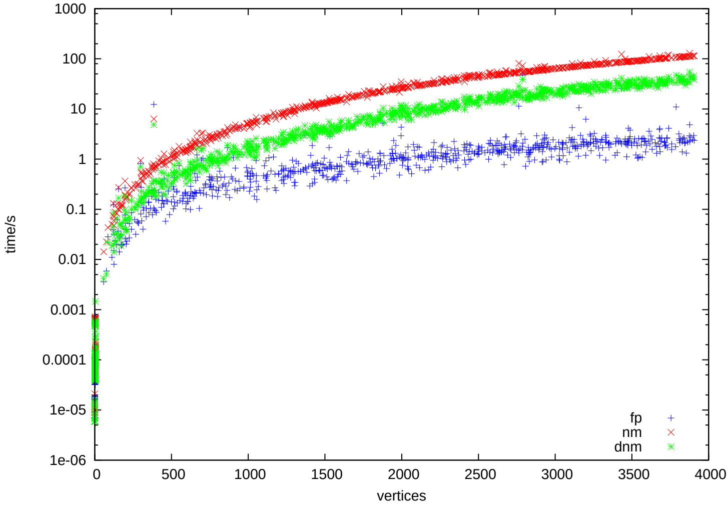
Then by Newton's method:

$$i_A = \frac{1}{3}, i_C = \frac{1}{3}.$$

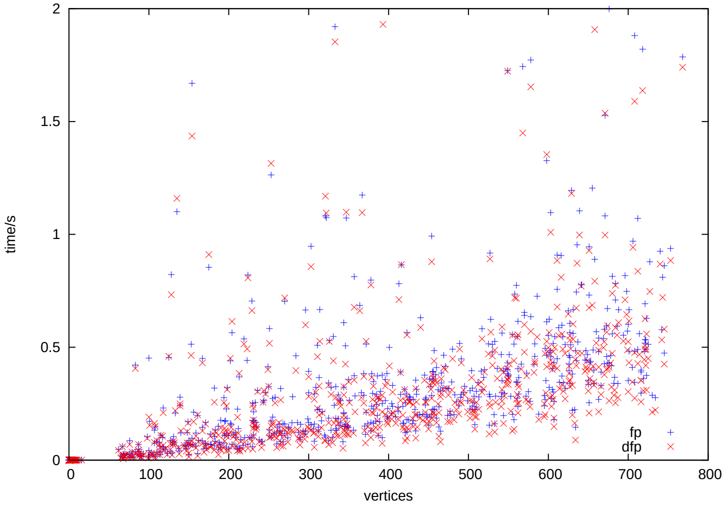
Performance



Performance



Decomposed fixed-point method



The End

Thank you for your attention!

References

- Javier Esparza, Stefan Kiefer, and Michael Luttenberger, *Convergence thresholds of Newton's method for monotone polynomial equations*, Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS) (Bordeaux, France) (Pascal Weil and Susanne Albers, eds.), 2008, http://arxiv.org/abs/0802.2856, pp. 289–300.
- Mark-Jan Nederhof and Giorgio Satta, *Computing partition functions of pcfgs*, Research on Language and Computation **6** (2008), 139–162, 10.1007/s11168-008-9052-8, http://dx.doi.org/10.1007/s11168-008-9052-8.
- Robert Tarjan, *Depth-first search and linear graph algorithms*, SIAM Journal on Computing **1** (1972), no. 2, 146–160, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=45