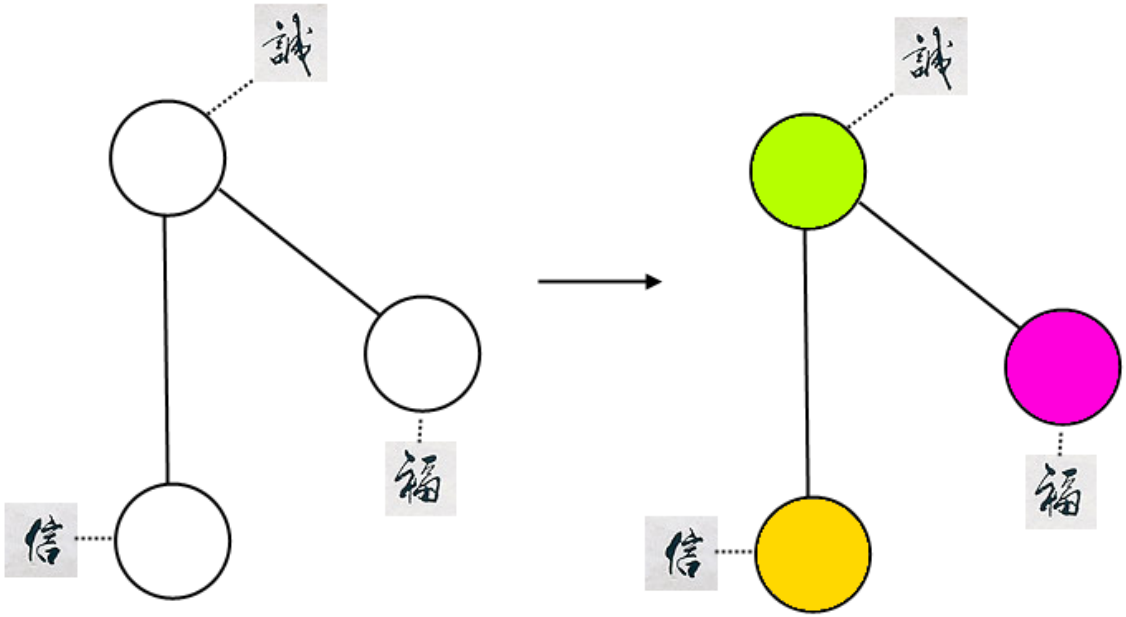
Strukturelle SVM zum Graph-*labelling*

Florian Blümel

23. Juni 2009

1. Problemstellung
   - Was wir gerne hätten . . .
   - . . . und der Weg dorthin
   - Erinnerung: strukturelle SVM
2. Lösungsstrategien
   - Junction Tree–Algorithmus
   - Loopy Belief Propagation
   - Gibbs Sampling
3. Experimente
   - Umfang
   - Qualität der Algorithmen
   - Schlussfolgerungen

Was wir gerne hätten: *labelling* von Graphen



... und der Weg dorthin

Wir konstruieren einen (anderen!) Graphen $G = (V, E)$, wobei

- $X = \{x_1, \dots\}$ die Menge der Beobachtungen,
- $Y = \{y_1, \dots\}$ die Menge der latenten *label*-Variablen und
- $E \subseteq (X \times Y) \cup (Y \times Y)$ ist.

Zwei Knoten sollen verbunden sein, wenn sie direkt voneinander abhängen.
Gesucht ist für die Y-Knoten in G ein *labelling* $y \in \Sigma^{|Y|}$, wobei

- Σ das *label*-Alphabet ist und
- das *labelling* auch verstanden werden kann als Zuordnung $Y \to \Sigma$.

Wir haben einen Unabhängigkeitsgraphen konstruiert!
Wenn wir zu G ein Markov-Feld betrachten, ist die Wahrscheinlichkeit von Y gegeben Beobachtungen x

$$\hat{p}(y|x) = \exp\{\langle \lambda, \Phi(x,y)\rangle - \log g(\lambda|x)\}$$

mit Partitionsfunktion

$$g(\lambda|x) = \sum_{\bar{y}} \exp\{\langle \lambda, \Phi(x,\bar{y})\rangle\} < \infty.$$

Diese Verteilung faktorisiert über den Cliquen von G, und damit auch

$$\Phi(x,y) = \sum_{C \in \mathcal{C}} \phi_C(x_C, y_C).$$

Merkmalsvektoren

Wir verstehen ϕ_C als gemeinsamen Merkmalsvektor für eine Clique.

- ϕ_C trägt die Abhängigkeitsstruktur
- Aber ... wie berechnet man $\phi_C(x_C, y_C)$ bei unbeschränkt großen Cliquen?

Als Ausweg kann man über alle 2-Cliquen faktorisieren und jedes ϕ einer Kante zuordnen. Es gibt nur Kanten zwischen Beobachtungen und *labels*

$$\phi_1(x_i, y_i) = (\delta_{k_1 y_i}, \ldots, \delta_{k_{|\Sigma|} y_i})^\top \otimes \psi(x_i)$$

und zwischen zwei *labels*

$$\phi_2(y_i, y_j) = \begin{pmatrix} \delta_{k_1 y_i} \\ \vdots \\ \delta_{k_{|\Sigma|} y_i} \end{pmatrix} \otimes \begin{pmatrix} \delta_{k_1 y_j} \\ \vdots \\ \delta_{k_{|\Sigma|} y_j} \end{pmatrix}.$$

($k \in \Sigma, \otimes$ Tensorprodukt)

... wohin wollten wir noch gleich?

Nachdem wir jetzt

$$\hat{p}(y|x) = \exp\left\{\langle \lambda, \Phi(x,y)\rangle - \log g(\lambda|x)\right\}$$

definiert haben, ist unsere Lernaufgabe

$$\hat{y} = \arg\max_y \hat{p}(y|x) = \arg\max_y \langle \lambda, \Phi(x,y)\rangle \,.$$

Erinnerung: strukturelle SVM

Definition

Für Trainingsdaten $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{l}$, eine Fehlerfunktion Δ und $\eta > 0$ ist das Problem, eine strukturelle SVM zu optimieren,

$$\min_{\lambda,\xi} \frac{|\lambda|^2}{2} + \eta \langle \xi, \mathbb{1} \rangle$$

unter den Nebenbedingungen

$$\forall i : \left\langle \lambda, \Phi(x^{(i)}, y^{(i)}) \right\rangle \geq \max_{\bar{y} \neq y^{(i)}} \left[\Delta(y^{(i)}, \bar{y}) + \left\langle \lambda, \Phi(x^{(i)}, \bar{y}) \right\rangle \right] - \xi_i$$

Erinnerung: strukturelle SVM

Definition

Für Trainingsdaten $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{l}$, eine Fehlerfunktion Δ und $\eta > 0$ ist das Problem, eine strukturelle SVM zu optimieren,

$$\min_{\lambda,\xi} \frac{|\lambda|^2}{2} + \eta \langle \xi, \mathbb{1} \rangle \leftrightarrow \min_{w,\xi} \left\{ \frac{\|w\|^2}{2} + \frac{C}{n} \sum_{i=1}^{n} \xi_i \right\}$$

unter den Nebenbedingungen

$$\forall i : \left\langle \lambda, \Phi(x^{(i)}, y^{(i)}) \right\rangle \geq \max_{\bar{y} \neq y^{(i)}} \left[\Delta(y^{(i)}, \bar{y}) + \left\langle \lambda, \Phi(x^{(i)}, \bar{y}) \right\rangle \right] - \xi_i$$

Erinnerung: strukturelle SVM

Definition

Für Trainingsdaten $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{l}$, eine Fehlerfunktion Δ und $\eta > 0$ ist das Problem, eine strukturelle SVM zu optimieren,

$$\min_{\lambda,\xi} \frac{|\lambda|^2}{2} + \eta \langle \xi, \mathbb{1} \rangle \leftrightarrow \min_{w,\xi} \left\{ \frac{\|w\|^2}{2} + \frac{C}{n} \sum_{i=1}^{n} \xi_i \right\} \quad (\text{SVM}_1 \quad)$$

unter den Nebenbedingungen

$$\forall i : \left\langle \lambda, \Phi(x^{(i)}, y^{(i)}) \right\rangle \geq \max_{\bar{y} \neq y^{(i)}} \left[\Delta(y^{(i)}, \bar{y}) + \left\langle \lambda, \Phi(x^{(i)}, \bar{y}) \right\rangle \right] - \xi_i$$

Erinnerung: strukturelle SVM

Definition

Für Trainingsdaten $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{l}$, eine Fehlerfunktion Δ und $\eta > 0$ ist das Problem, eine strukturelle SVM zu optimieren,

$$\min_{\lambda,\xi} \frac{|\lambda|^2}{2} + \eta \langle \xi, \mathbb{1} \rangle \leftrightarrow \min_{w,\xi} \left\{ \frac{\|w\|^2}{2} + \frac{C}{n} \sum_{i=1}^{n} \xi_i \right\} \quad (\text{SVM}_1 \quad)$$

unter den Nebenbedingungen

$$\forall i : \left\langle \lambda, \Phi(x^{(i)}, y^{(i)}) \right\rangle \geq \max_{\bar{y} \neq y^{(i)}} \left[\Delta(y^{(i)}, \bar{y}) + \left\langle \lambda, \Phi(x^{(i)}, \bar{y}) \right\rangle \right] - \xi_i$$

$$\leftrightarrow \forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq \Delta(y_i, y) - \xi_i$$

Erinnerung: strukturelle SVM

Definition

Für Trainingsdaten $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{l}$, eine Fehlerfunktion Δ und $\eta > 0$ ist das Problem, eine strukturelle SVM zu optimieren,

$$\min_{\lambda,\xi} \frac{|\lambda|^2}{2} + \eta \langle \xi, \mathbb{1} \rangle \leftrightarrow \min_{w,\xi} \left\{ \frac{\|w\|^2}{2} + \frac{C}{n} \sum_{i=1}^{n} \xi_i \right\} \quad (\text{SVM}_1^{\Delta m})$$

unter den Nebenbedingungen

$$\forall i : \left\langle \lambda, \Phi(x^{(i)}, y^{(i)}) \right\rangle \geq \max_{\bar{y} \neq y^{(i)}} \left[\Delta(y^{(i)}, \bar{y}) + \left\langle \lambda, \Phi(x^{(i)}, \bar{y}) \right\rangle \right] - \xi_i$$

$$\leftrightarrow \forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq \Delta(y_i, y) - \xi_i$$

Lösungsstrategien

- *belief propagation*
  - *junction tree*–Algorithmus
  - *loopy belief propagation*
- *Gibbs sampling*

belief propagation

Idee: Wiederholtes „Verbreiten von Ansichten“

- Jeder Knoten im Graphen schickt seinen Nachbarn Nachrichten, die seine „Ansicht“ weiterreichen.
- Jeder Variablenknoten aktualisiert seinen Wert, wenn er die eingehenden Nachrichten kennt.

belief propagation

Idee: Wiederholtes „Verbreiten von Ansichten“

- Jeder Knoten im Graphen schickt seinen Nachbarn Nachrichten, die seine „Ansicht“ weiterreichen.
- Jeder Variablenknoten aktualisiert seinen Wert, wenn er die eingehenden Nachrichten kennt.

belief propagation . . .

- konvergiert für Bäume, Graphen mit nur einem Kreis, . . .

belief propagation

Idee: Wiederholtes „Verbreiten von Ansichten“

- Jeder Knoten im Graphen schickt seinen Nachbarn Nachrichten, die seine „Ansicht“ weiterreichen.
- Jeder Variablenknoten aktualisiert seinen Wert, wenn er die eingehenden Nachrichten kennt.

belief propagation . . .

- konvergiert für Bäume, Graphen mit nur einem Kreis, . . . aber nicht für beliebige Graphen, und nicht notwendigerweise gegen das globale Optimum

belief propagation

Idee: Wiederholtes „Verbreiten von Ansichten“

- Jeder Knoten im Graphen schickt seinen Nachbarn Nachrichten, die seine „Ansicht“ weiterreichen.
- Jeder Variablenknoten aktualisiert seinen Wert, wenn er die eingehenden Nachrichten kennt.

belief propagation . . .

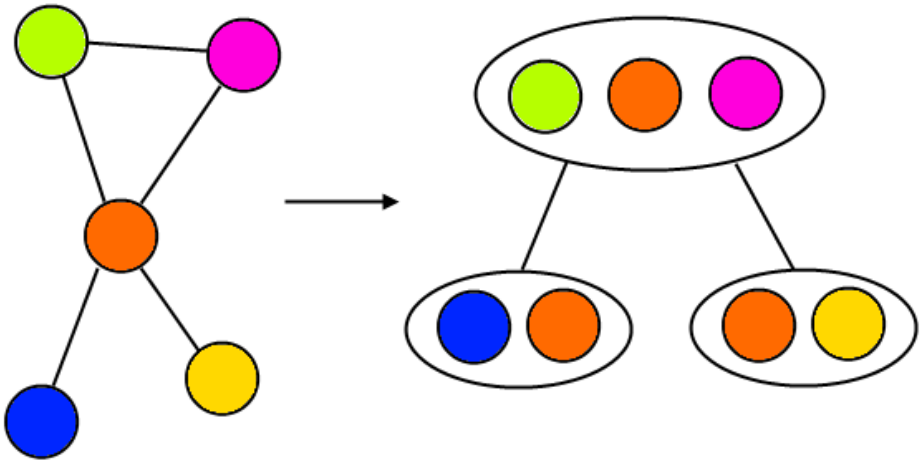
- konvergiert für Bäume, Graphen mit nur einem Kreis, . . . aber nicht für beliebige Graphen, und nicht notwendigerweise gegen das globale Optimum
- liefert für Bäume die exakte Wahrscheinlichkeit für eine Belegung nach 2 Schritten.

junction tree–Algorithmus

Wir wissen

- dass BP für Bäume konvergiert
- dass Φ über den Cliquen faktorisiert

$\longrightarrow$ wir bauen einen Cliquenbaum!



Nachrichten im *junction tree*

1. Jeder Knoten $A \in C$ schickt an alle seine Kinder

$$m_{AB}(y_{A\cap B}) = \arg\max_{y_{A\setminus B}} \langle \lambda, \phi(x_A, y_A) \rangle \, .$$

2. Jeder Knoten, der von allen seinen Kindern Nachrichten erhalten hat, schickt seinem Elter m_{BA}.

Nach diesen zwei Schritten ist der Baum im Gleichgewicht. Das wahrscheinlichste *labelling* kann jetzt als

$$\hat{y} = \arg\max_{y} \sum_{C\in\mathcal{C}} \langle \lambda, \phi_C(x_C, y_C) \rangle - \sum_{(A,B)\in E_J} m_{BA}(y_{A\cap B})$$

berechnet werden.

Dieser Algorithmus hat Komplexität ...

- $\mathcal{O}(\exp |V|)$ für den Aufbau des Cliquenbaums und
- $\mathcal{O}(|\Sigma|^{\max |C|})$ für das Versenden der Nachrichten.

Das ist nicht so toll.

loopy belief propagation

Auf dem ursprünglichen Graphen sollen sich alle Knoten gleichzeitig Nachrichten schicken.

1. Die Nachricht von Knoten i an Knoten j ist

$$m_{ij}(y_j) = \max_{y_i} \left\{ \langle \lambda, \phi_1(x_i, y_i) \rangle + \langle \lambda, \phi_2(y_i, y_j) \rangle + \sum_{j:(i,j)\in E} m_{ji}(y_i) \right\}.$$

2. Nach Zustellung der Nachrichten werden die Werte der Knoten aktualisiert.
3. Graph ist nicht stabil? Zurück zu Schritt 1.

Falls das zu einem stabilen Graphen konvergiert, ist

$$\hat{y}_i = \arg\max_{y_i} \left[\langle \lambda, \phi_1(x_i, y_i) \rangle + \sum_{j:(i,j)\in E} m_{ji}(y_i) \right].$$

Jeder Durchlauf ist in $\mathcal{O}(|E||Y|^2)$.

Falls das zu einem stabilen Graphen konvergiert, ist

$$\hat{y}_i = \arg\max_{y_i} \left[\langle \lambda, \phi_1(x_i, y_i) \rangle + \sum_{j:(i,j)\in E} m_{ji}(y_i) \right].$$

Jeder Durchlauf ist in $\mathcal{O}(|E||Y|^2)$.

- Das ist schon wesentlich besser, aber ...

Falls das zu einem stabilen Graphen konvergiert, ist

$$\hat{y}_i = \arg\max_{y_i} \left[\langle \lambda, \phi_1(x_i, y_i) \rangle + \sum_{j:(i,j)\in E} m_{ji}(y_i) \right].$$

Jeder Durchlauf ist in $\mathcal{O}(|E||Y|^2)$.

- Das ist schon wesentlich besser, aber ...
- was bringt uns ein Verfahren, das nicht konvergiert?

Falls das zu einem stabilen Graphen konvergiert, ist

$$\hat{y}_i = \arg\max_{y_i} \left[\langle \lambda, \phi_1(x_i, y_i) \rangle + \sum_{j:(i,j)\in E} m_{ji}(y_i) \right].$$

Jeder Durchlauf ist in $\mathcal{O}(|E||Y|^2)$.

- Das ist schon wesentlich besser, aber ...
- was bringt uns ein Verfahren, das nicht konvergiert?
- Wenn es konvergiert, wie schnell konvergiert es?

Falls das zu einem stabilen Graphen konvergiert, ist

$$\hat{y}_i = \arg\max_{y_i} \left[\langle \lambda, \phi_1(x_i, y_i) \rangle + \sum_{j:(i,j)\in E} m_{ji}(y_i) \right].$$

Jeder Durchlauf ist in $\mathcal{O}(|E||Y|^2)$.

- Das ist schon wesentlich besser, aber ...
- was bringt uns ein Verfahren, das nicht konvergiert?
- Wenn es konvergiert, wie schnell konvergiert es?
- Das liefert uns nicht notwendigerweise das globale Optimum!

Gibbs sampling

Proposition: *Gibbs sampling rule*

Wählt man wiederholt eine latente Variable y_i und weist ihr mit Hilfe der bedingten Wahrscheinlichkeit $\hat{p}(y_i|x, \{y_j : j \neq i\})$ einen Wert zu, erhält man eine Markov-Kette von Beobachtungen, die gegen die Wahrscheinlichkeit $\hat{p}(y|x)$ konvergiert.

- Wir starten mit einem zufällig gewählten y.

Herleitung von $\hat{p}(y_i|x, \{y_j : j \neq i\})$

$$\hat{p}(y_i|x, \{y_j : j \neq i\}) = \frac{p(Y_i = y_i, x, \{y_j : j \neq i\})}{\sum_{\sigma \in \Sigma} p(Y_i = \sigma|x, \{y_j : j \neq i\})}$$

(Setze $\hat{p}(y)$ ein, setze $\bar{y}_j^\sigma = \sigma$ für $j = i$ und $= y_j$ für $j \neq i$)

$$= \frac{\exp\left\{\sum_{k \in V} \langle \lambda, \phi_1(x_k, y_k)\rangle + \sum_{(k,l) \in E} \langle \lambda, \phi_2(y_k, y_l)\rangle\right\}}{\sum_{\sigma \in \Sigma} \exp\left\{\sum_{k \in V} \langle \lambda, \phi_1(x_k, \bar{y}_k^\sigma)\rangle + \sum_{(k,l) \in E} \langle \lambda, \phi_2(\bar{y}_k^\sigma, \bar{y}_l^\sigma)\rangle\right\}}$$

$$= \frac{\exp\left\{\langle \lambda, \phi_1(x_i, y_i)\rangle + \sum_{(k,i) \in E} \langle \lambda, \phi_2(y_k, y_i)\rangle\right\}}{\sum_{\sigma \in \Sigma} \exp\left\{\langle \lambda, \phi_1(x_i, \sigma)\rangle + \sum_{(k,i) \in E} \langle \lambda, \phi_2(y_k, \sigma)\rangle\right\}}$$

- In der Praxis wählt man y_i nicht zufällig, sondern sequentiell.
- Ein einzelner Durchlauf über alle Variablen ist in $\mathcal{O}(|E||Y|^2)$
- Wieder wissen wir nicht, wie schnell der Prozess konvergiert.

Offene Fragen

Wir kennen jetzt drei verschiedene Ansätze. Aber . . .

- Was ist „zu groß“ für den *junction tree*–Algorithmus?
- Ist *loopy belief propagation* praktisch relevant, das heißt
  - konvergiert es für konkrete Datensätze
  - in annehmbar vielen Durchläufen
  - zum globalen Optimum?
- Wie schnell konvergiert *Gibbs sampling*?

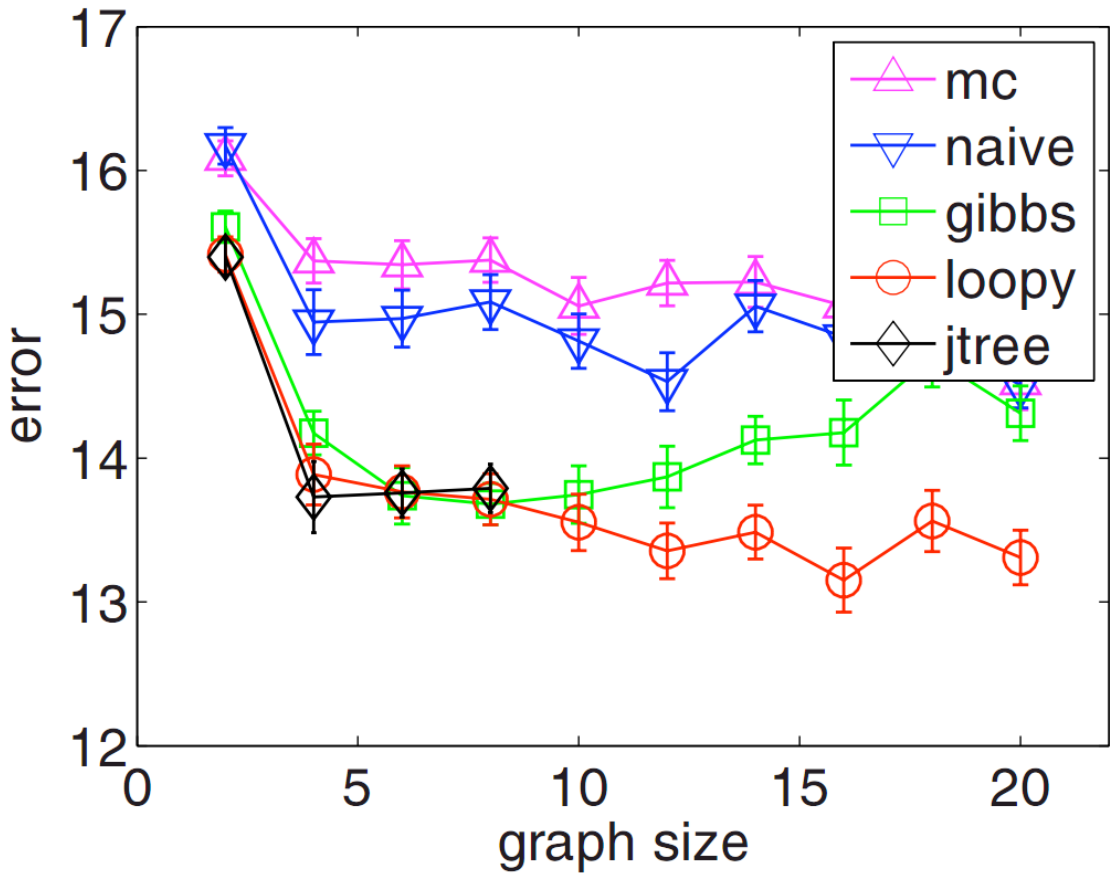
Rahmen für die Versuche

Drei Datensätze,

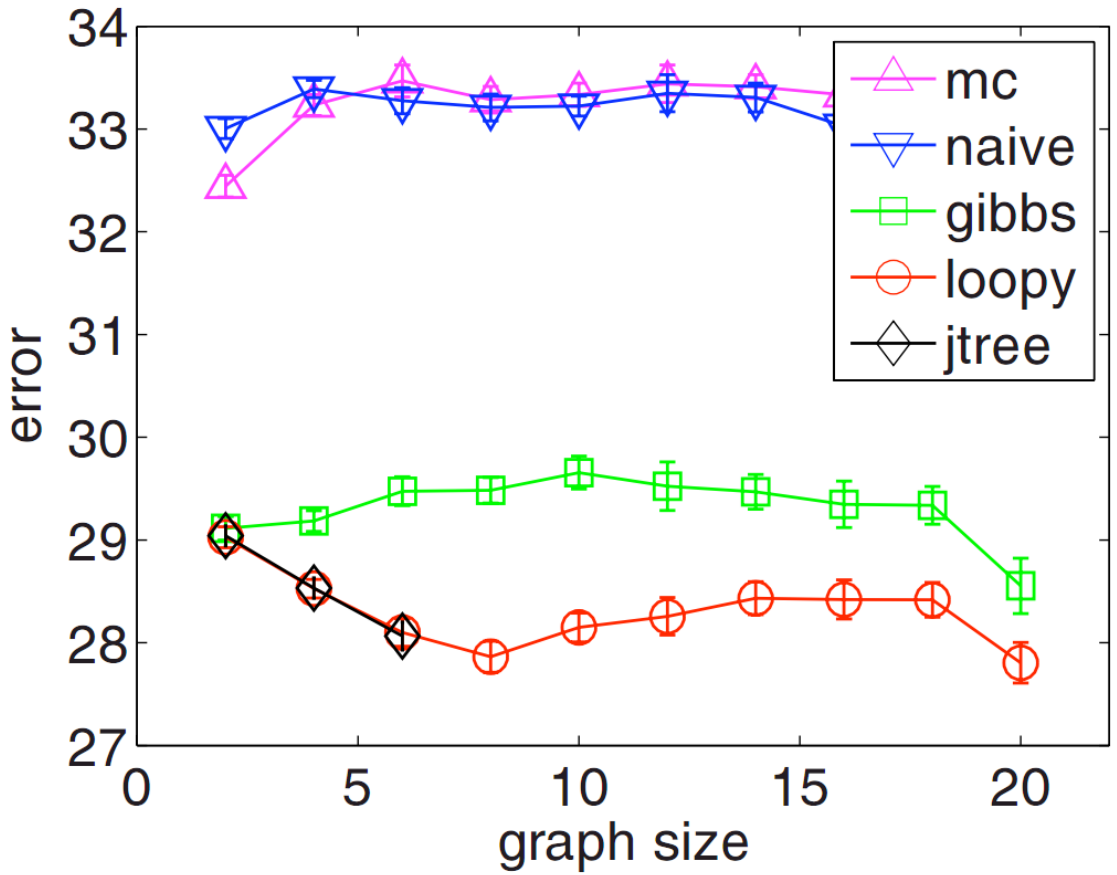
- WebKB: Universitäts-Webseiten
  Struktur durch Links gegeben
- Cora: wissenschaftliche Artikel aus der Informatik
  Struktur durch Zitierung gegeben
- Reuters21578: Artikel aus einem Nachrichtenarchiv

Betrachtet werden Teilgraphen aus den Datensätzen.

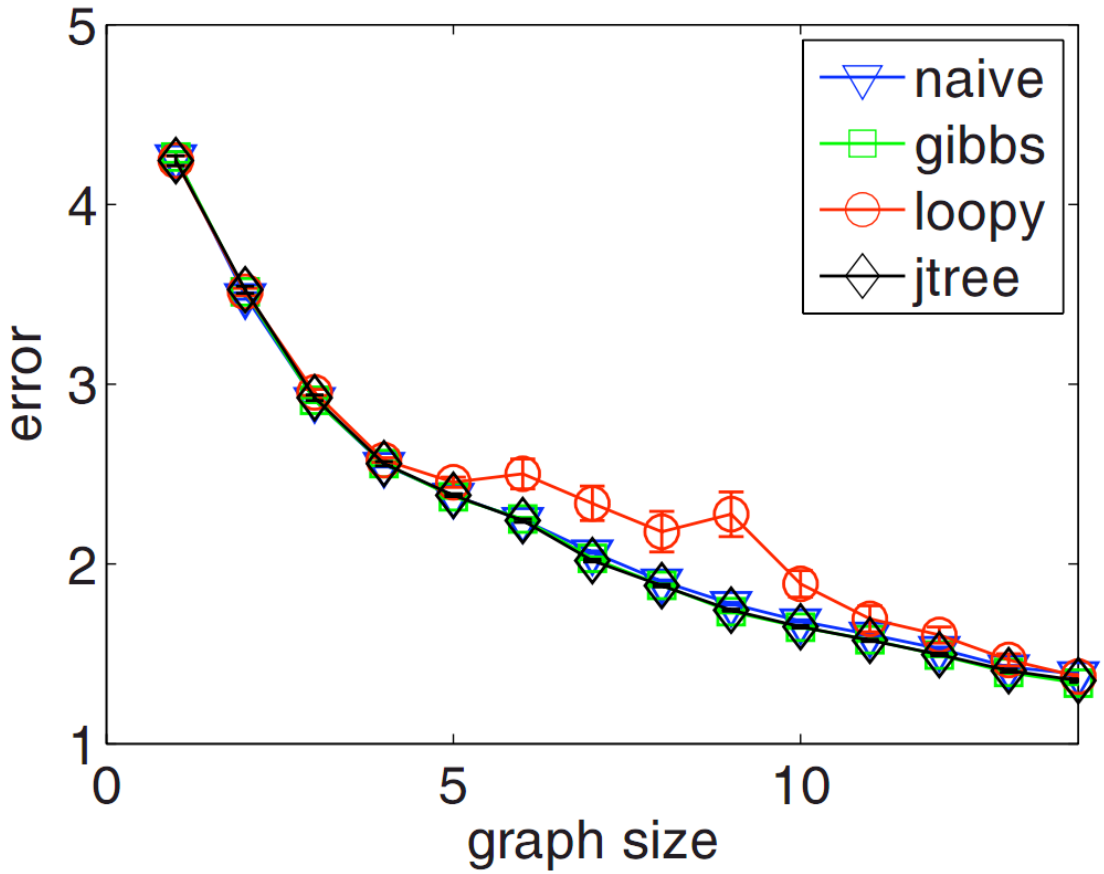
WebKB


17
16
15
14
13
12
error
0
5
10
15
20
graph size
mc
naive
gibbs
loopy
jtree

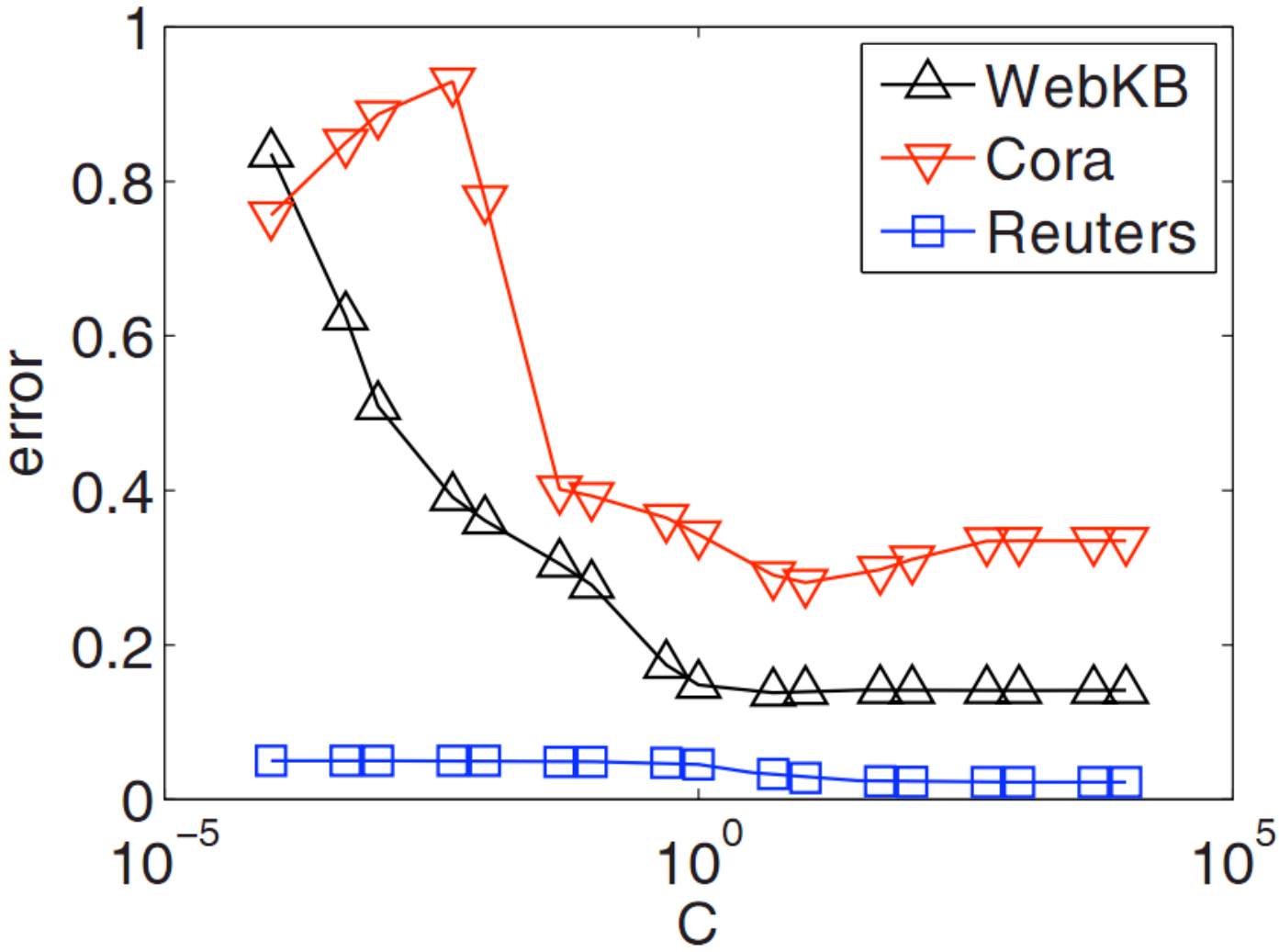
Cora


34
33
32
31
30
29
28
27
error
0
5
10
15
20
graph size
mc
naive
gibbs
loopy
jtree

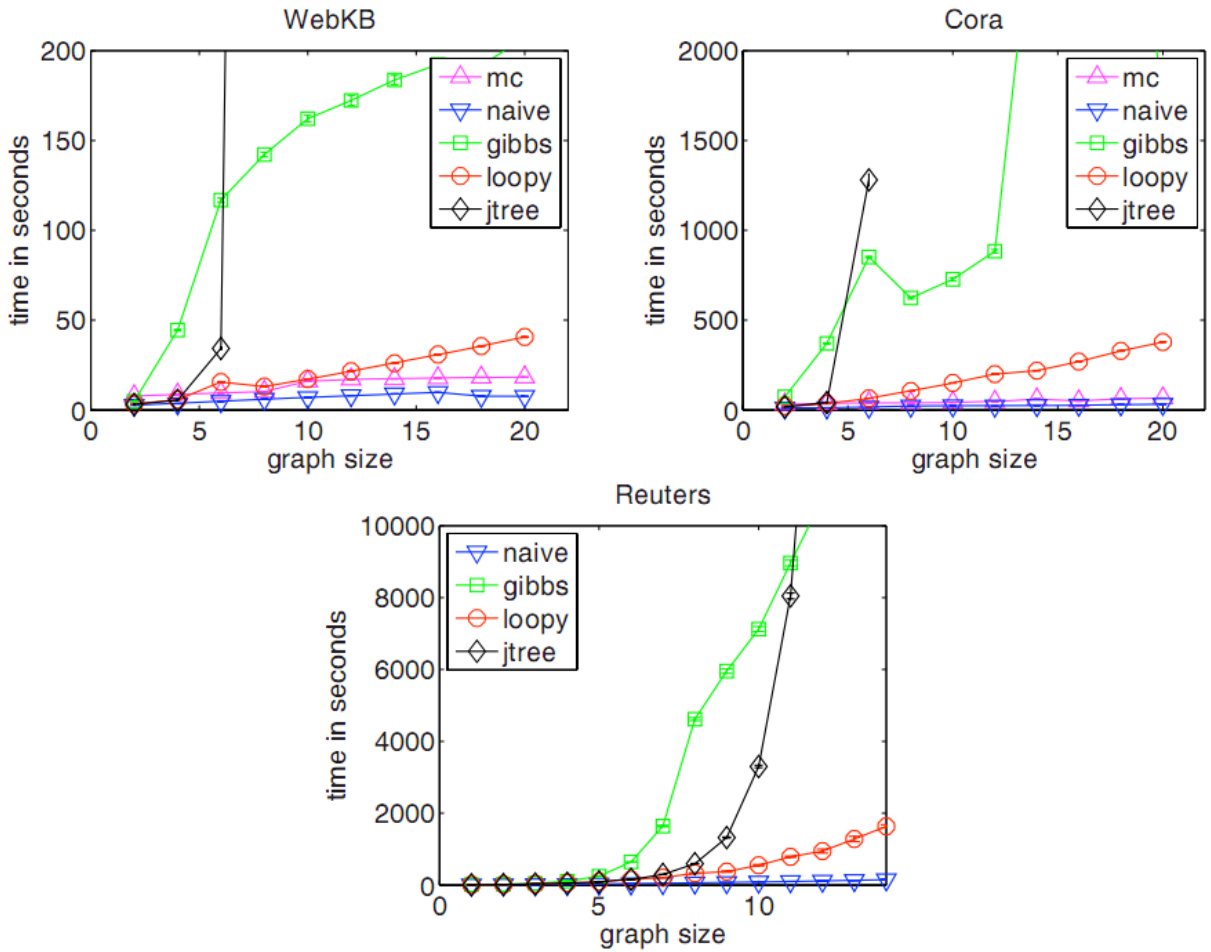
Reuters21578



Parametersuche (jtree)



Laufzeiten


WebKB
time in seconds
graph size
mc
naive
gibbs
loopy
jtree
Cora
time in seconds
graph size
mc
naive
gibbs
loopy
jtree
Reuters
time in seconds
graph size
naive
gibbs
loopy
jtree

Schlussfolgerungen

- größere Graphen liefern bessere Ergebnisse

Schlussfolgerungen

- größere Graphen liefern bessere Ergebnisse
- *loopy belief propagation* ist eine gute Alternative zum *junction tree*–Algorithmus

Schlussfolgerungen

- größere Graphen liefern bessere Ergebnisse
- *loopy belief propagation* ist eine gute Alternative zum *junction tree*–Algorithmus
- . . . außer auf dem Reuters-Datensatz

Schlussfolgerungen

- größere Graphen liefern bessere Ergebnisse
- *loopy belief propagation* ist eine gute Alternative zum *junction tree*–Algorithmus
- . . . außer auf dem Reuters-Datensatz
- *Gibbs sampling* konvergiert sehr langsam

Schlussfolgerungen

- größere Graphen liefern bessere Ergebnisse
- *loopy belief propagation* ist eine gute Alternative zum *junction tree*–Algorithmus
- . . . außer auf dem Reuters-Datensatz
- *Gibbs sampling* konvergiert sehr langsam
- jtree liefert auf kleinen Graphen die besten Ergebnisse, ist auf großen aber nicht praktikabel
  $\longrightarrow$ es kann sich auszahlen, große Graphen in kleine zu zerlegen

Schlussfolgerungen

- größere Graphen liefern bessere Ergebnisse
- *loopy belief propagation* ist eine gute Alternative zum *junction tree*–Algorithmus
- . . . außer auf dem Reuters-Datensatz
- *Gibbs sampling* konvergiert sehr langsam
- jtree liefert auf kleinen Graphen die besten Ergebnisse, ist auf großen aber nicht praktikabel
  $\longrightarrow$ es kann sich auszahlen, große Graphen in kleine zu zerlegen
- . . . und die Experimente zeigen, dass das brauchbare Resultate liefert!

Vielen Dank für eure Aufmerksamkeit!