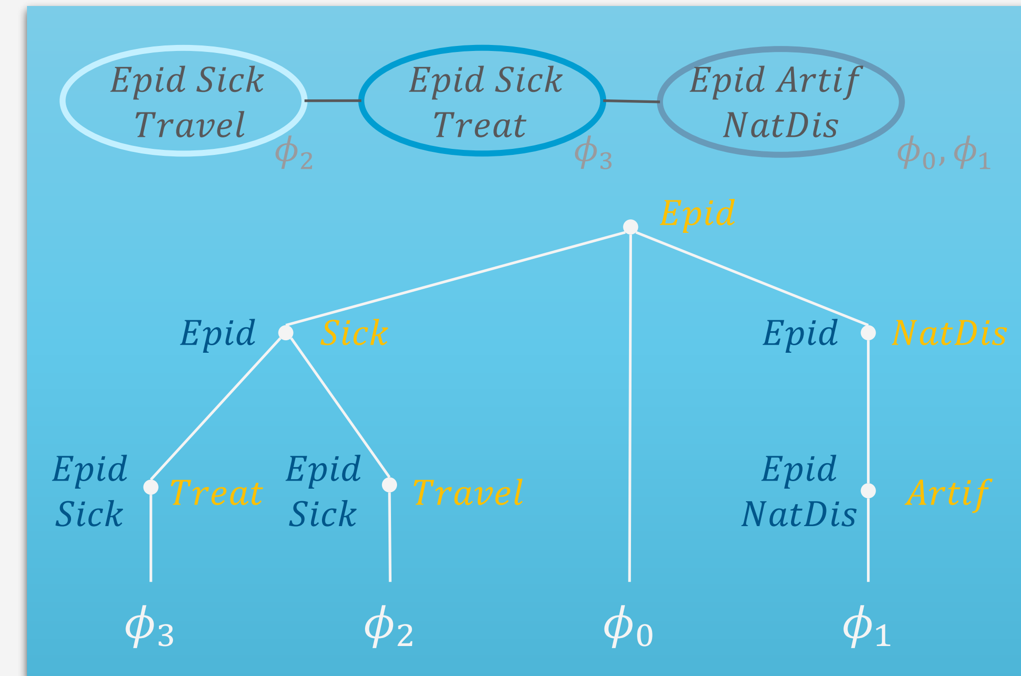
WWU MÜNSTER

Wir tragen Maske! We wear masks!

Exakte Inferenz in Episodischen PGMs

(b) Multi-Anfragen: Junction Tree Algorithmus

Einführung in die
Künstliche Intelligenz



wissen.leben

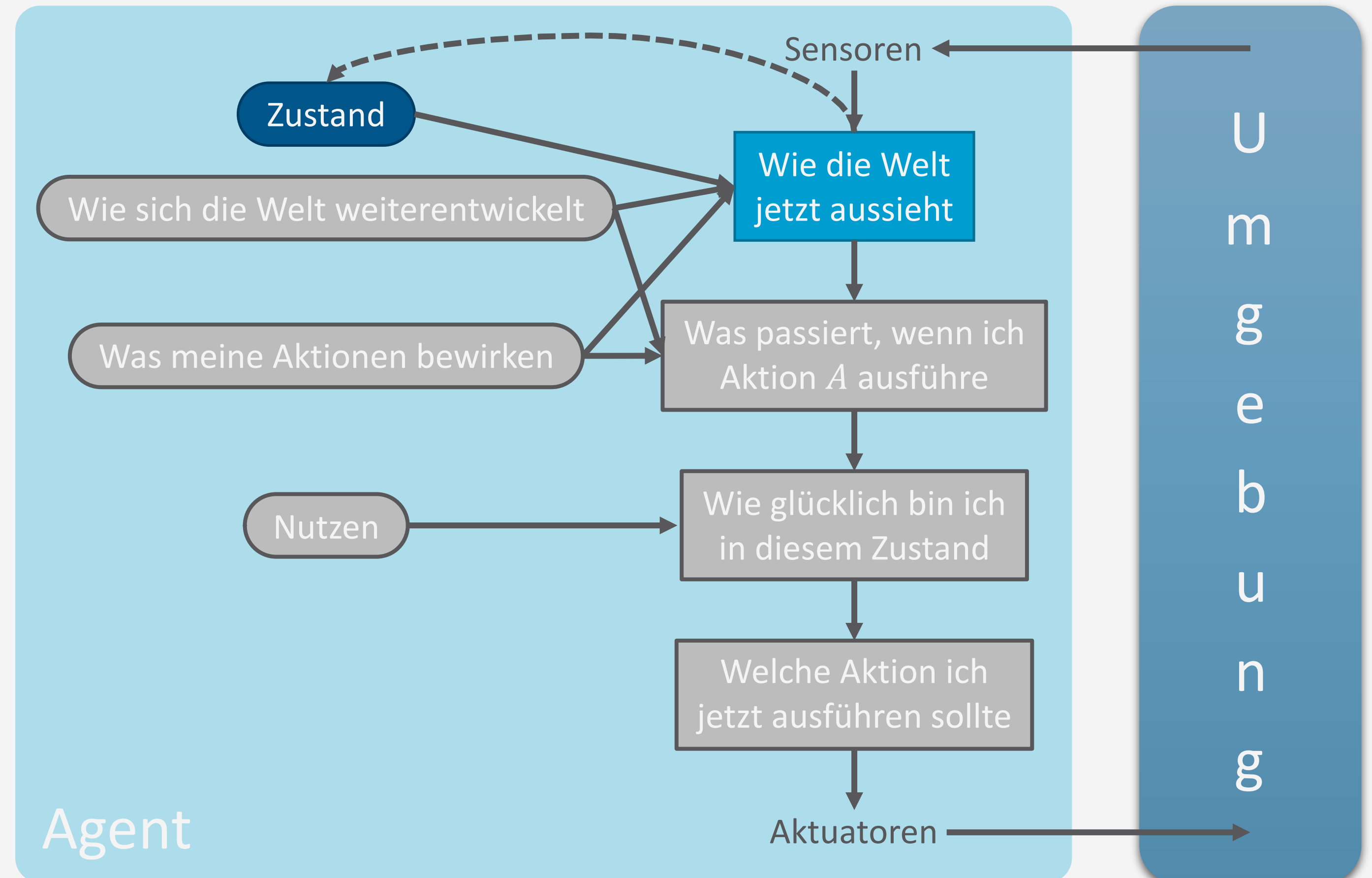
Tanya Braun
Arbeitsgruppe Data Science, Institut für Informatik

Inhalte

1. **Künstliche Intelligenz & Agenten**
   - Agentenabstraktion, Rationalität
   - Aufgabenumgebung
2. **Episodische PGMs**
   - Gerichtetes Modell: Bayes Netze (BNs)
   - Ungerichtete Modelle
3. **Exakte Inferenz in episodischen PGMs**
   - Wahrscheinlichkeits- und Zustandsanfragen
   - Direkt auf den Modellen, mittels Hilfsstrukturen
4. **Approximative Inferenz in episodischen PGMs**
   - Wahrscheinlichkeitsanfragen
   - Deterministische, stochastische Algorithmen
5. **Lernalgorithmen für episodische PGMs**
   - Bei (nicht) vollständigen Daten, (un)bekannter Struktur
6. **Sequentielle PGMs und Inferenz**
   - Dynamische BNs, Hidden-Markov-Modelle
   - filtering / prediction / hindsight Anfragen, wahrscheinlichste Zustandssequenz
   - Exakter, approximativer Algorithmus
7. **Entscheidungstheoretische PGMs**
   - Präferenzen, Nutzenprinzip
   - PGMs mit Entscheidungs- und Nutzenknoten
   - Berechnung der besten Aktion (Aktionssequenz)
8. **Abschlussbetrachtungen**

Einordnung der Vorlesung: *Modell- und nutzenbasierter Agent*

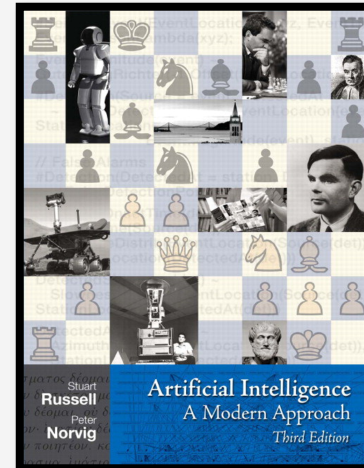
- Nachfolgende Themen der Vorlesung
  2. Episodische PGMs
  3. Exakte Inferenz in episodischen PGMs
  4. Approximative Inferenz in episodischen PGMs
  5. Lernalgorithmen für episodische PGMs
  6. Sequentielle PGMs und Inferenz
  7. Entscheidungstheoretische PGMs

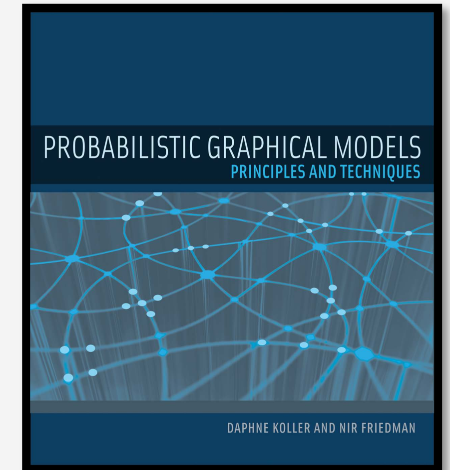


Literaturhinweise

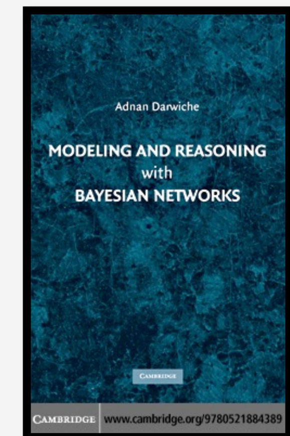
Inhalte dieses Themenblocks werden in den folgenden Kapiteln der Vorlesungsbücher behandelt

- AIMA(de)
  - Kap. 14.4: Exakte Inferenz in Bayes Netzen
- PGM
  - Kap. 9: Variableneliminierung
  - Kap. 10: Exakte Inferenz: Cliquen-Bäume
  - Kap. 13: MAP Inferenz





- Wer gerne ein anderes Buch ausprobieren möchte (Fokus auf BNs):
  - Adnan Darwiche, *Modelling and Reasoning with Bayesian Networks*, 2009.



Überblick: 3. Exakte Inferenz in episodischen PGMs

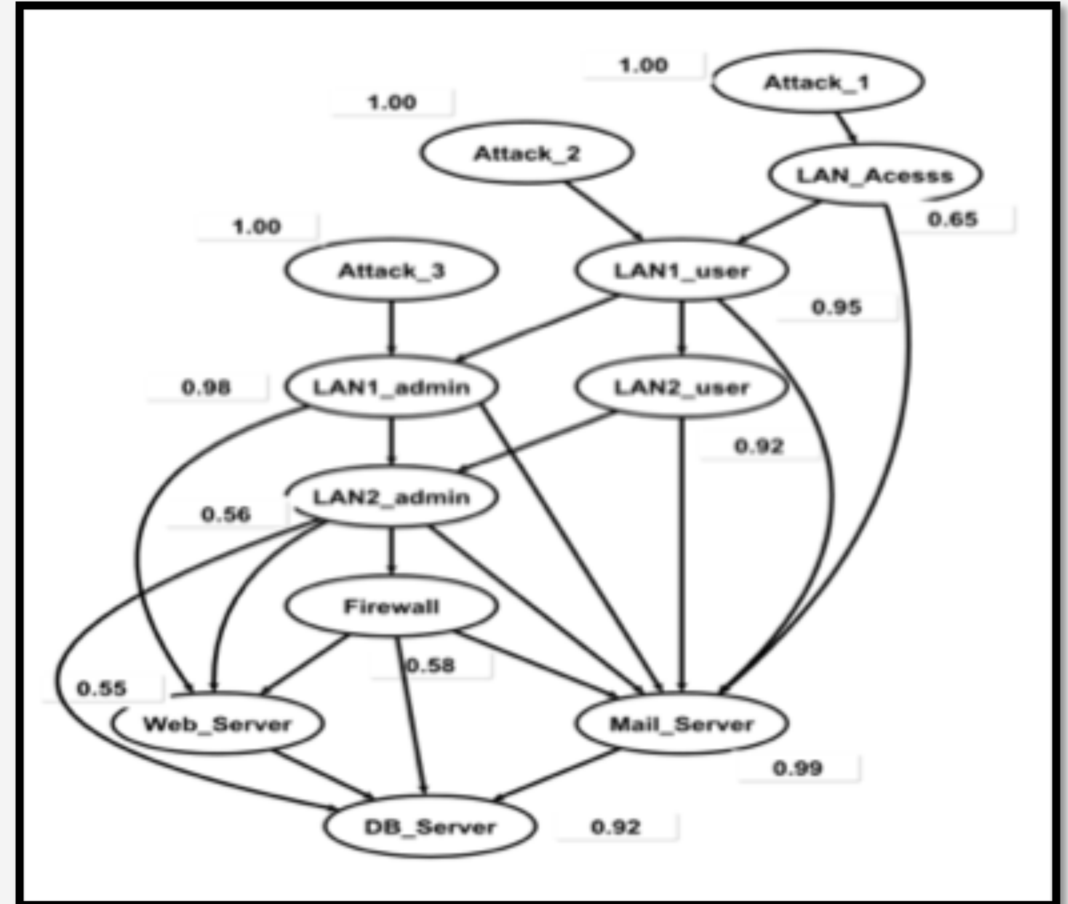
A. *Einzelanfragen: Variableneliminierung (VE)*
- Algorithmus, Operatoren für Wahrscheinlichkeitsanfragen
- Dekompositionsbäume, Komplexität

B. *Multi-Anfragen: Junction Tree (Cliquen-Bäume) Algorithmus (JT)*
- Cliquen, Junction Tree als Hilfsstruktur, Vorverarbeitung und Anfragebeantwortung
- Zusammenhang mit VE, Komplexität
- *Geschichtsstunde: Pearl‘s Probability Propagation (PP)* auf Polytree BNs

C. *Inferenzproblem Zustandsanfragen*
- Ausprägungen: Most probable explanation (MPE) / maximum a posteriori Anfragen (MAP)
- Semantik: Ausmaximieren statt aussummieren; max-out Operator in VE
- Auswirkungen auf die Komplexität

Anwendung: *Bayesian Attack Graphs*

- Anfragen: $P(A_1), P(A_2), P(A_3), P(L_{ac}), P(L_{1u}), P(L_{2u}), P(L_{1a}), P(L_{2a}), P(F), P(S_w), P(S_m), P(S_{db})$
  - Von einer Anfrage zur nächsten ändert sich eine Summe bzw. jeweils eine fällt raus aus $P(.)$:
    $$\sum_{s_{db}} \sum_{s_m} \sum_{s_w} \sum_{f} \sum_{l_{2a}} \sum_{l_{1a}} \sum_{l_{2u}} \sum_{l_{1u}} \sum_{l_{ac}} \sum_{a_3} \sum_{a_2} \sum_{a_1} P_B$$
  - Gute Eliminationsreihenfolgen ändern sich nur minimal zwischen benachbarten Knoten
  - Ergebnisse von aufwendigen Aussummierungen möchte man wiederverwenden
  - Problem der Mehrfachanfragen effizient lösen



Luis Muñoz-González, Daniele Sgandurra, Martín Barrère, and Emil C. Lupu: Exact Inference Techniques for the Analysis of Bayesian Attack Graphs. In: *IEEE Transactions on dependable and Secure Computing*, 2017.
Abbildungen ebenda.

Problem: Mehrfachanfragen

- Inferenzproblem noch immer das gleiche: *Anfragebeantwortung*
  - Berechne die Antwort auf eine Anfrage gegeben eine vollständigen gemeinsamen Wahrscheinlichkeitsverteilung $P_{\boldsymbol{R}}$
    - Faktorisierung von $P_{\boldsymbol{R}}$ in einem Modell kodiert
- Nur jetzt: eine *Menge von Anfragen* gegeben eine vollständigen gemeinsamen Wahrscheinlichkeitsverteilung $P_{\boldsymbol{R}}$
- Ziel: Möglichst effizient vorgehen
  - Bzw. besser sein als VE, was immer wieder mit dem Original Eingabemodell anfängt
    - Achtung: Implementierungen sind manchmal „schlauer“, wenn Programmiersprachen gutes Caching mitbringen
  - Wie schaffen wir das auf der algorithmischen Seite? Können wir Bekanntes nutzen?

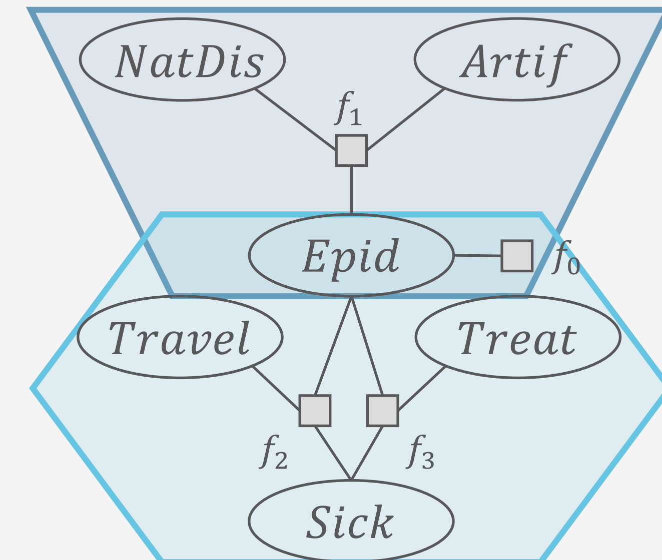
Beispiel

- Menge von Anfragen
  - $P(Epid), P(NatDis), P(Artif), P(Travel), P(Treat), P(Sick)$
    - Bei $P(NatDis), P(Artif)$ würde der *untere* Teil des Modells ...
    - Bei $P(Travel), P(Treat), P(Sick)$ würde der *obere* Teil des Modells ...

    ... gleich aussummiert werden
- Wäre es da nicht schön, wenn für die Anfragen
  - $P(NatDis), P(Artif)$ nur der *obere* Teil des Modells ...
  - $P(Travel), P(Treat), P(Sick)$ nur der *untere* Teil des Modells ...

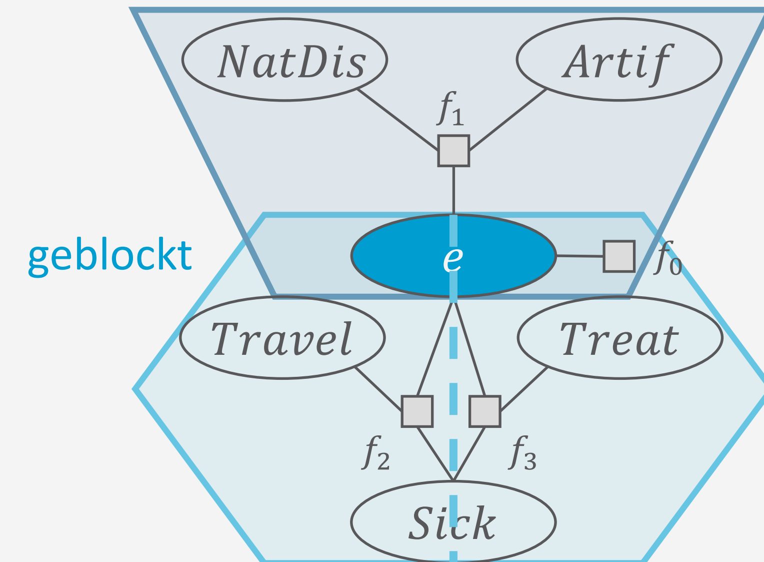
  ... relevant wäre?
  - Und, was außerhalb passiert, jeweils im Vorhinein zur Verfügung stünde?

$P(Epid)$ könnte man beiden Teilen zuordnen bzw. es gilt „sowohl ... als auch...“



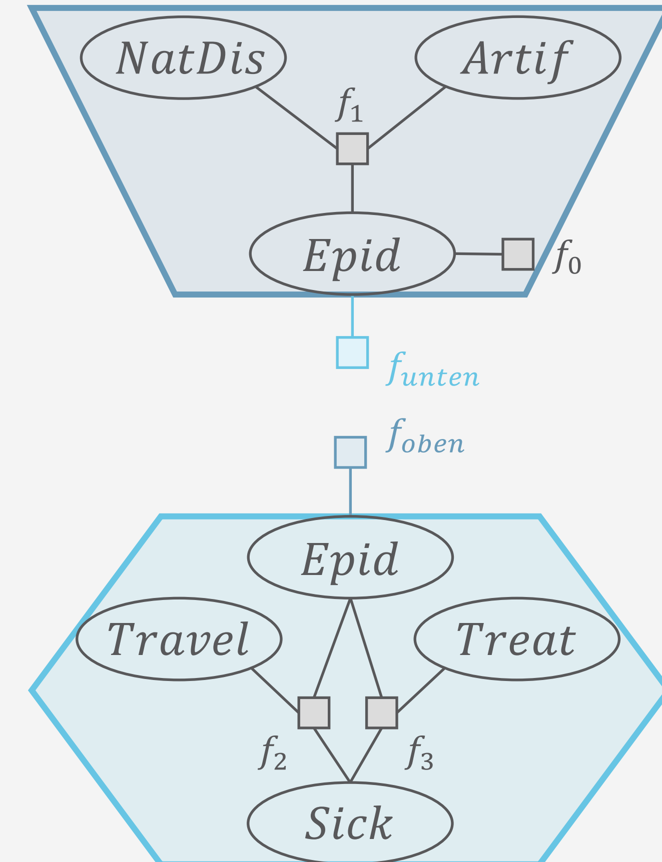
Unabhängigkeiten zur Rettung?

- Globale Unabhängigkeiten im Modell:
  - $Treat, Travel, Sick \perp NatDis, Artif \mid Epid$
  - $Treat \perp Travel \mid \{Epid, Sick\}$
  - Wenn $Epid$ gegeben ist, dann ist oben von unten unabhängig und vice versa
  - Wenn dazu noch $Sick$ gegeben ist, dann ist auch unten rechts von unten links unabhängig und vice versa
  - Anfragen könnten unabhängig vom jeweils anderen Teil beantwortet werden
- Dafür müssten ...
  - [Variante 1] die Separatoren $Epid$ bzw. $Epid, Sick$ immer mit Evidenz belegt sein
    - Unwahrscheinlich und würde keine reinen Marginalanfragen und nur bestimmte bedingte Anfragen zulassen → nur bedingt hilfreich ☇



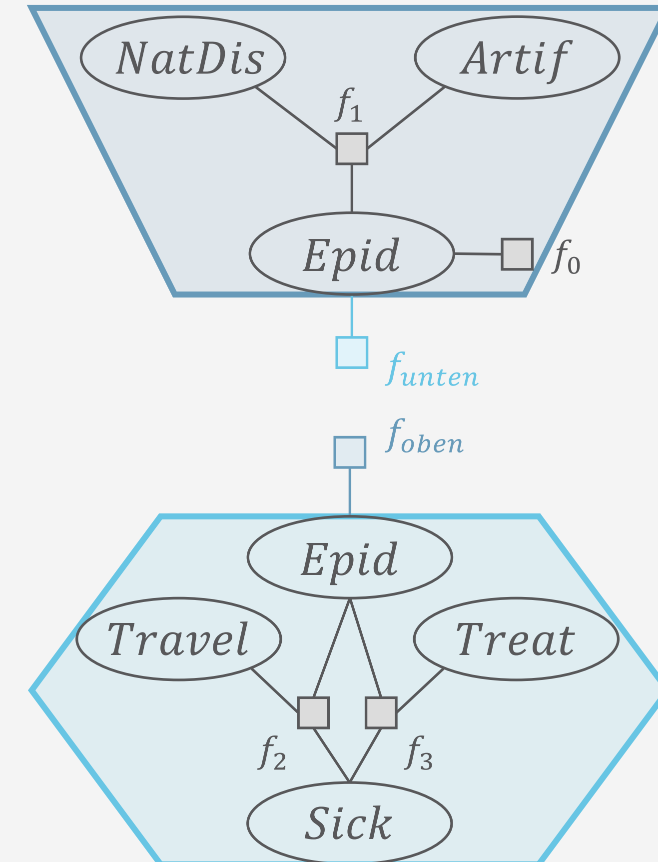
Unabhängigkeiten zur Rettung?

- Globale Unabhängigkeiten im Modell:
  - $Treat, Travel, Sick \perp NatDis, Artif \mid Epid$
  - $Treat \perp Travel \mid \{Epid, Sick\}$
  - Wenn $Epid$ gegeben ist, dann ist oben von unten unabhängig und vice versa
  - Wenn dazu noch $Sick$ gegeben ist, dann ist auch unten rechts von unten links unabhängig und vice versa
  - Anfragen könnten unabhängig vom jeweils anderen Teil beantwortet werden
- Dafür müssten ...
  - [Variante 2] die Informationen aus dem jeweils anderen Teil des Modells jenseits des Separators im Vorhinein zusammengetragen und „ausgetauscht" werden → In allen Fällen hilfreich ✓



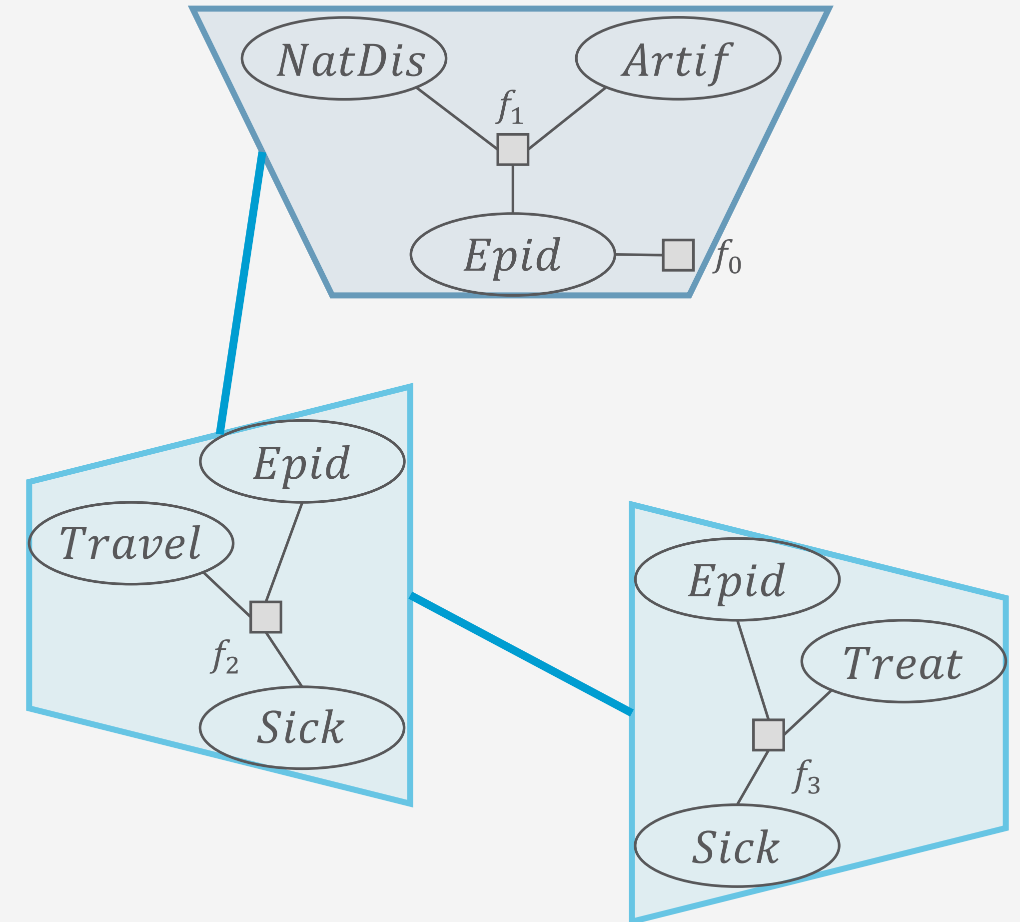
Unabhängigkeiten zur Rettung!

- So einfach? – Erstmal ja...
  - Durch globale Unabhängigkeit kommt die Information vom anderen Teil des Modells nicht über andere Wege an, sondern kann an der Schnittstelle, i.e., über die Separatoren, mittels *Nachrichten* ausgetauscht werden
  - Beispiel:
    - Separator $Epid$ zwischen unten und oben:
      - Nachricht f_{unten} beinhaltet Informationen über unten:
        VE Resultat von $Treat, Travel, Sick$ aus f_2, f_3 aussummiert
      - Nachricht f_{oben} beinhaltet Informationen über oben:
        VE Resultat von $Artif, NatDis$ aus f_1, f_0 aussummiert



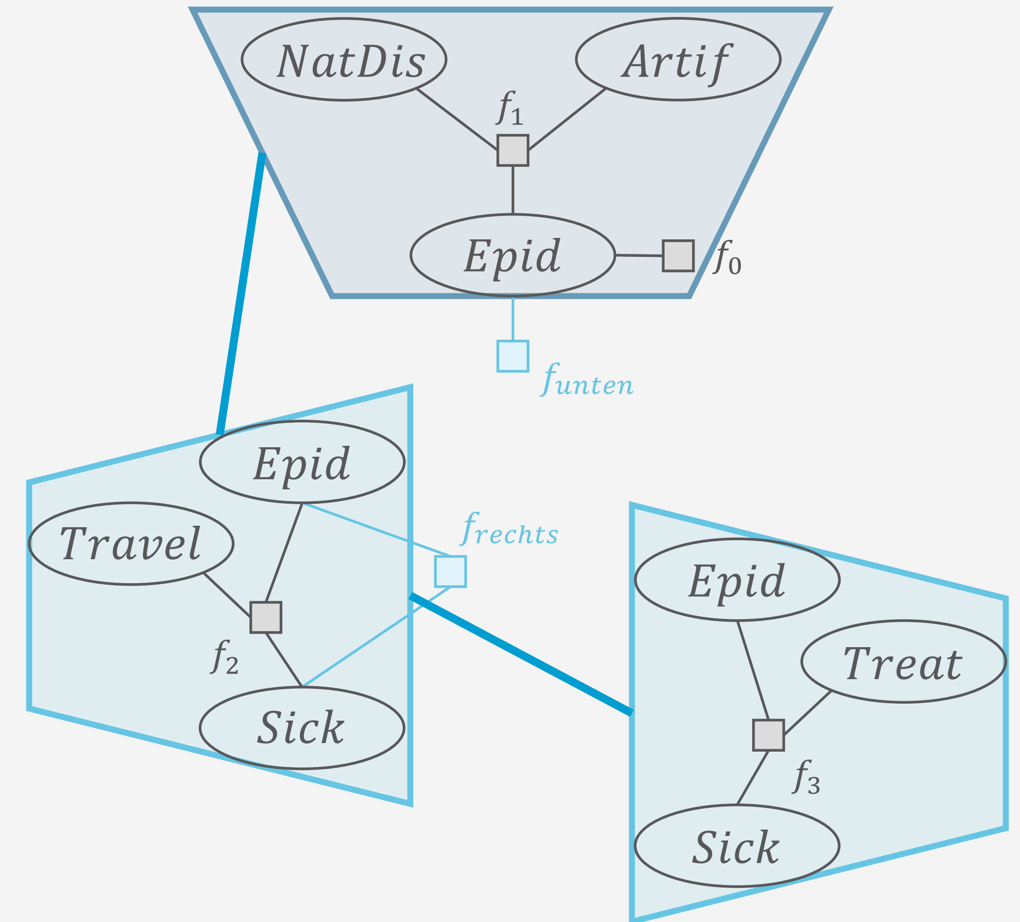
Netzwerk von Cluster

- Aber...
  - Pro Anfrage beste Schnittstellen suchen nicht effizient
  - Plus, Motivation: doppelte Rechnungen einsparen
    - Im Beispiel: Aussummierung von $Travel$ gespart für Anfragen oben, aber fällt auch bei Anfrage an $Treat$ an
- Abhilfe
  - Netzwerk von möglichst kleinen *Clustern* von Zufallsvariablen finden, zwischen denen globale Unabhängigkeiten herrschen
    - Z.B. rekursiv größere Cluster in kleinere aufteilen
    - Beispiel: unteres Cluster in zwei Subcluster aufteilen



Nachrichten im Netzwerk

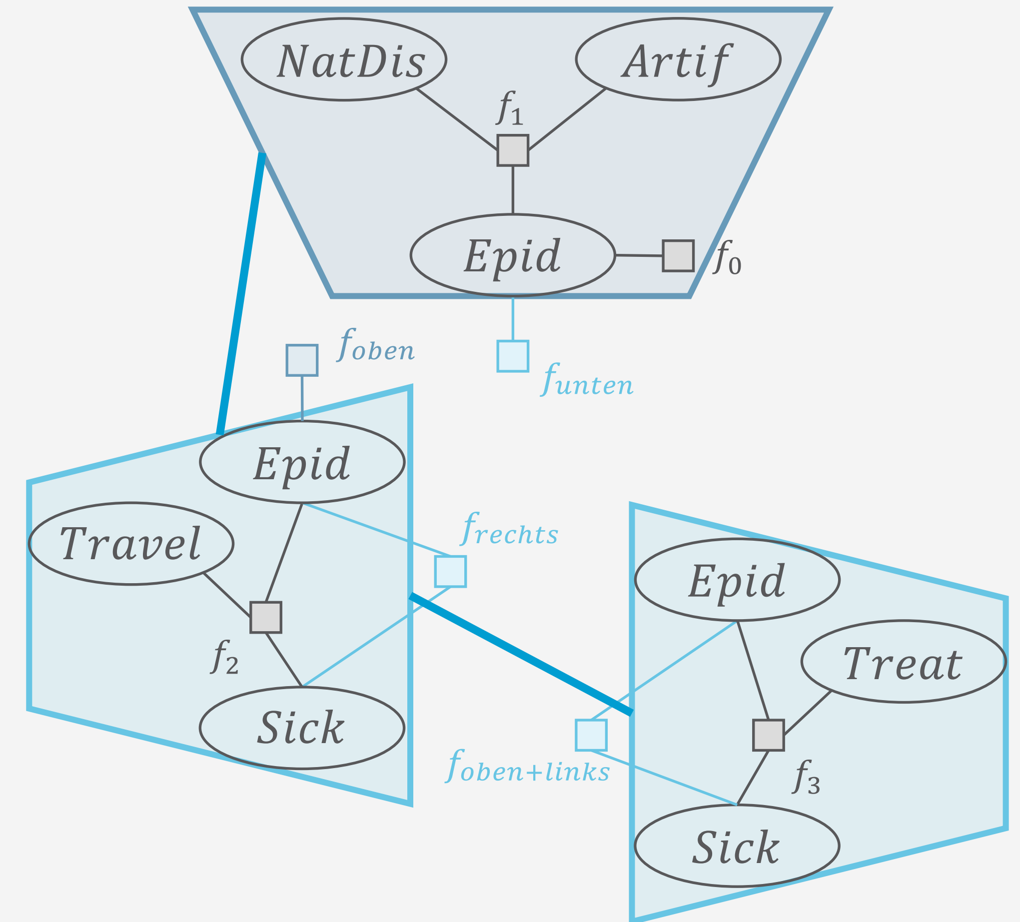
- Damit Anfragen auf Clustern beantwortet werden können, müssen die Cluster unabhängig sind
  - Dafür müssen die Nachrichten an sämtlichen Schnittstellen zur Verfügung stehen
  - Wie?
    - $\boldsymbol{C}_{oben}$ könnte eine Nachricht von $\boldsymbol{C}_{links}$ anfordern
    - Damit $\boldsymbol{C}_{links}$ Nachricht berechnen kann, braucht es Nachricht von $\boldsymbol{C}_{rechts}$
    - $\boldsymbol{C}_{links}$ fordert also die Nachricht von $\boldsymbol{C}_{rechts}$ an
    - $\boldsymbol{C}_{rechts}$ berechnet und schickt die Nachricht, f_{rechts}, an $\boldsymbol{C}_{links}$
      - Aussummieren von $Treat$ in f_3
    - $\boldsymbol{C}_{links}$ berechnet und schickt die Nachricht, f_{unten} , an $\boldsymbol{C}_{oben}$
      - Aussummieren von $Travel, Sick$ in f_2, f_{rechts}
    - $\boldsymbol{C}_{oben}$ hat dann alle Informationen aus dem Modell, um Anfragen an seine Clustervariablen zu beantworten



Nachrichten im Netzwerk

Idee verkörpert durch den Junction Tree Algorithmus

- Um jedes beliebige Cluster auf Anfragen vorzubereiten, muss jedes Cluster diese Nachrichten anfordern
- Verschicken der Nachrichten in zwei Phasen:
  - Einmal „hin“ und einmal „zurück“, i.e., zwei Nachrichten pro Kante im Netzwerk
    - „hin“-Nachrichten von $\boldsymbol{C}_{rechts}$ über $\boldsymbol{C}_{links}$ nach $\boldsymbol{C}_{oben}$ schon vorhanden
    - “zurück“-Nachrichten: f_{oben} von $\boldsymbol{C}_{oben}$ an $\boldsymbol{C}_{links}$ und $f_{oben+links}$ von $\boldsymbol{C}_{links}$ an $\boldsymbol{C}_{rechts}$
      - Für f_{oben}: $NatDis, Artif$ aus f_0, f_1 aussummieren
      - Für $f_{oben+links}$: $Travel$ aus f_{oben}, f_2 aussummieren
  - Für maximale Parallelisierbarkeit: Von der Peripherie des Netzes gen Zentrum senden und zurück



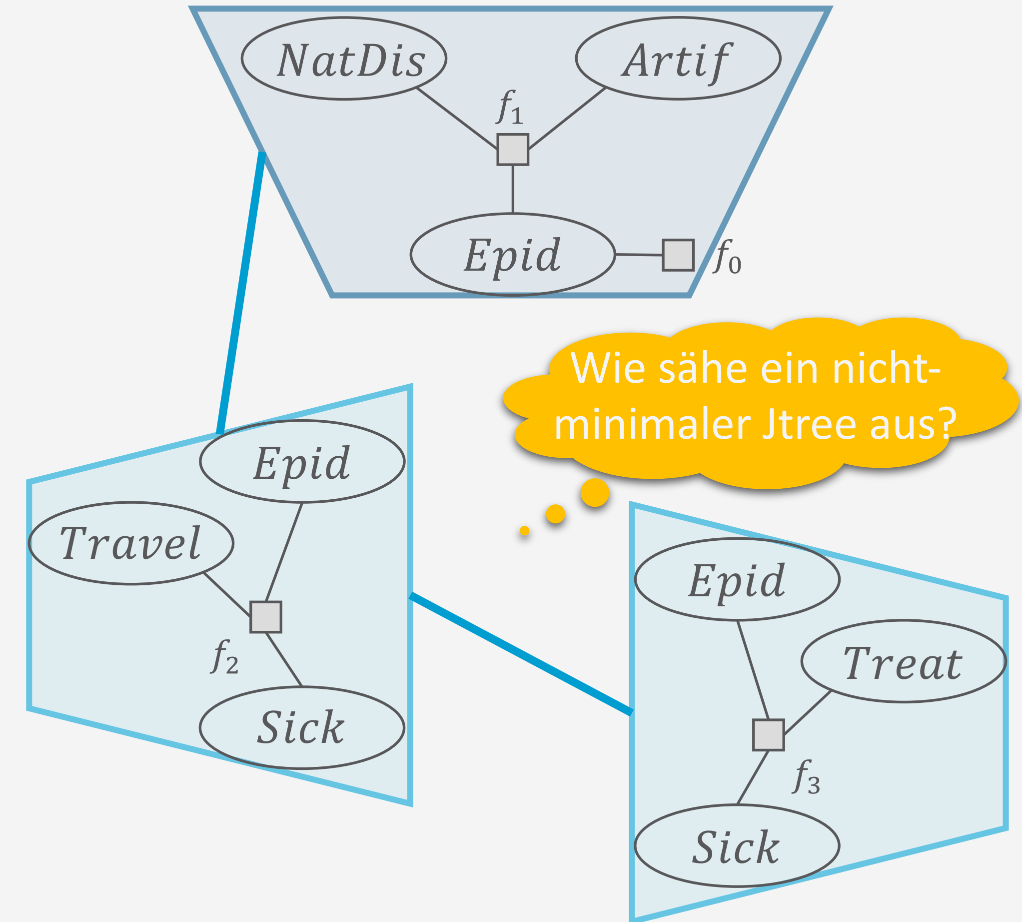
Junction Trees

- Netzwerk von Clustern Junction Tree (Jtree) genannt
  - Manchmal auch *clique tree* (PGM Buch), *jointree* (Darwiche Buch), Markov Bäume genannt
- Formale Definition: Jtree $J = (V, E)$
  - Azyklischer ungerichteter Graph
    - Knoten sind Mengen von Zufallsvariablen, genannt Cluster
    - Muss folgende Eigenschaften für ein Modell $F = \{f_i\}_{i=1}^{n}$ erfüllen (sonst kein Jtree):
      1. Alle Cluster $\boldsymbol{C}$ bestehen aus Zufallsvariablen in F: $\forall\, \boldsymbol{C} \in V : \boldsymbol{C} \subseteq \mathrm{rv}(F)$
      2. Die Argumente der Faktoren kommen zusammen in einem Cluster vor: $\forall\, f \in F : \mathrm{rv}(f) \subseteq \boldsymbol{C}$
      3. [*Running intersection property*] Wenn eine Variable in zwei Clustern vorkommt, dann kommt sie in jedem Cluster auf dem Pfad zwischen den zwei Clustern vor:
         $$\left(\exists\, R \in \mathrm{rv}(F) : R \in \boldsymbol{C}_i \wedge R \in \boldsymbol{C}_j \wedge \boldsymbol{C}_i, \boldsymbol{C}_j \in V\right) \Rightarrow \left(\forall\, \boldsymbol{C}_k \in \mathrm{path}\left(\boldsymbol{C}_i, \boldsymbol{C}_j\right) : R \in \boldsymbol{C}_k\right)$$
  - Minimal: Wenn man keine Zufallsvariable aus einem der Cluster entfernen kann, ohne dass J aufhört ein Jtree für F zu sein, i.e., mindestens eine der drei Jtree-Eigenschaften nicht mehr gilt

Jtree Eigenschaften am Beispiel

1. Alle Cluster $\boldsymbol{C}$ bestehen aus Zufallsvariablen in F: $\forall\, \boldsymbol{C} \in V : \boldsymbol{C} \subseteq \mathrm{rv}(F)$
2. Die Argumente der Faktoren kommen zusammen in einem Cluster vor: $\forall\, f \in F : \mathrm{rv}(f) \subseteq \boldsymbol{C}$
3. [*Running intersection property*] Wenn eine Variable in zwei Clustern vorkommt, dann kommt sie in jedem Cluster auf dem Pfad zwischen den zwei Clustern vor:
   $$\left(\exists\, R \in \mathrm{rv}(F) : R \in \boldsymbol{C}_i \wedge R \in \boldsymbol{C}_j \wedge \boldsymbol{C}_i, \boldsymbol{C}_j \in V\right)$$
   $$\Rightarrow \left(\forall\, \boldsymbol{C}_k \in \mathrm{path}\left(\boldsymbol{C}_i, \boldsymbol{C}_j\right) : R \in \boldsymbol{C}_k\right)$$

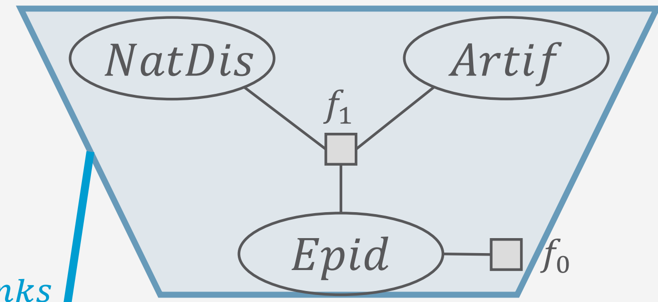
- Beispiel:
  1. Jedes Cluster besteht aus Zufallsvariablen des Modells
  2. Jeder Faktor wird von mindestens einem Cluster abgedeckt
  3. Variable $Epid$ kommt in $\boldsymbol{C}_{oben}, \boldsymbol{C}_{rechts}$ vor und auf dem Pfad dazwischen vor, nämlich in $\boldsymbol{C}_{links}$
  - Minimal, da sonst 2. nicht mehr gilt



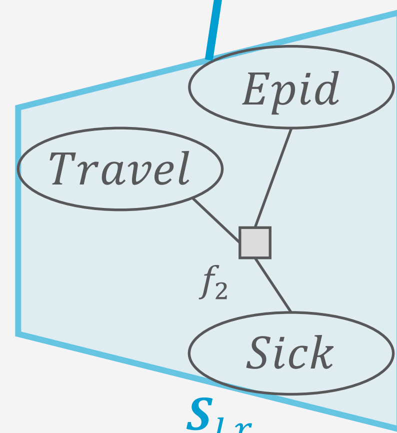
Jtrees

- Gegeben ein Faktormodell F mit einem Jtree $J = (V, E)$
- Separator $\boldsymbol{S}_{ij}$
  - Menge der Zufallsvariablen, die in beiden benachbarten Clustern $\boldsymbol{C}_i, \boldsymbol{C}_j$ vorkommen
    - Formal: $\boldsymbol{S}_{ij} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$
- Lokales Modell F_i eines Clusters $\boldsymbol{C}_i$
  - Einem Cluster zugeordnete Faktoren
    - Es gilt: $\forall f \in F_i : \mathrm{rv}(f) \subseteq \boldsymbol{C}_i$
  - Die F_i partitionieren F: $F = \bigcup_{\boldsymbol{C}_i \in V} F_i, F_i \cap F_j = \emptyset$
  - Beispiel: $F_{oben} = \{f_0, f_1\}, F_{links} = \{f_2\}, F_{rechts} = \{f_3\}$

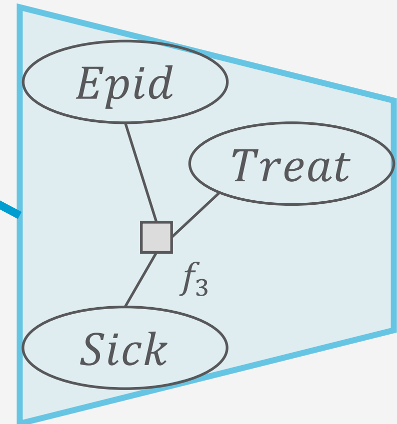
Eindeutige Zuordnung?



$\boldsymbol{S}_{o,l} = \boldsymbol{C}_{oben} \cap \boldsymbol{C}_{links} = \{Epid\}$

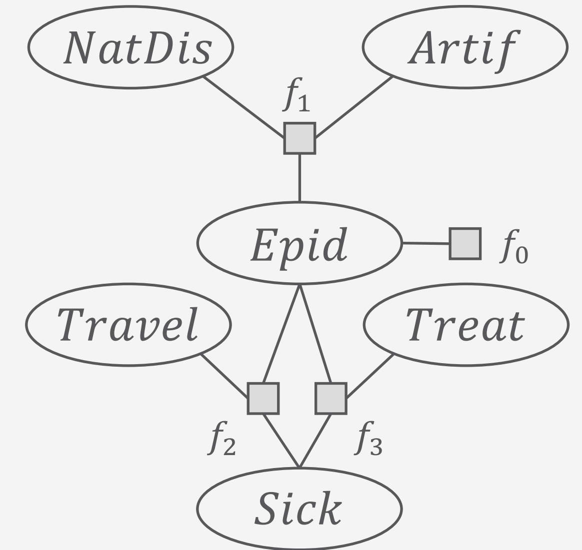


$\boldsymbol{S}_{l,r} = \boldsymbol{C}_{links} \cap \boldsymbol{C}_{rechts} = \{Epid, Sick\}$



Junction Tree Algorithmus (JT)

- Eingabe
  - Faktormodell F
  - Evidenz $\boldsymbol{e}$
  - Anfragevariablen $\{\boldsymbol{Q}_i\}_{i=1}^m$
    - Gestellte Anfragen an F: $P(\boldsymbol{Q}_i \mid \boldsymbol{e})$ für $i \in \{1, \dots, m\}$



- JT besteht konzeptionell aus vier Schritten
  1. Jtree J für das Eingabemodell F bauen
  2. Evidenz $\boldsymbol{e}$ in J eingeben
  3. Nachrichten in J schicken (*message passing*)
  4. Anfragen $\{\boldsymbol{Q}_i\}_{i=1}^m$ beantworten

Offline Vorverarbeitung
- unabhängig von $\{\boldsymbol{Q}_i\}_{i=1}^m$

Evidenz:
$sick$

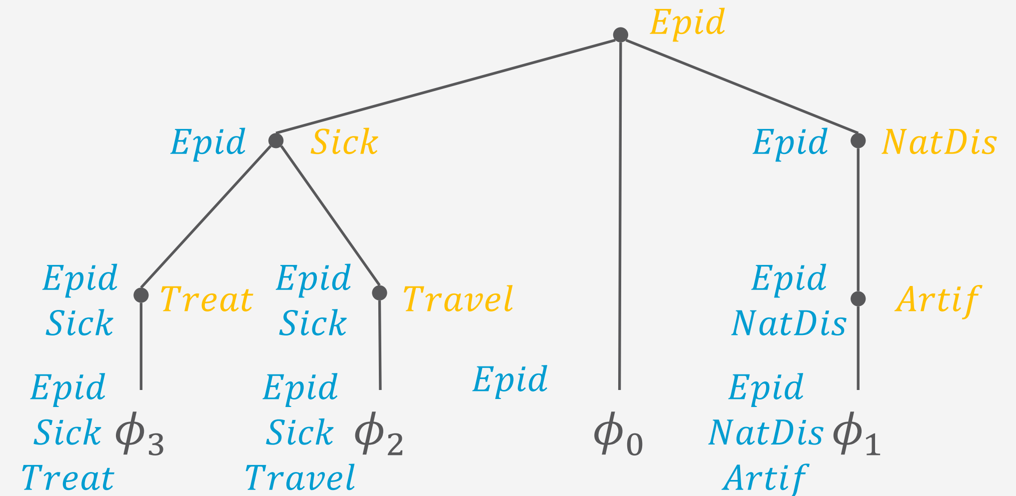
Anfragen:
$\{\{Travel\}, \{Epid\}, \{Travel, Treat\}\}$

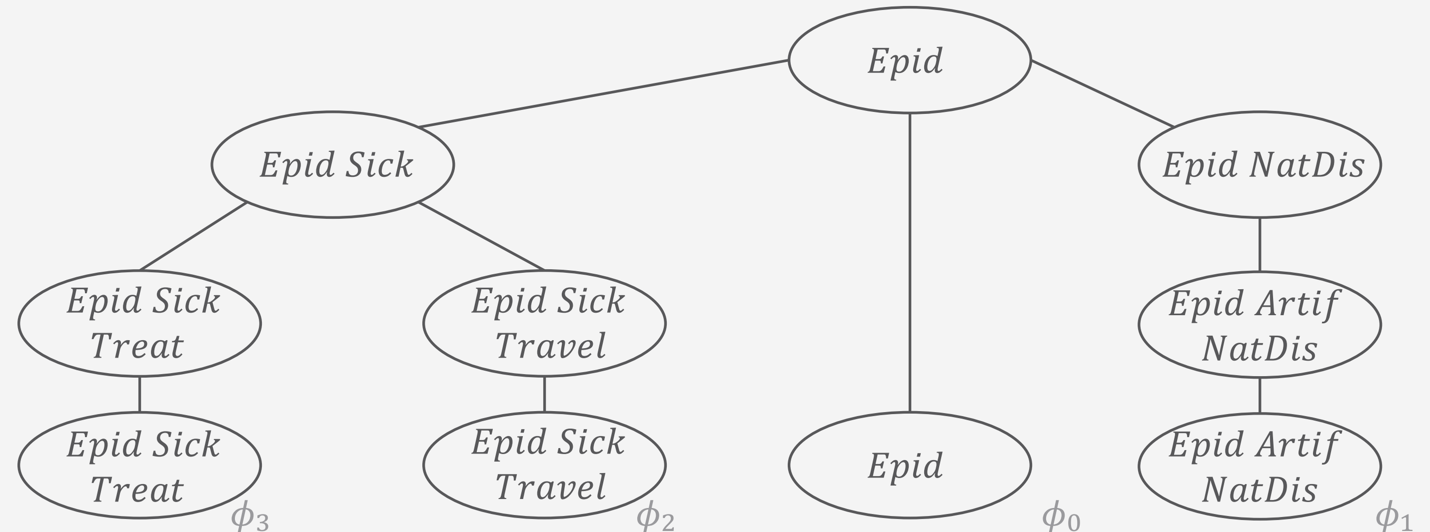
Jtrees bauen

- Mit Hilfe von Dtrees:
  - Cluster eines Dtrees formen einen nicht-minimalen Jtree
    - [Ohne Beweis] Darwiche, 2001

1. Alle Cluster $\boldsymbol{C}$ bestehen aus Zufallsvariablen in F: $\forall\ \boldsymbol{C} \in V : \boldsymbol{C} \subseteq \mathrm{rv}(F)$
2. Die Argumente der Faktoren kommen zusammen in einem Cluster vor: $\forall\ f \in F : \mathrm{rv}(f) \subseteq \boldsymbol{C}$
3. [*Running intersection property*] Wenn eine Variable in zwei Clustern vorkommt, dann kommt sie in jedem Cluster auf dem Pfad zwischen den zwei Clustern vor:

$$\left(\exists\ R \in \mathrm{rv}(F) : R \in \boldsymbol{C}_i \wedge R \in \boldsymbol{C}_j \wedge \boldsymbol{C}_i, \boldsymbol{C}_j \in V\right) \Rightarrow \left(\forall\ \boldsymbol{C}_k \in \mathrm{path}\left(\boldsymbol{C}_i, \boldsymbol{C}_j\right) : R \in \boldsymbol{C}_k\right)$$

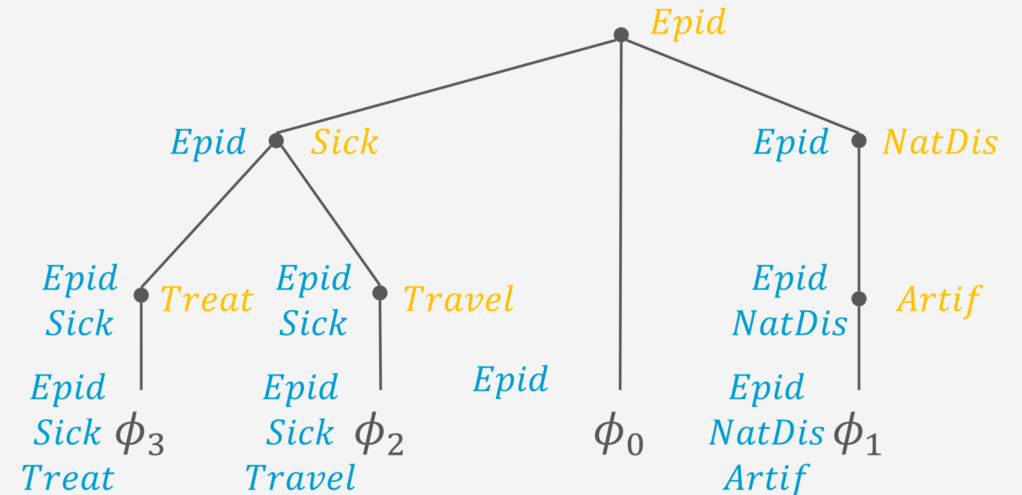




Adnan Darwiche: Recursive Conditioning. In: *Artificial Intelligence*, 2001.
Weitere Informationen auch: Darwiche (2009) Buch, Kap. 9.4.2

Jtrees bauen

- Mit Hilfe von Dtrees:
  - Cluster eines Dtrees formen einen nicht-minimalen Jtree
    - [Ohne Beweis] Darwiche, 2001
  - Jtree bauen

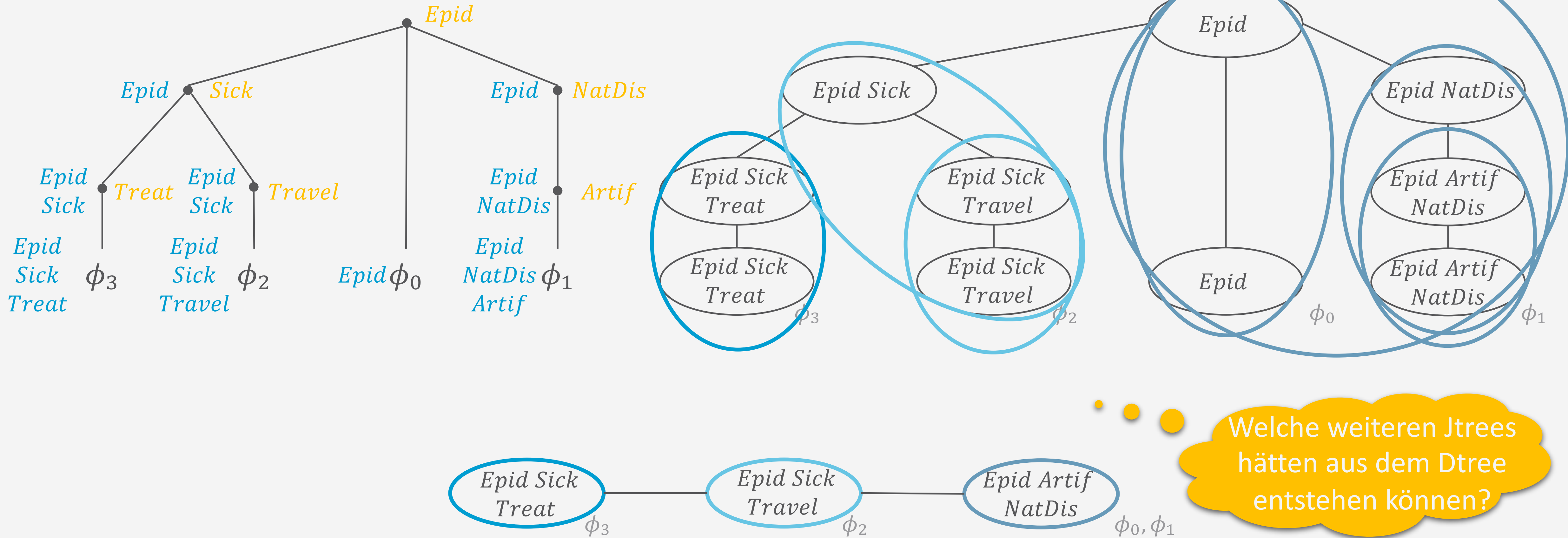


1. Dtree bauen
   - Eliminationsreihenfolge suchen und Dtree bauen (bottom up)
   - Modell mittels Heuristik partitionieren (top down) → Hinweis: *Hypergraph Partitioning*
2. Cluster im Dtree über Cutset und Kontext bestimmen
3. Nicht-minimalen Jtree als Kopie des Dtrees mit ungerichteten Kanten und Clustern als Knoten bauen, Faktoren als lokale Modelle übernehmen
4. Jtree minimieren
   - Cluster, die Untermengen von benachbarten Clustern sind, vereinigen, inklusive lokalen Modellen

JT Schritt 1

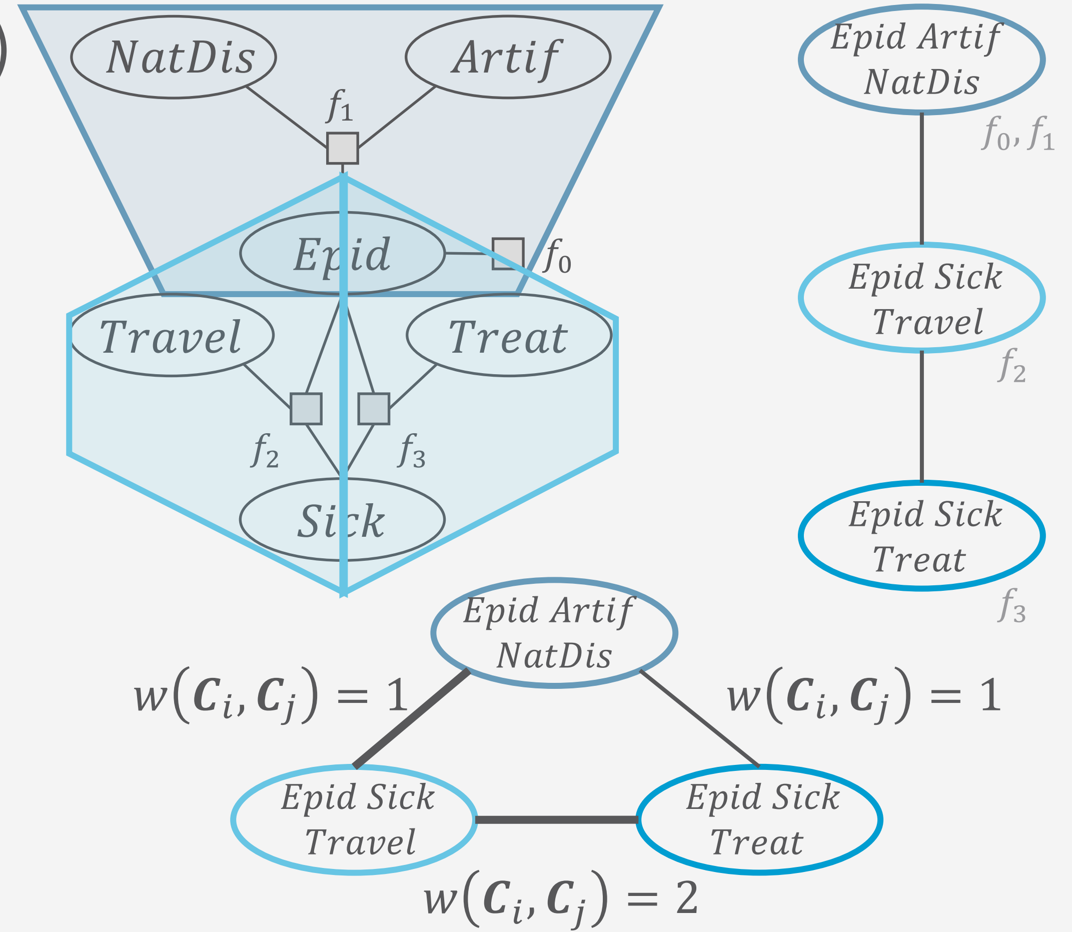
Adnan Darwiche: Recursive Conditioning. In: *Artificial Intelligence*, 2001.
Weitere Informationen auch: Darwiche (2009) Buch, Kap. 9.4.2

Jtrees bauen: Beispiel



Jtrees bauen: Eine Alternative

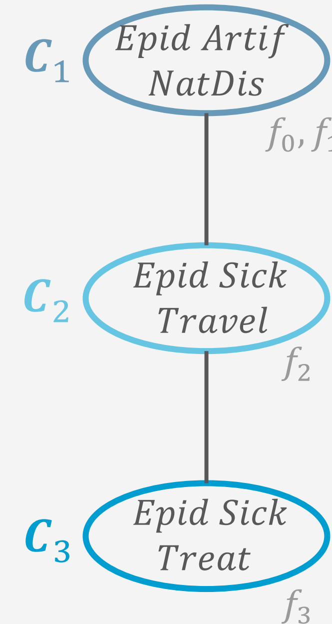
- Triangulierung + Max-ST (*maximum spanning tree*)
  1. Graph triangulieren → chordaler Graph
  2. Maximale Cliquen identifizieren
  3. Ungerichteten Graph mit maximalen Cliquen als Knoten bauen mit Kanten gewichtet mit $w(\boldsymbol{C}_i, \boldsymbol{C}_j) = |\boldsymbol{C}_i \cap \boldsymbol{C}_j|$, wann immer $w(\boldsymbol{C}_i, \boldsymbol{C}_j) > 0$
  4. Maximalen Spannbaum basierend auf $w(\boldsymbol{C}_i, \boldsymbol{C}_j)$ bauen → Jtree
  5. Jeden Faktor einem Cluster zuordnen
  - Wenn BN als Eingabe: BN als erstes moralisieren
  - Braucht keine Eliminationsreihenfolge, erfordert dafür aber Triangulierung des Graphen
    - Gleich schweres Problem

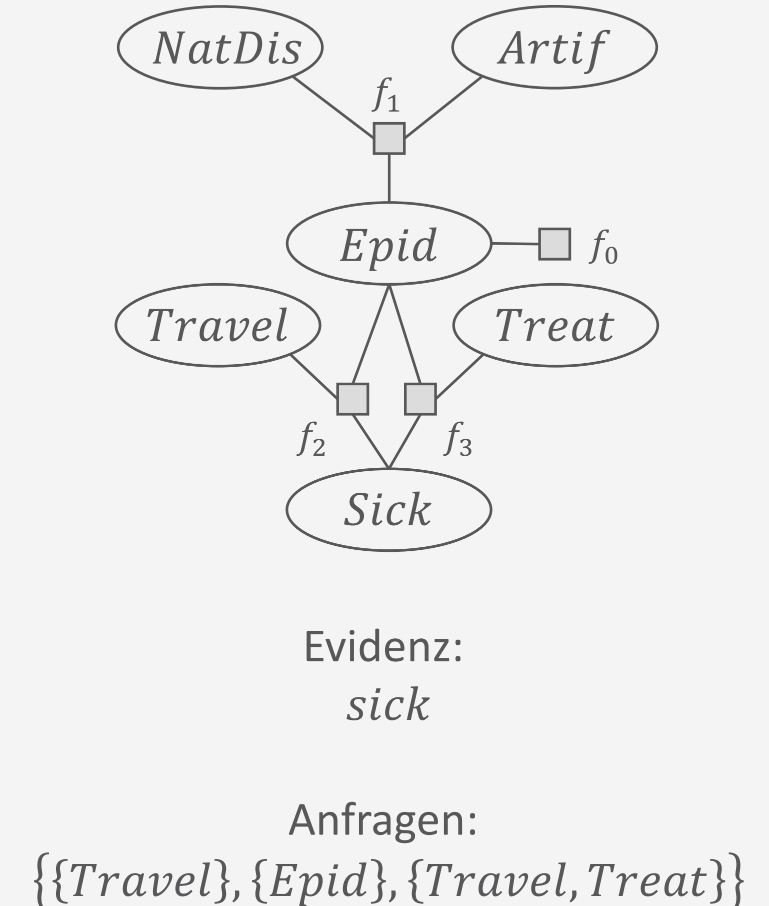


Junction Tree Algorithmus (JT)

- Eingabe
  - Faktormodell F
  - Evidenz $\boldsymbol{e}$
  - Anfragevariablen $\{\boldsymbol{Q}_i\}_{i=1}^m$
    - Gestellte Anfragen an F: $P(\boldsymbol{Q}_i \mid \boldsymbol{e})$ für $i \in \{1, \dots, m\}$

- JT besteht konzeptionell aus vier Schritten
  1. Jtree J für das Eingabemodell F bauen
  2. Evidenz $\boldsymbol{e}$ in J eingeben
  3. Nachrichten in J schicken (*message passing*)
  4. Anfragen $\{\boldsymbol{Q}_i\}_{i=1}^m$ beantworten





Evidenz in JT

- Ergebnis nach Schritt 1: Jtree $J = (V, E)$ mit lokalen Modellen F_i aus F
- Gegeben Evidenz $\boldsymbol{e}$ als Faktoren $\{\phi_e(E)\}_{e \in \boldsymbol{e}}$ kodiert

$sick \rightarrow \phi_s(Sick)$

Zwei Möglichkeiten

1. Für jedes $f_e = \phi_e(E) \in \{\phi_e(E)\}_{e \in \boldsymbol{e}}$
   - Für jedes $\boldsymbol{C}_i \in V$
     - Wenn $E \in \boldsymbol{C}_i$, dann für jedes $f \in F_i$ mit $E \in \mathrm{rv}(f)$
       - $F_i \leftarrow F_i \setminus \{f\} \cup \text{ABSORB}(f, E, f_e)$
2. Für jedes $f_e = \phi_e(E) \in \{\phi_e(E)\}_{e \in \boldsymbol{e}}$
   - Finde ein $\boldsymbol{C}_i \in V$, so dass $E \in \boldsymbol{C}_i$
   - Wähle ein $f \in F_i$, so dass $E \in \mathrm{rv}(f)$
   - $F_i \leftarrow F_i \setminus \{f\} \cup \text{MULTIPLY}(f, f_e)$

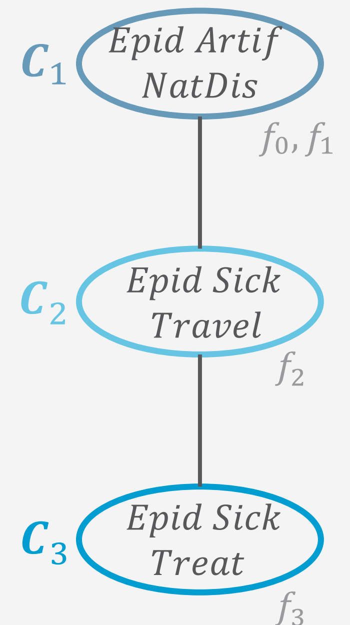
$Sick \notin C_1$

$Sick \in C_2 \rightarrow f_2^s = \text{ABSORB}(f_2, Sick, f_s)$

$Sick \in C_3 \rightarrow f_3^s = \text{ABSORB}(f_3, Sick, f_s)$

$Sick \in C_2 \rightarrow$
$f_2^0 = \text{MULTIPLY}(f_2, f_s)$

$Travel$	$Epid$	$Sick$	ϕ_2
$false$	$false$	$false$	0
$false$	$false$	$true$	24
$false$	$true$	$false$	0
$false$	$true$	$true$	6
$true$	$false$	$false$	0
$true$	$false$	$true$	8
$true$	$true$	$false$	0
$true$	$true$	$true$	2



Evidenz in JT

- Ergebnis nach Schritt 1: Jtree $J = (V, E)$ mit lokalen Modellen F_i aus F
- Gegeben Evidenz $\boldsymbol{e}$ als Faktoren $\{\phi_e(E)\}_{e \in \boldsymbol{e}}$ kodiert

Zwei Möglichkeiten

1. Für jedes $f_e = \phi_e(E) \in \{\phi_e(E)\}_{e \in \boldsymbol{e}}$ JT Schritt 2
   - Für jedes $\boldsymbol{C}_i \in V$
     - Wenn $E \in \boldsymbol{C}_i$, dann für jedes $f \in F_i$ mit $E \in \mathrm{rv}(f)$
       - $F_i \leftarrow F_i \setminus \{f\} \cup \text{ABSORB}(f, E, f_e)$
2. Für jedes $f_e = \phi_e(E) \in \{\phi_e(E)\}_{e \in \boldsymbol{e}}$
   - Finde ein $\boldsymbol{C}_i \in V$, so dass $E \in \boldsymbol{C}_i$
   - Wähle ein $f \in F_i$, so dass $E \in \mathrm{rv}(f)$
   - $F_i \leftarrow F_i \setminus \{f\} \cup \text{MULTIPLY}(f, f_e)$

Wir konzentrieren uns auf Möglichkeit 1.

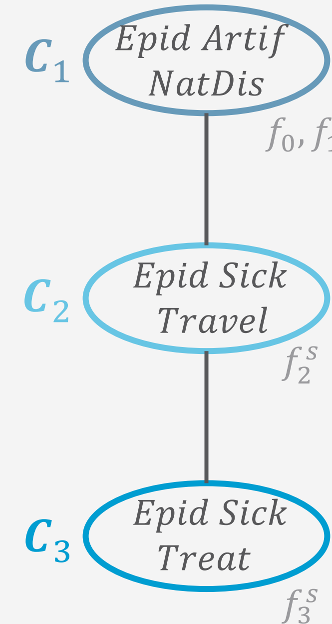
Absorbierung von Evidenz wie bei VE in jedem Faktor
- Jedes Cluster wird getestet ($E \in \boldsymbol{C}_i$), aber nicht mehr jeder Faktor (wenn $E \notin \boldsymbol{C}_i$, dann nicht F_i)
- Dimensionsreduktion um $|\boldsymbol{e}|$, da $\boldsymbol{e}$ eliminiert wird

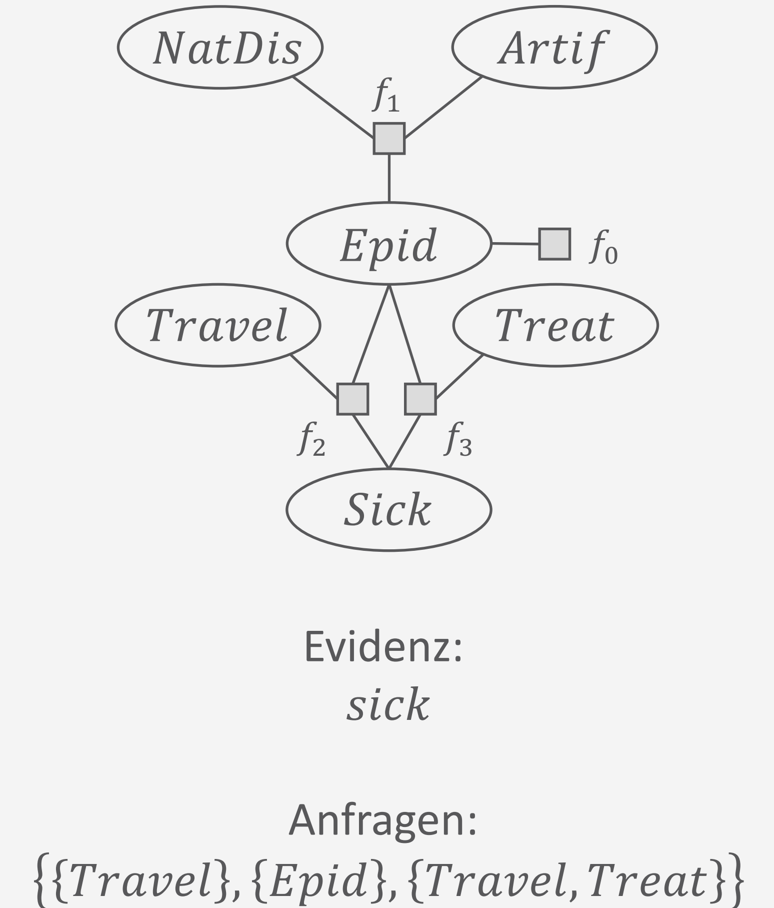
Multiplikation mit $\langle 1,0 \rangle$-Faktor, 0 später propagiert
- „Finde" und „Wähle" gleicher Worst-case, aber nur einmal Multiplikationsaufwand
- Keine Dimensionsreduktion → E muss später aussummiert werde

Junction Tree Algorithmus (JT)

- Eingabe
  - Faktormodell F
  - Evidenz $\boldsymbol{e}$
  - Anfragevariablen $\{\boldsymbol{Q}_i\}_{i=1}^m$
    - Gestellte Anfragen an F: $P(\boldsymbol{Q}_i \mid \boldsymbol{e})$ für $i \in \{1, \ldots, m\}$

- JT besteht konzeptionell aus vier Schritten
  1. Jtree J für das Eingabemodell F bauen
  2. Evidenz $\boldsymbol{e}$ in J eingeben
  3. Nachrichten in J schicken (*message passing*)
  4. Anfragen $\{\boldsymbol{Q}_i\}_{i=1}^m$ beantworten





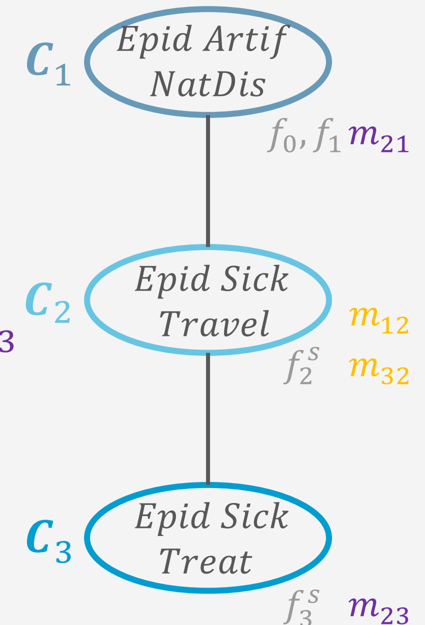
Nachrichten verschicken

- Ergebnis nach Schritt 2: lokale Modelle mit Evidenz aktualisiert
- Zwei Bedingungen zum Verschicken einer Nachricht
  1. Wenn Cluster $\boldsymbol{C}_i$ alle Nachrichten bis auf die von Nachbar $\boldsymbol{C}_j$ erhalten hat, sendet $\boldsymbol{C}_i$ die Nachricht m_{ij} an $\boldsymbol{C}_j$.
     - Automatisch wahr an den Blättern des Jtrees
  2. Wenn Cluster $\boldsymbol{C}_i$ die letzte fehlende Nachricht von Nachbar $\boldsymbol{C}_j$ erhält, sendet $\boldsymbol{C}_i$ Nachrichten m_{ik} an alle anderen Nachbarn $\boldsymbol{C}_k$
     - In paralleler Ausführung kann es sein, dass ein zentraler Cluster alle Nachrichten gleichzeitig erhält; dieser Cluster sendet dann an alle Nachbarn Nachrichten
  - Induziert einen 2-Phasen-Nachrichtenversand (nach innen, nach außen)

$\boldsymbol{C}_1$: Bed. 1 ✓
→ sendet m_{12}

$\boldsymbol{C}_2$: Bed. 2 ✓
→ sendet m_{21}, m_{23}

$\boldsymbol{C}_3$: Bed. 1 ✓
→ sendet m_{32}



Nachrichten verschicken

- Berechnen einer Nachricht von Cluster $\boldsymbol{C}_i$ zu Nachbar $\boldsymbol{C}_j$
  - Eliminieren aller Nicht-Separatorvariablen in $\boldsymbol{C}_i$, aus der Vereinigung des lokalen Modells und den Nachrichten aller anderen Nachbarn
    - Aufruf an VE ohne Multiplikation und Normalisierung am Ende:

$$\mathrm{VE}\left(F_i \cup \bigcup_{\boldsymbol{C}_k \in \mathrm{Nb}(\boldsymbol{C}_i), k \neq j} m_{ki}, \boldsymbol{S}_{ij}, \emptyset, .\right)$$

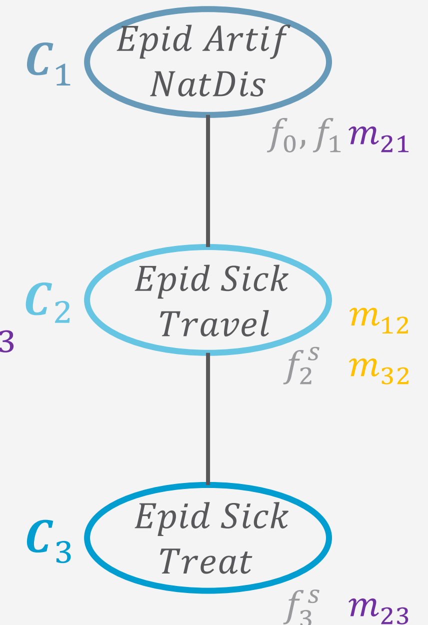
Warum?

- Eliminationsreihenfolge $\mathcal{U}$ für $\boldsymbol{C}_i \setminus \boldsymbol{S}_{ij}$ oder Heuristik h an letzter Position
    - Eliminiert die in $\boldsymbol{C}_j$ unbekannten Zufallsvariablen
    - Nachrichten aller anderen Nachbarn $\boldsymbol{C}_{k \neq j}$ machen $\boldsymbol{C}_i$ unabhängig von den Teilen des Modells
    - Vereinigung kombiniert alle Informationen, die $\boldsymbol{C}_j$ noch unbekannt sind und hinter $\boldsymbol{S}_{ij}$ liegen, so dass $\boldsymbol{C}_j$ dann unabhängig von $\boldsymbol{C}_i$ ist

$\boldsymbol{C}_1$: Bed. 1 ✓
→ sendet m_{12}

$\boldsymbol{C}_2$: Bed. 2 ✓
→ sendet m_{21}, m_{23}

$\boldsymbol{C}_3$: Bed. 1 ✓
→ sendet m_{32}



Nachrichten verschicken – Beispiel

- $m_{12} \rightarrow \mathrm{VE}\big(\{f_0, f_1\}, \{Epid\}, \emptyset, (Artif, NatDis)\big)$

$Epid$	ϕ_0
$false$	50
$true$	1

$Epid$	$NatDis$	$Artif$	ϕ_1
$false$	$false$	$false$	12
$false$	$false$	$true$	2
$false$	$true$	$false$	3
$false$	$true$	$true$	1
$true$	$false$	$false$	7
$true$	$false$	$true$	4
$true$	$true$	$false$	5
$true$	$true$	$true$	1

+ + + +

$Epid$	$NatDis$	ϕ_1'
$false$	$false$	14
$false$	$true$	4
$true$	$false$	11
$true$	$true$	6

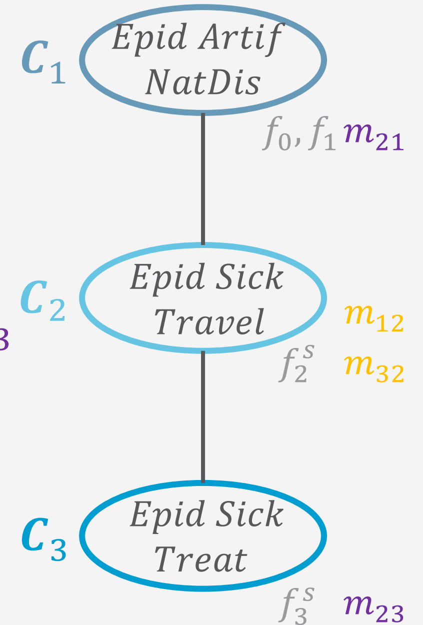
+ +

$Epid$	ϕ_1''
$false$	18
$true$	17

C_1: Bed. 1 ✓
→ sendet m_{12}

C_2: Bed. 2 ✓
→ sendet m_{21}, m_{23}

C_3: Bed. 1 ✓
→ sendet m_{32}



Nachrichten verschicken – Beispiel

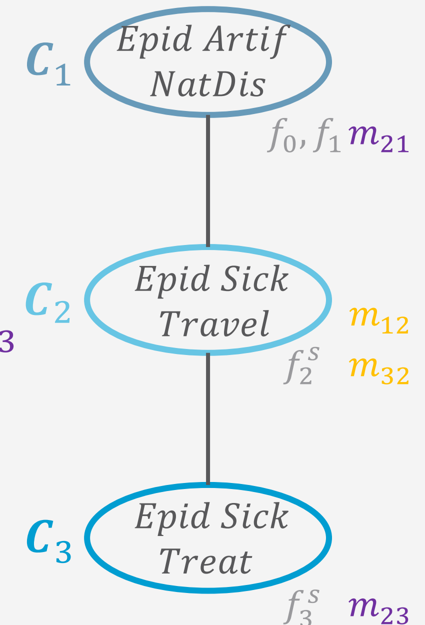
- $m_{12} \rightarrow \mathrm{VE}\big(\{f_0, f_1\}, \{Epid\}, \emptyset, (Artif, NatDis)\big)$
  - $m_{12} = \{f_0, f_1''\}$
- $m_{32} \rightarrow \mathrm{VE}\big(\{f_3^s\}, \{Epid, Sick\}, \emptyset, (Treat)\big)$
  - $m_{32} = \{f_3^{s\prime}\}$
- $m_{21} \rightarrow \mathrm{VE}\big(\{f_2^s, f_3^{s\prime}\}, \{Epid\}, \emptyset, (Travel, Sick)\big)$
  - $m_{21} = \{f_2^{s\prime}, f_3^{s\prime}\}$
- $m_{23} \rightarrow \mathrm{VE}\big(\{f_2^s, f_0, f_1''\}, \{Epid, Sick\}, \emptyset, (Travel)\big)$
  - $m_{23} = \{f_2^{s\prime}, f_0, f_1''\}$

$Sick$ mit Evidenz belegt und schon durch Absorption eliminiert.
Anfrage bzw. Aussummierung von $Sick$ hat hier keinen Effekt.

C_1: Bed. 1 ✓
→ sendet m_{12}

C_2: Bed. 2 ✓
→ sendet m_{21}, m_{23}

C_3: Bed. 1 ✓
→ sendet m_{32}



VE für Nachrichtenversand in JT: Ohne Evidenz und Normalisierung

$\mathbf{VE}\text{–JT}(F, \boldsymbol{S}, h)$

while $\mathrm{rv}(F) \setminus \boldsymbol{S} \neq \emptyset$ **do** — Nichtseparator-Variablen eliminieren

$U \leftarrow \arg\min_R h(F)$ ▹ Wähle nächstes U zur Eliminierung

while $\exists f_1, f_2 \in F : U \in \mathrm{rv}(f_1) \wedge \mathrm{rv}(f_2)$ **do**

$F \leftarrow F \setminus \{f_1, f_2\} \cup \{\textsc{multiply}(f_1, f_2)\}$ ▹ Multipliziere f_1, f_2 in F

$F \leftarrow F \setminus \{f\} \cup \{\textsc{sum-out}(f, U)\}$ ▹ Summiere U aus F aus

return F

Algorithmische Übersicht: Nachrichtenversand

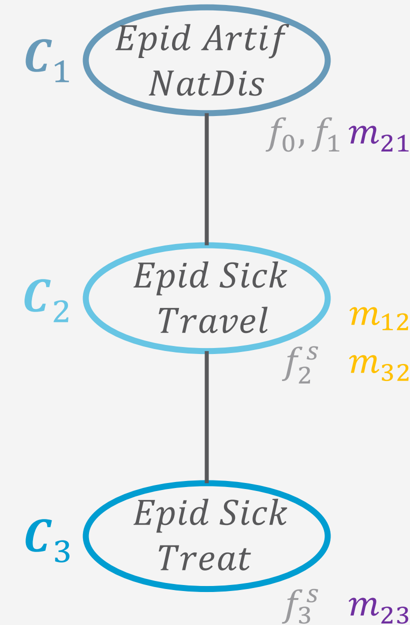
- Gegeben eine Heuristik h
- Während es ein Cluster $\boldsymbol{C}_i$ gibt, welches noch nicht Nachrichten an alle Nachbarn verschickt hat, i.e., $\sum_{j \in \text{Nb}(\boldsymbol{C}_i)} 1 \mid m_{ij}\ gesendet \leq |\text{Nb}(\boldsymbol{C}_i)|$,
  - Wenn Bedingung 1 mit Nachbar $\boldsymbol{C}_j$ gilt
    - Berechne Nachricht m_{ij}: $m_{ij} \leftarrow \text{VE–JT}\left(F_i \cup_{\boldsymbol{C}_k \in \text{Nb}(C_i)} m_{ki}, \boldsymbol{S}_{ij}, h\right)$
    - Sende m_{ij} an $\boldsymbol{C}_j$
  - Wenn Bedingung 2 gilt
    - Für alle Nachbarn $\boldsymbol{C}_j$, die noch keine Nachricht erhalten haben
      - Berechne Nachricht m_{ij}: $m_{ij} \leftarrow \text{VE–JT}\left(F_i \cup_{\boldsymbol{C}_k \in \text{Nb}(C_i), k \neq j} m_{ki}, \boldsymbol{S}_{ij}, h\right)$
      - Sende m_{ij} an $\boldsymbol{C}_j$

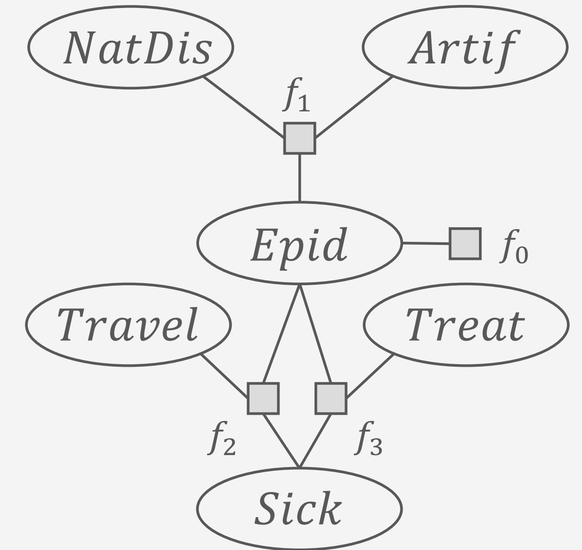
JT Schritt 3

Junction Tree Algorithmus (JT)

- Eingabe
  - Faktormodell F
  - Evidenz $\boldsymbol{e}$
  - Anfragevariablen $\{\boldsymbol{Q}_i\}_{i=1}^m$
    - Gestellte Anfragen an F: $P(\boldsymbol{Q}_i \mid \boldsymbol{e})$ für $i \in \{1, \dots, m\}$

- JT besteht konzeptionell aus vier Schritten
  1. Jtree J für das Eingabemodell F bauen
  2. Evidenz $\boldsymbol{e}$ in J eingeben
  3. Nachrichten in J schicken (*message passing*)
  4. Anfragen $\{\boldsymbol{Q}_i\}_{i=1}^m$ beantworten





Evidenz:
sick

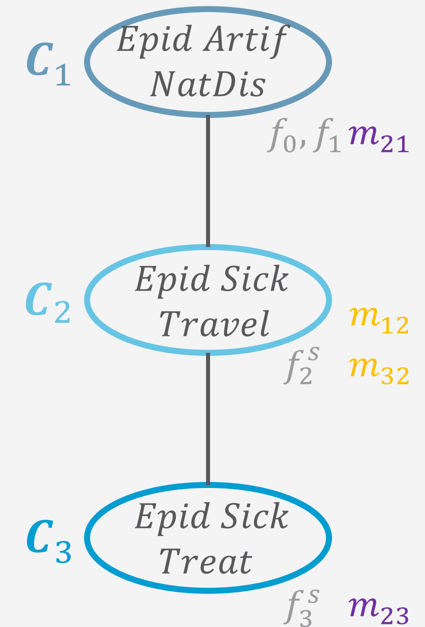
Anfragen:
$\{\{Travel\}, \{Epid\}, \{Travel, Treat\}\}$

Anfragen beantworten

- Ergebnis nach Schritt 3: Cluster unabhängig durch Nachrichten
- Einfacher Fall: Anfragevariablen $\boldsymbol{Q}$ kommen in einem Cluster vor
  - Für jede Anfrage $\boldsymbol{Q}$
    - Finde ein Cluster $\boldsymbol{C}_i$, so dass $\boldsymbol{Q} \subseteq \boldsymbol{C}_i$
    - Eliminiere alle Nicht-Anfragevariablen aus lokalem Modell und erhaltenen Nachrichten:

$$\mathrm{VE}\left(F_i \cup \bigcup_{\boldsymbol{C}_j \in \mathrm{Nb}(\boldsymbol{C}_i)} m_{ji}, \boldsymbol{Q}, \emptyset, .\right)$$

  - Beispiel:
    - $\boldsymbol{Q} = \{Travel\}$ kommt in $\boldsymbol{C}_2$ vor → $\mathrm{VE}(\{f_2^s, f_0, f_1'', f_3^{s\prime}\}, \{Travel\}, \emptyset, .)$
      - Multipliziert alle Faktoren und eliminiert $Epid$
    - $\boldsymbol{Q} = \{Epid\}$: Jedes Cluster kommt in Frage
      - Optimierungskriterien: Kleinstes Cluster, kleinstes lokales Modell, schnellster Fund



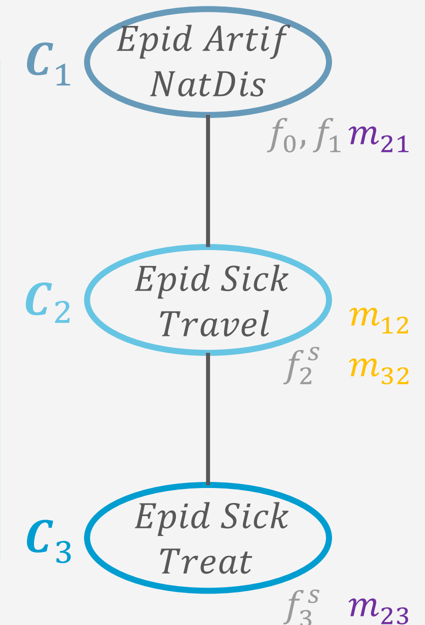
Anfragen beantworten

- Ergebnis nach Schritt 3: Cluster unabhängig durch Nachrichten
- Allgemeiner Fall:
  - Für jede Anfrage $\boldsymbol{Q}$
    - Finde ein Sub-Jtree J', so dass $\boldsymbol{Q} \subseteq \mathrm{rv}(J')$
      - Optimierungskriterien: Anzahl Cluster, Anzahl Zufallsvariablen in Clustern, erster Fund
    - Eliminiere alle Nicht-Anfragevariablen aus den lokalen Modellen und den Nachrichten von außerhalb J':

$$\mathrm{VE}\left(\bigcup_{\boldsymbol{C}_i \in J'} F_i \cup \bigcup_{\boldsymbol{C}_j \in \mathrm{Nb}(\boldsymbol{C}_i), \boldsymbol{C}_j \notin J'} m_{ji}, \boldsymbol{Q}, \emptyset, .\right)$$

JT Schritt 4

  - Beispiel:
    - $\boldsymbol{Q} = \{Travel, Treat\}$ kommt in $\boldsymbol{C}_2, \boldsymbol{C}_3$ vor $\rightarrow \mathrm{VE}(\{f_2^S, f_3^S, f_0, f_1''\}, \{Travel, Treat\}, \emptyset, .)$
      - Multipliziert alle Faktoren und eliminiert $Epid$

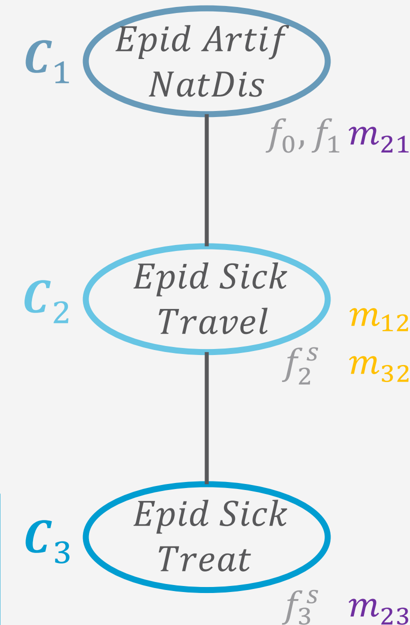


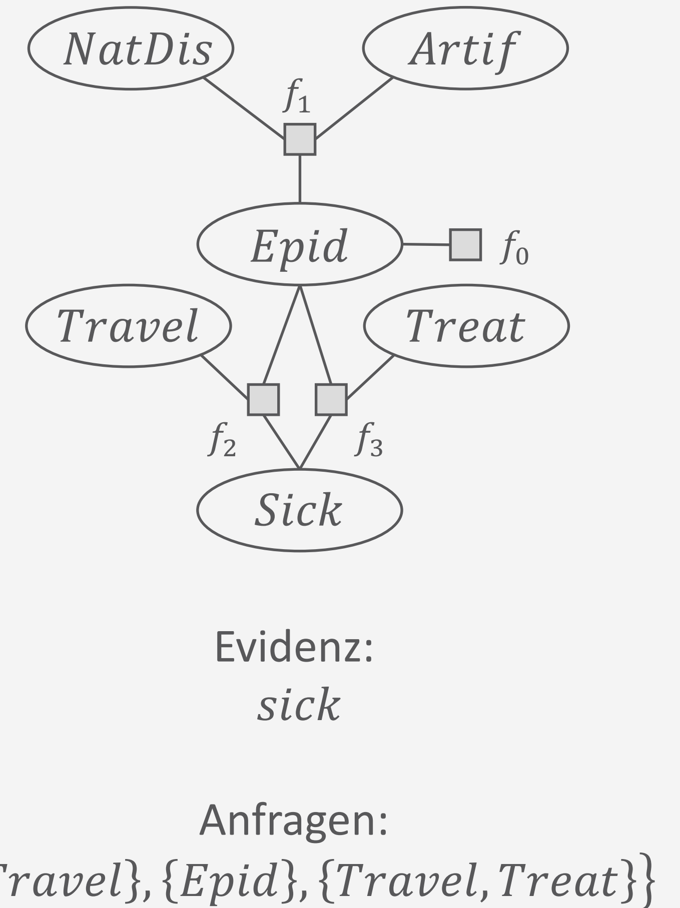
Junction Tree Algorithmus (JT)

- Eingabe
  - Faktormodell F
  - Evidenz $\boldsymbol{e}$
  - Anfragevariablen $\{\boldsymbol{Q}_i\}_{i=1}^m$
    - Gestellte Anfragen an F: $P(\boldsymbol{Q}_i \mid \boldsymbol{e})$ für $i \in \{1, \dots, m\}$
- JT besteht konzeptionell aus vier Schritten

1. Jtree J für das Eingabemodell F bauen
2. Evidenz $\boldsymbol{e}$ in J eingeben
3. Nachrichten in J schicken (*message passing*)
4. Anfragen $\{\boldsymbol{Q}_i\}_{i=1}^m$ beantworten
   - Umsetzung der Schritte in den blau-schattierten Boxen auf den vorhergehenden Folien

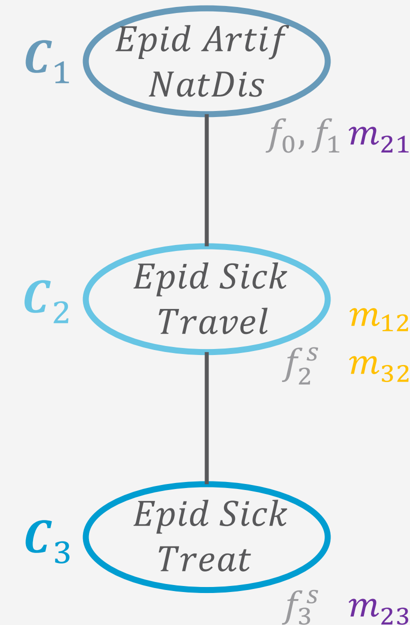
JT

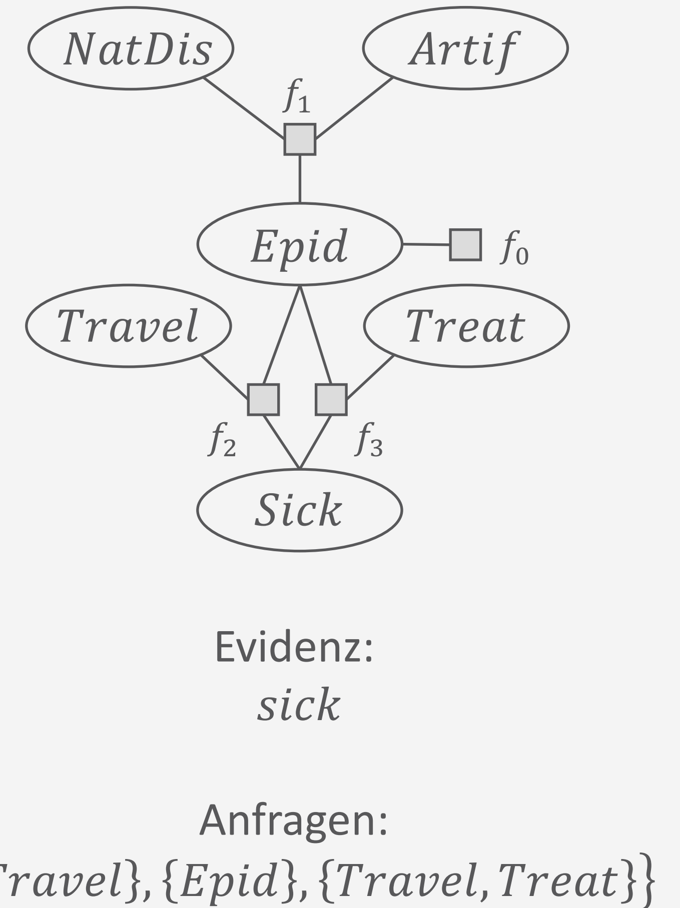




Eliminierungsfolgen

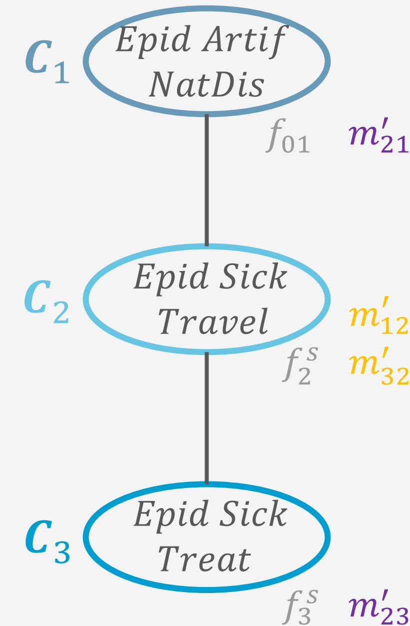
- Beim Bau von Jtrees über Dtrees wird quasi am Anfang einmal eine gute Eliminierungsfolge gesucht und in die Jtree-Struktur gegossen
  - Jtree repräsentiert dann eine Menge von Folgen der gleichen Baumweite
    - Gegeben eine Anfrage in Cluster $\boldsymbol{C}_i$ könnte man Nachrichten in Richtung $\boldsymbol{C}_i$ schicken und dort die verbleibenden Nichtanfragevariablen eliminieren: Rechnungen in JT = Rechnungen in VE unter einer dieser Folgen
- Eliminierungsfolgen für Nachrichten und Anfragen
  - Kleiner Suchraum, da nur das Cluster betrachtet werden muss
  - VE muss für jede Anfrage Folge suchen in einem Suchraum über alle Variablen, sofern es nicht gute Folgen abspeichert bzw. eine Heuristik lernt

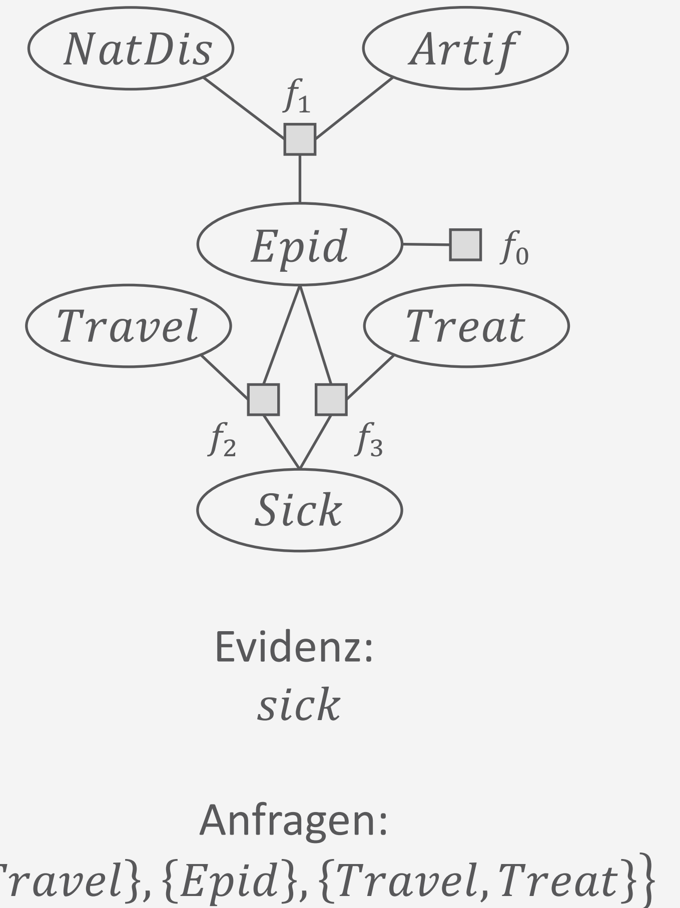




Minimierung der Größe der lokalen Modelle und Nachrichten

- Immer nur ein Faktor $f_i = \phi_i(\boldsymbol{C}_i)$ pro Cluster $\boldsymbol{C}_i$
  - Möglicherweise weniger Operationen
  - Möglicherweise größere Faktoren / versteckte Unabhängigkeiten, mehr Aufwand pro Operation
- Auswirkungen auf JT
  1. Jtree bauen
     - Faktoren der lokalen Modelle multiplizieren
  2. Evidenz behandeln: Pro Cluster mit $E \in \boldsymbol{C}_i$
     - $F_i \leftarrow F_i \setminus \{f_i\} \cup \textsc{absorb}(f_i, E, f_e)$
       - Bzw. $F_i \leftarrow F_i \setminus \{f_i\} \cup \textsc{multiply}(f_i, f_e)$
  3. Nachrichten senden:
     - Beim VE-Aufruf nur auf Normalisierung verzichten, aber nicht auf Multiplikation davor: $m'_{ij} = \phi(\boldsymbol{S}_{ij})$
  4. Anfragen beantworten: *keine Auswirkung*

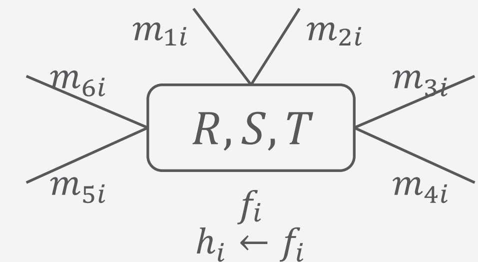




Mehr zum Nachrichtenversand

- Verfolgte Strategie hier: so genannte Shafer-Shenoy Architektur [Shafer and Shenoy, 1989]
  - Nachteil: viele Operationen (Multiplikationen) wiederholt möglich
    - Vor allem in Jtrees mit hohem Grad
      - Selbst bei nur einem Faktor pro Cluster und Nachricht
      - Beispiel: Pro Berechnung einer ausgehenden Nachricht ändert sich immer nur eine der berücksichtigten eingehenden Nachrichten
- Alternative: Hugin Architektur [Jensen et al., 1989]
  - Hugin-Faktor $h_i = \phi_i(\boldsymbol{C}_i)$ pro Cluster $\boldsymbol{C}_i$ als Produkt des lokalen Modells
  - Einkommende Nachrichten m_{ji} in den Faktor multiplizieren (und einzeln speichern): $h_i \leftarrow h_i \cdot m_{ji}$
  - Beim Berechnen der Nachricht m_{ij} zurück: $\text{VE}-\text{JT}\left(f_i \,/\, m_{ji}, S_{ij}, \emptyset, .\right)$
    - Reinmultiplizierte Nachricht wieder rausdividieren, danach Nicht-Separatoren aussummieren
      - Eine Division anstatt diverse Multiplikationen

$$m_{i1} \leftarrow \text{VE}-\text{JT}(\{f_i, m_{2i}, m_{3i}, m_{4i}, m_{5i}, m_{6i}\}, ., ., .)$$
$$m_{i2} \leftarrow \text{VE}-\text{JT}(\{f_i, m_{1i}, m_{3i}, m_{4i}, m_{5i}, m_{6i}\}, ., ., .)$$
$$m_{i3} \leftarrow \text{VE}-\text{JT}(\{f_i, m_{1i}, m_{2i}, m_{4i}, m_{5i}, m_{6i}\}, ., ., .)$$
$$m_{i4} \leftarrow \text{VE}-\text{JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{5i}, m_{6i}\}, ., ., .)$$
$$m_{i5} \leftarrow \text{VE}-\text{JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{4i}, m_{6i}\}, ., ., .)$$
$$m_{i6} \leftarrow \text{VE}-\text{JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{4i}, m_{5i}\}, ., ., .)$$



$h_i \leftarrow h_i \cdot m_{1i}$	$h_i \,/\, m_{1i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{2i}$	$h_i \,/\, m_{2i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{3i}$	$h_i \,/\, m_{3i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{4i}$	$h_i \,/\, m_{4i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{5i}$	$h_i \,/\, m_{5i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{6i}$	$h_i \,/\, m_{6i} \rightarrow$ VE-JT

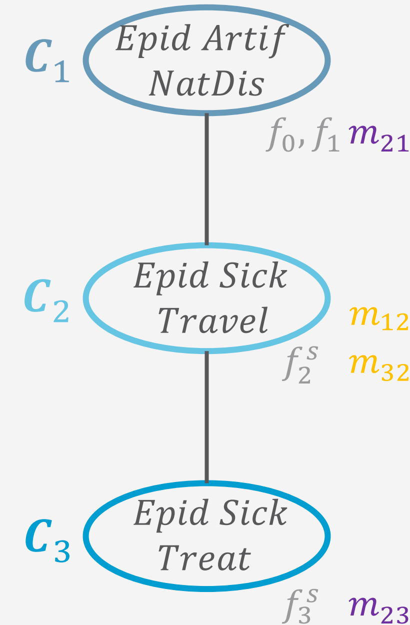
Glenn R. Shafer and Prakash P. Shenoy: Probability Propagation. In: Annals of Mathematics and Artificial Intelligence, 1990.
Finn V. Jensen and Steffen L. Lauritzen and Kristian G. Olesen: Bayesian Updating in Recursive Graphical Models by Local Computations. In: *Computational Statistics Quarterly*, 1990.

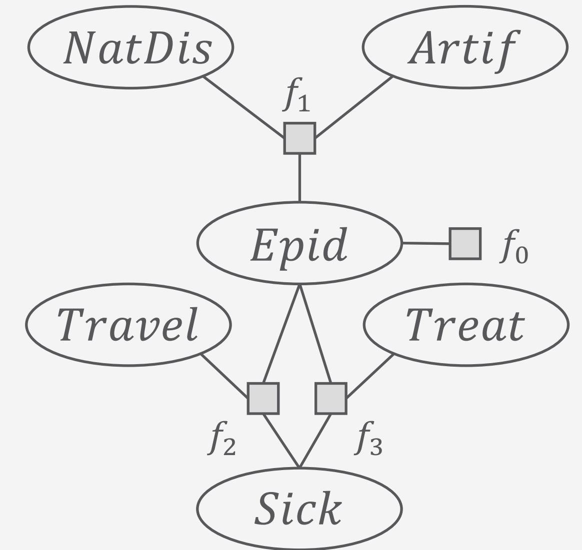
Junction Tree Algorithmus (JT)

- Eingabe
  - Faktormodell F
  - Evidenz $\boldsymbol{e}$
  - Anfragevariablen $\{\boldsymbol{Q}_i\}_{i=1}^{m}$
    - Gestellte Anfragen an F: $P(\boldsymbol{Q}_i \mid \boldsymbol{e})$ für $i \in \{1, \dots, m\}$

Was ist, wenn sich was ändert?

- JT besteht konzeptionell aus vier Schritten
  1. Jtree J für das Eingabemodell F bauen
  2. Evidenz $\boldsymbol{e}$ in J eingeben
  3. Nachrichten in J schicken (*message passing*)
  4. Anfragen $\{\boldsymbol{Q}_i\}_{i=1}^{m}$ beantworten



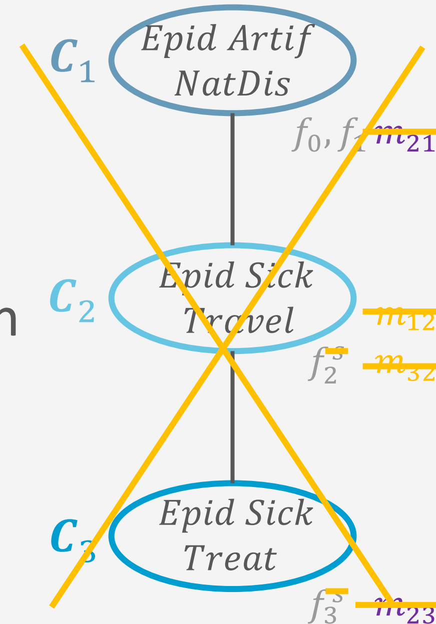


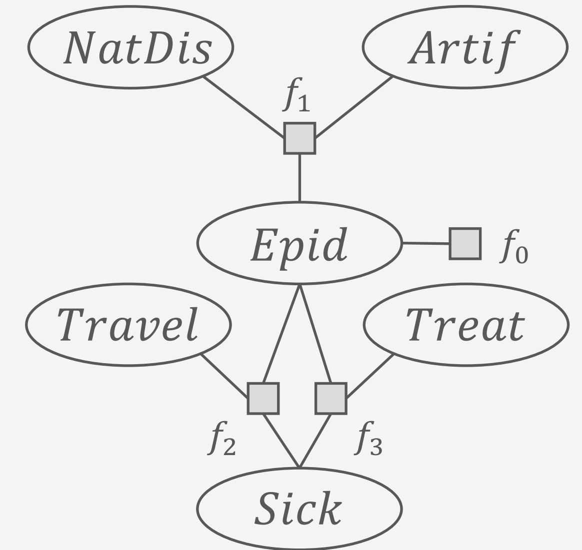
Evidenz: *sick*

Anfragen: $\{\{Travel\}, \{Epid\}, \{Travel, Treat\}\}$

Sich ändernde Eingaben

- Auch bekannt als adaptive Inferenz
  - Ziel: Nicht wieder von vorn anfangen
- Neue Anfragen $\{\boldsymbol{Q}'_i\}_{i=1}^{m}$
  - Bei Schritt 4 einsetzen: Anfragen in J beantworten
    - JT erlaubt *Online* Anfragebeantwortung
      - Anfragen nicht vorher bekannt → Strom von Anfragen
- Andere Evidenz $\boldsymbol{e}'$
  - Bei Schritt 2 starten: Originale lokale Modelle wiederherstellen, $\boldsymbol{e}'$ behandeln, Schritte 3+4 folgen
- Anderes Modell F
  - Bei Schritt 1 starten: Neuen Jtree bauen, Schritte 2-4 folgen



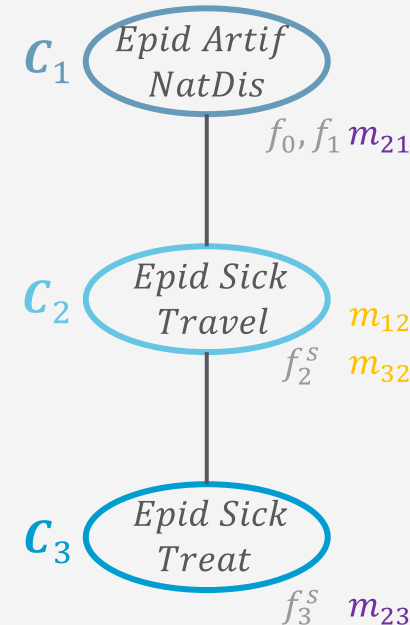


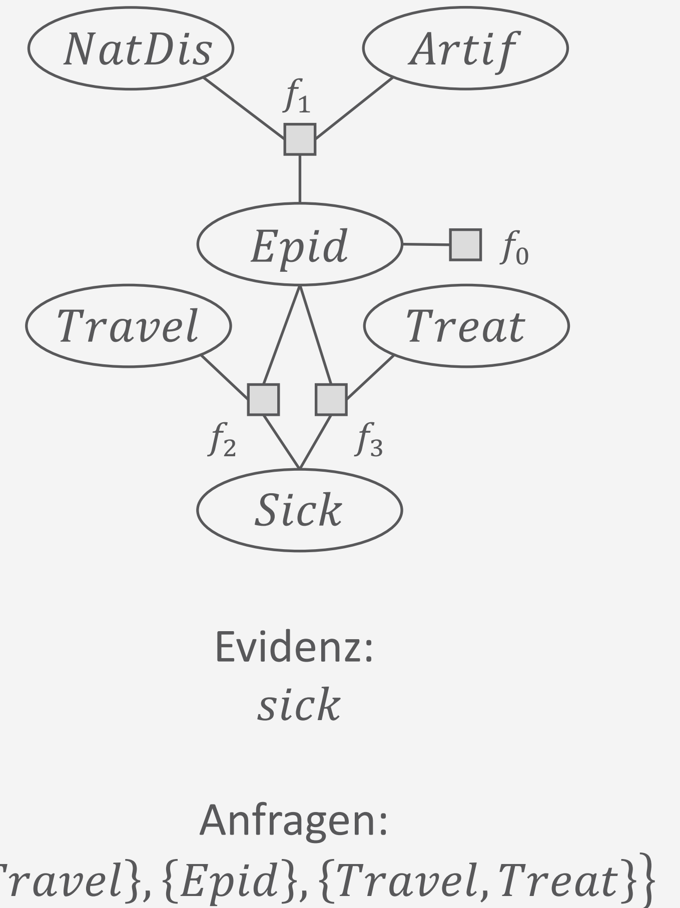
Wenn sich nur lokal Änderungen in $\boldsymbol{e}$ oder F ergeben, kann man adaptiv vorgehen.

Evidenz:
~~$sick$~~
$\neg travel, \neg sick$
Anfragen:
~~$\{\{Travel\}, \{Epid\}, \{Travel, Treat\}\}$~~
$\{\{NatDis\}, \{Artif\}\}$

Lokale Änderungen und adaptiver Nachrichtenversand

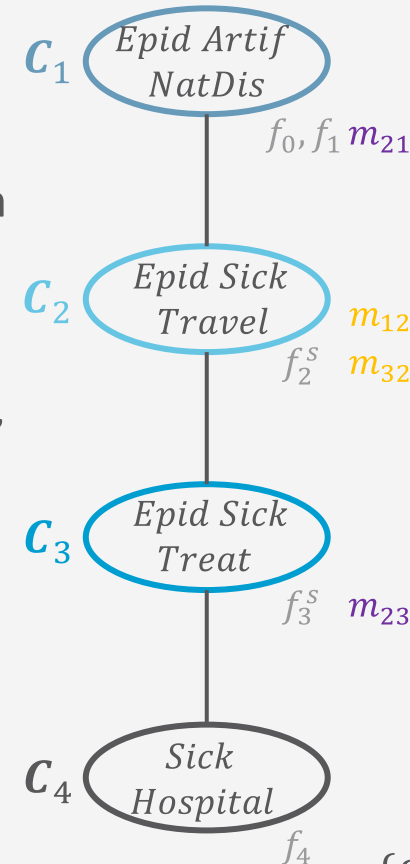
- Lokale Änderungen in F
  - Andere Zahlen in Faktoren
  - Faktoren fallen weg oder kommen hinzu
    - Wenn Zufallsvariablen hinzukommen, wegfallen oder Unabhängigkeiten verändern: Cluster hinzufügen, löschen, erweitern, verkleinern oder zusammenfügen
      - Jtree Eigenschaften beibehalten!
      - Lohnt nur bei lokal beschränkten Änderungen, sonst neu bauen
- Lokale Änderungen in $\boldsymbol{e}$
  - Nur bei betroffenen Clustern lokale Modelle wiederherstellen und Änderungen behandeln
- Adaptiver Nachrichtenversand
  - Wenn sich Änderungen am lokalen Modell oder einer eingehenden Nachricht ergeben, neue Nachrichten berechnen
    - Andernfalls: leere Nachricht schicken
  - Bei lokal beschränkten Änderungen ermöglicht das bis zur Hälfte der Nachrichten einzusparen

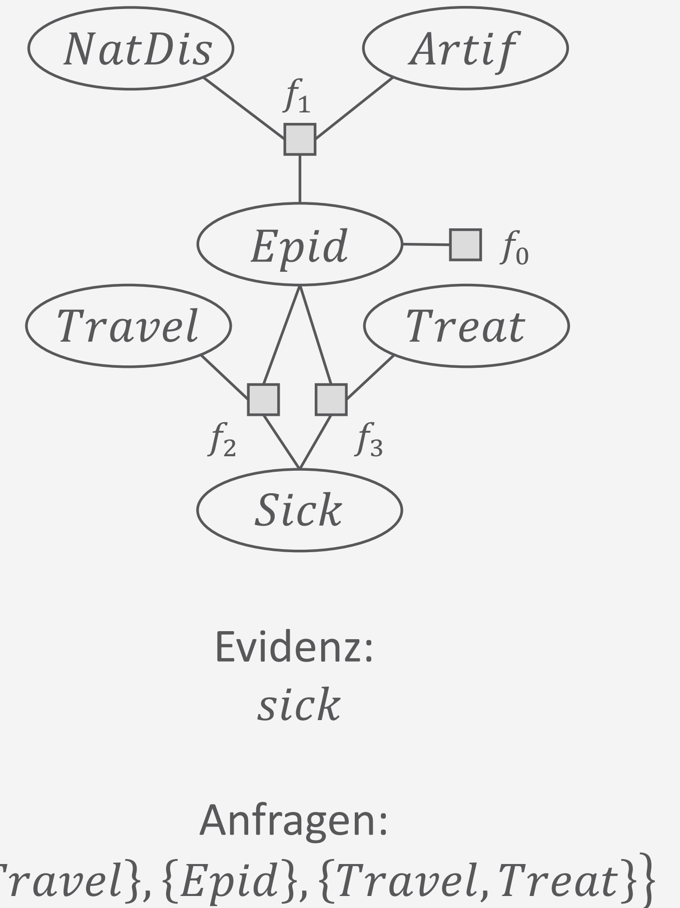




Lokale Änderungen und adaptiver Nachrichtenversand

- Beispiele:
  - Andere Zahlen in f_0
    - Nachrichten m_{32} und m_{21} valide, da keine Änderung im lokalen Modell oder einer eingehenden Nachricht, die in Berechnung eingeflossen ist
    - Nachrichten m_{12} neu berechnen, da sich f_0 geändert hat, und m_{23} neu berechnen, da sich m_{12} geändert hat, was in die Berechnung von m_{23} einfließt
  - Neuer Faktor $f_4 = \phi_4(Sick, Hospital)$
    - Neues Cluster $\boldsymbol{C}_4 = \{Sick, Hospital\}$ mit $F_4 = \{f_4\}$ als Nachbar von $\boldsymbol{C}_3$ ($\boldsymbol{C}_2$ ginge auch)
    - Evidenz in $\boldsymbol{C}_4$ absorbieren
    - Nachrichten m_{12}, m_{23} valide
    - Nachricht m_{43} und m_{34} erstmalig berechnen
    - Nachrichten m_{32} neu berechnen, da m_{43} neu, und m_{21} neu berechnen, da Änderung in m_{32}





Komplexität

- JT Laufzeitkomplexität folgt der VE Laufzeitkomplexität
  - Zur Erinnerung: VE Komplexität bei einem Dtree T mit n_T inneren Knoten
    $$O(n_T \cdot r^w)$$
    - $r = \max_{R \in \boldsymbol{R}} |\text{Val}(R)|$ als größte Domäne
    - $w = \max_{T_i \in \text{Desc}(T)} |cluster(T_i)|$ als Baumweite bzw. größtes Cluster
  - Jtree J folgt aus Dtree T über die Cluster → Baumweite in JT und VE äquivalent
    - $w = \max_{T_i \in \text{Desc}(T)} |cluster(T_i)| = \max_{\boldsymbol{C}_i \in J} |\boldsymbol{C}_i|$
    - Durch Minimierung von J unterschiedliche Anzahl von Knoten
- Rechnungen in JT-Schritten sind Aufrufe von VE
  - VE Operationen durch $O(r^w)$ gedeckelt
- Zur Vereinfachung, Betrachtung von einem Jtree mit einem Faktor pro Cluster und von Anfragen $\{Q_i\}_{i=1}^m$

Schrittweise Komplexität von JT

- Jtree Bau: *vernachlässigbar*
  - Abhängig von der Anzahl der Knoten n_J bzw. n_T, keine Berechnungen abhängig von r^w
- Evidenz behandeln: $O(e \cdot n_J \cdot r^w)$
  - Absorbieren von Evidenz an jedem Knoten (worst-case) → $n_J \cdot O(r^w)$
  - Für jeden Evidenzfaktor, e viele → $e \cdot O(n_J \cdot r^w)$
- Nachrichtenversand: $O(n_J \cdot r^w)$
  - Berechnen einer Nachricht = Beantwortung einer Anfrage an ein Cluster mit $O(r^w)$
  - Zwei Nachrichten per Kante, $n_J - 1$ Kanten in J (azyklisch!) → $O(n_J \cdot r^w)$
- Anfragebeantwortung: $O(m \cdot r^w)$
  - Beantwortung einer Anfrage an ein Cluster mit $O(r^w)$
  - m Anfragen → $m \cdot O(r^w)$

Schrittweise Komplexität von JT

- Jtree Bau: *vernachlässigbar*
- Evidenz behandeln: $O(e \cdot n_J \cdot r^w)$
- Nachrichtenversand: $O(n_J \cdot r^w)$
- Anfragebeantwortung: $O(m \cdot r^w)$
- Zusammen: $O\left((e \cdot n_J + m) \cdot r^w\right)$
  - $(e \cdot n_J \cdot r^w) + (n_J \cdot r^w) + (m \cdot r^w) = (e \cdot n_J + n_J + m) \cdot r^w = \left((e-1) \cdot n_J + m\right) \cdot r^w$
- Da VE Komplexitätsbetrachtung mit $O(n_T \cdot r^w)$ ohne explizite Betrachtung von Evidenz war, vergleichbare JT Laufzeitkomplexität:

$$O\left((n_J + m) \cdot r^w\right)$$

Laufzeitkomplexität von JT

- Laufzeitkomplexität von JT ohne Evidenz:

$$O\left((n_J + m) \cdot r^w\right)$$

  - VE Komplexität $O(n_T \cdot r^w)$ ungefähr die Komplexität vom Nachrichtenversand mit $O(n_J \cdot r^w)$
    - Allerdings effektiv doppelt so viel Aufwand, da zwei Mal Nachrichten pro Kante kommen
    - Lohnt sich oft ab der zweiten bis dritten Anfrage
  - VE Komplexität für m Anfragen: $O(m \cdot n_T \cdot r^w)$
    - Bei Anfragen an alle Zufallsvariablen, $|\boldsymbol{R}| = n = n_T$: $O(n^2 \cdot r^w)$ vs. $O(n \cdot r^w)$
      - Da $n \geq n_J$: $O\left((n_J + n) \cdot r^w\right) \leq O\left((n + n) \cdot r^w\right) = O(2n \cdot r^w) \rightarrow O(n \cdot r^w)$
  - Wenn man die Vorverarbeitungsschritte wegabstrahiert, dann bleibt $O(n_T \cdot r^w)$ vs. $O(r^w)$ pro Anfrage
    - *Online* Anfragebeantwortung unabhängig von der Anzahl der Knoten insgesamt

JT Implementierung aufbauend auf der VE Implementierung: Ausgabe

- Aufruf wie bisher, aber mit `-e jt.JTEngine`
- Ausgabe:

```
Using fixed random seed for repeatability.
Parsing from: examples/epid.blog
.........................................
.................... JT .................
Constructing inference engine of class jt.JTEngine
-------------------------------------------------------
Trial 0:

======== Query Results =========
Distribution of values for Travel
0.75 false
0.25 true
```

(Wichtig für Sampling)

(Wichtig für Sampling)

Anfrageergebnisse

Dahinter folgen noch ein paar Zahlen für die Statistik, die vor allem für die *Lifting* Varianten wichtig sind.

JT Implementierung aufbauend auf der VE Implementierung: Ausgabe

- Ein Blick auf die Zahlen für die Statistik:

Anzahl Faktoren
(Wichtig für *Lifting*)
Anzahl Anfragen
Anzahl Beobachtungen
Anzahl Jtree-Knoten, n_J
Baumweite w
Laufzeiten pro Schritt / Anfrage und insgesamt

```
engine jt.JTEngine
name epid
|G| 4
|gr| 4
|Q| 1
|E| 1
size 3
width 3
Split times
t_0 4635094 ns 4 ms
t_1 222711 ns 0 ms
t_2 2470327 ns 2 ms
t_3 379333 ns 0 ms
t_total 18456408 ns 18 ms
```

Schritt 1: Jtree bauen
Schritt 2: Evidenz
Schritt 3: Nachrichten
Anfrage beantworten

```
**TIME**2569628
======== Query Results ========
Distribution of values for Travel
0.75 false
0.25 true
```

VE Ausgabe
- VE Laufzeit in etwa Laufzeit von Schritt 3
- JT Laufzeit Bruchteil von VE Laufzeit für Anfrage (≈ 0.15), aber Overhead da

Dahinter folgen noch wieder ein paar Zahlen für die Statistik, die für die *Lifting* Varianten wichtig sind.

```
========= Query Results =========
Distribution of values for Travel
0.75 false
0.25 true

Distribution of values for Epid
0.99250110277900 3 false
0.0074988972209691 3 true

Distribution of values for Treat
0.5964380238200264 false
0.4035619761799735 true

Distribution of values for Artif
0.8323775915306573 false
0.1676224084693427 3 true

Distribution of values for NatDis
0.7767975297750331 false
0.2232024702249669 2 true
```

- Bei mehreren Anfragen:

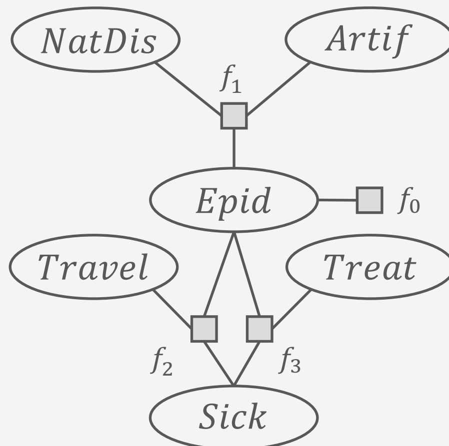
```
engine jt.JTEngine
name epid_5
|G| 4
|gr| 4
|Q| 5        <- Anzahl Anfragen jetzt 5
|E| 1
size 2
width 3
Split times
t_0 4574037 ns 4 ms   <- Schritt 1: Jtree bauen
t_1 262208 ns 0 ms    <- Schritt 2: Evidenz
t_2 2562122 ns 2 ms   <- Schritt 3: Nachrichten
t_3 364780 ns 0 ms  ┐
t_4 208314 ns 0 ms  │   Anfrage 1
t_5 211836 ns 0 ms  ├<- bis
t_6 204041 ns 0 ms  │   Anfrage 5
t_7 169691 ns 0 ms  ┘
t_total 22161632 ns 22 ms
```

JT Implementierung: Schrittweise Laufzeiten (ohne Evidenz)

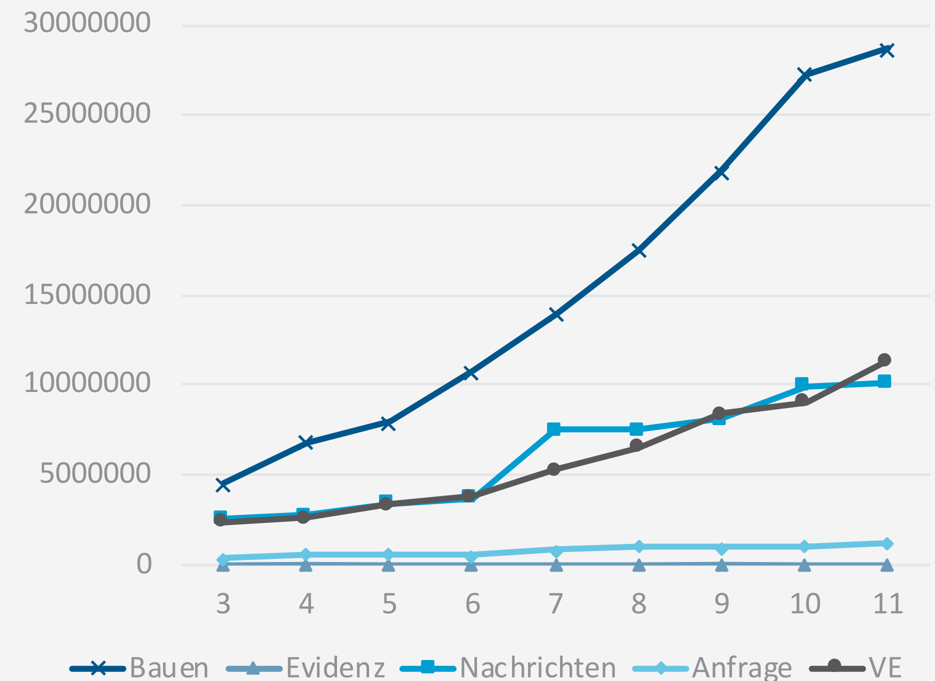
- Laufzeitkomplexität von JT:

$$O\left(\left(n_J + m\right) \cdot r^w\right)$$

- Verhalten bei steigendem
  - w: Immer mehr neue Zufallsvariablen zu Faktoren hinzufügen



- Laufzeiten:
  - Bauen braucht am meisten Zeit

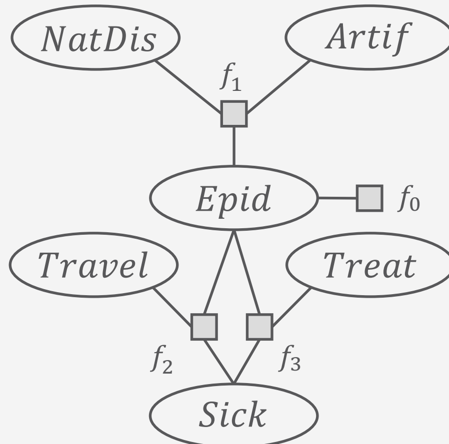


JT Implementierung: Schrittweise Laufzeiten (ohne Evidenz)

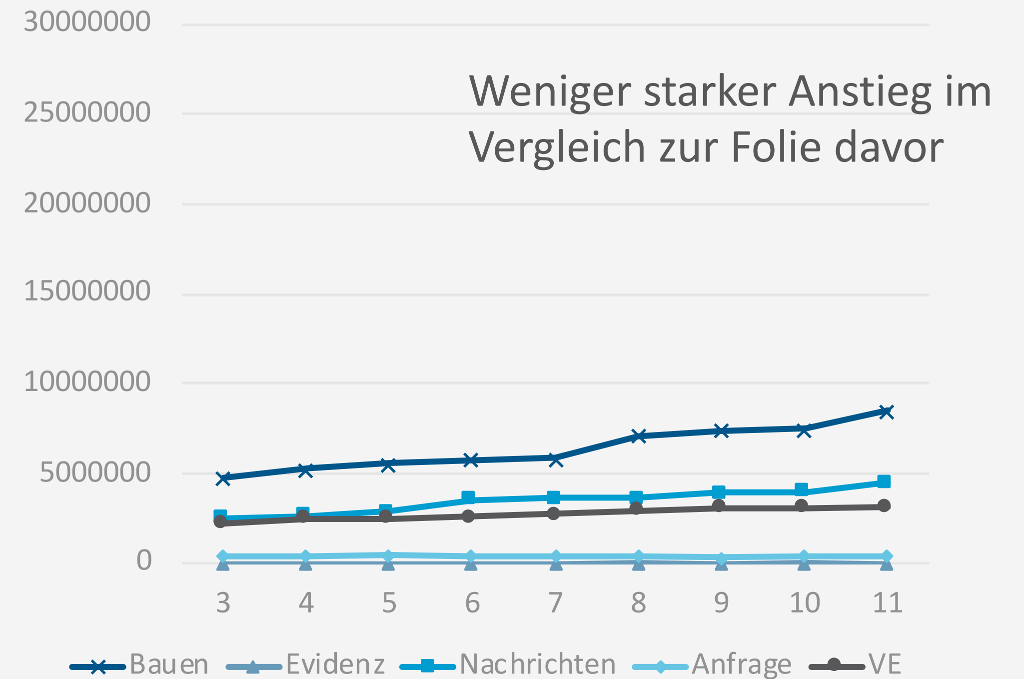
- Laufzeitkomplexität von JT:

$$O\left((n_J + m) \cdot r^w\right)$$

- Verhalten bei steigendem
  - n_J: Immer mehr Faktoren an $Epid$ mit zwei neuen Zufallsvariablen anfügen → bilden eigene Cluster



- Laufzeiten:
  - Bauen braucht am meisten Zeit

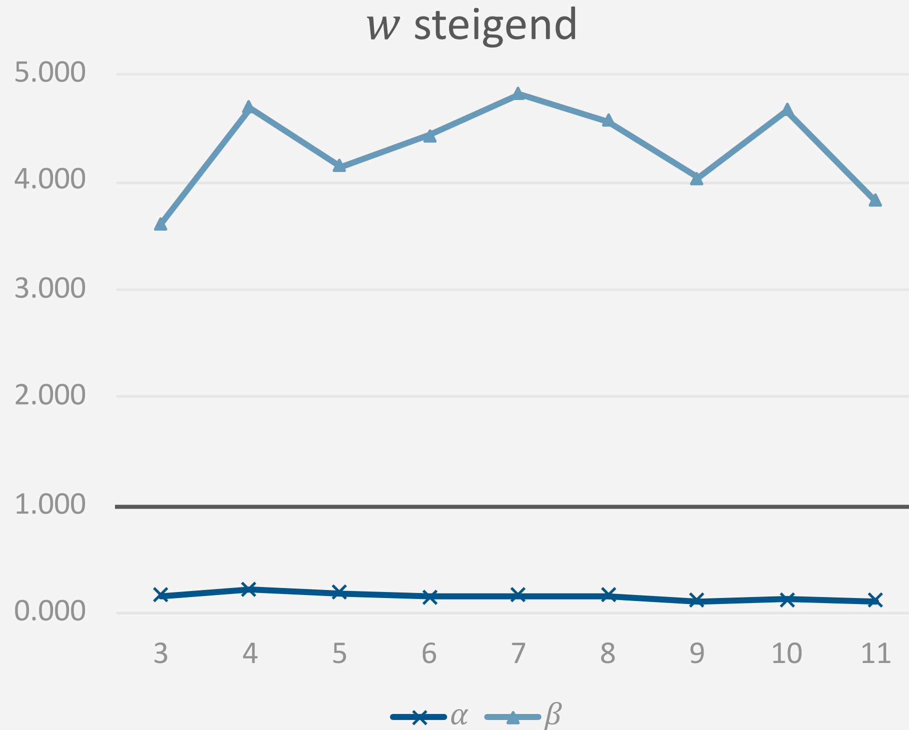


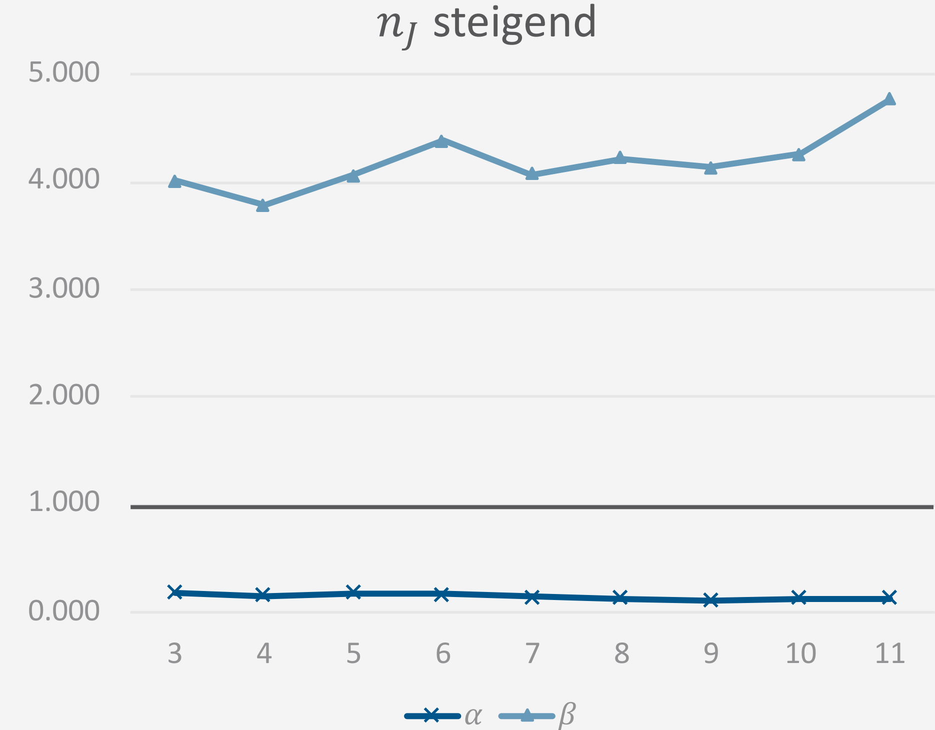
Vergleich Laufzeiten Implementierung

- Trade-off für Multi-Query-Algorithmen
  - Overhead, der gegengerechnet werden muss (Modell wird in eine Hilfsstruktur *kompiliert*)
  vs.
  - Kürzere Laufzeit bei der Beantwortung von individuellen Anfragen
- Mit
  - $t_{q,cpl}$ Laufzeit zur Beantwortung einer Anfrage mit einem Algorithmus mit Kompilierung
  - $t_{q,uncpl}$ Laufzeit zur Beantwortung einer Anfrage mit einem Algorithmus ohne Kompilierung
  - $t_{c,cpl}$ Laufzeit der Kompilierung des Algorithmus mit Kompilierung
  - Verhältnis der Anfragebeantwortungszeiten: $\alpha = \frac{t_{q,cpl}}{t_{q,uncpl}}$
  - Anzahl der Anfragen, die es braucht, um den Overhead aufzuwiegen: $\beta = \frac{t_{c,cpl}}{t_{q,uncpl} - t_{q,cpl}}$
    - Nur sinnvoll bei $\alpha > 1$

Vergleich Laufzeiten Implementierung

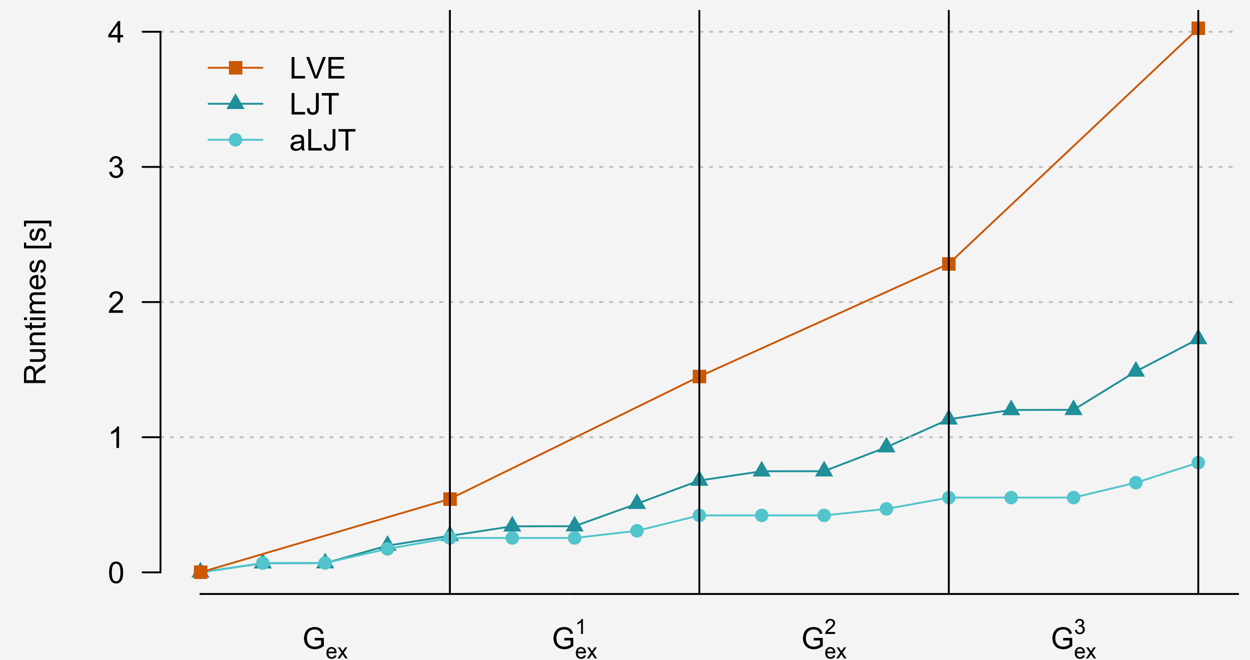
- Vergleichskennzahlen: $\alpha = \frac{t_{q,cpl}}{t_{q,uncpl}}$ und $\beta = \frac{t_{c,cpl}}{t_{q,uncpl} - t_{q,cpl}}$





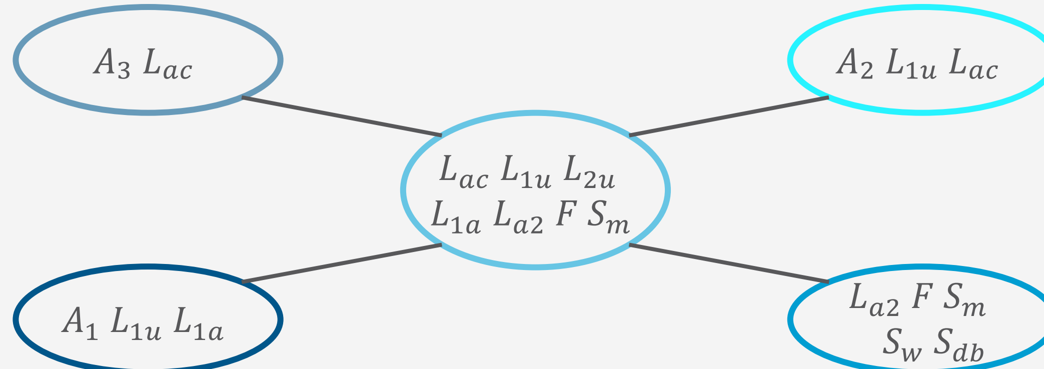
Vergleich Laufzeiten Implementierung

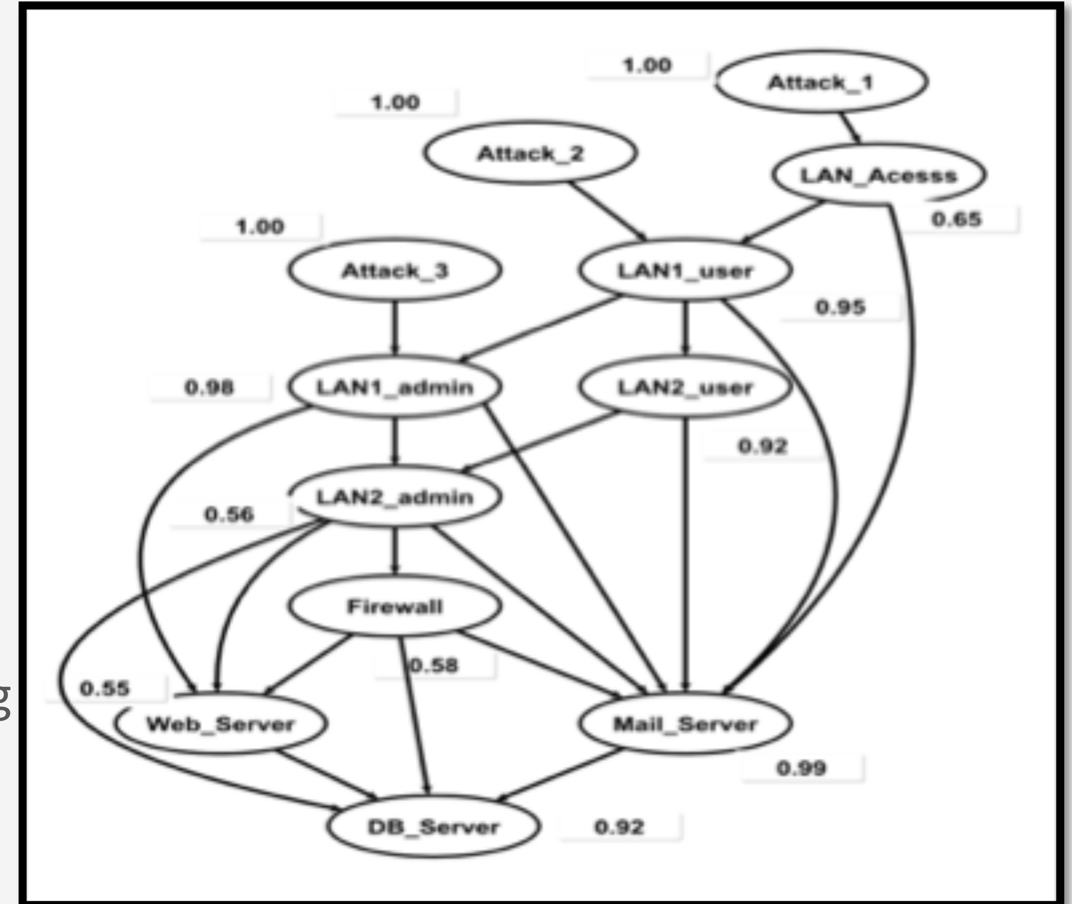
- Adaptive Inferenz
  - *Überlesen Sie bitte das ‘L‘ in der Graphik: Aussagen stimmen so auch für VE und JT bzw. aJT als adaptive Variante von JT*
  - Drei lokale Änderungen nacheinander
    - Neuen Faktor hinzufügen
    - Faktor ersetzen
    - Zusätzliche Beobachtungen einfügen
  - Jeweils zwei Anfragen
- VE startet jeweils mit dem ganzen Modell
- JT fängt bei Schritt 1 bzw. Schritt 2 an
- aJT geht adaptiv pro Änderung vor



Anwendung: *Bayesian Attack Graphs*

- Anfragen: $P(A_1)$, $P(A_2)$, $P(A_3)$, $P(L_{ac})$, $P(L_{1u})$, $P(L_{2u})$, $P(L_{1a})$, $P(L_{2a})$, $P(F)$, $P(S_w)$, $P(S_m)$, $P(S_{db})$
- Jtree





  - Anfragen farblich möglichen Clustern für Beantwortung zusortiert
  - `Mail_Server` Knoten ungünstig für effiziente Inferenz

Luis Muñoz-González, Daniele Sgandurra, Martín Barrère, and Emil C. Lupu: Exact Inference Techniques for the Analysis of Bayesian Attack Graphs. In: *IEEE Transactions on dependable and Secure Computing*, 2017.
Abbildungen ebenda.

Zwischenzusammenfassung

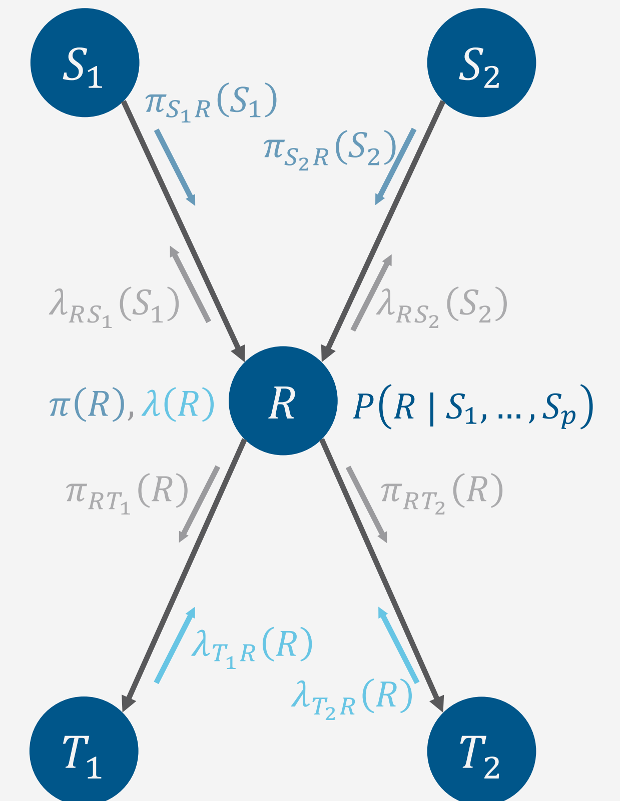
- Junction Tree als Hilfsstruktur
  - Cluster (Cliquen) von Zufallsvariablen mit Separatoren zwischen ihnen
- JT Schritte
  - Junction Tree bauen: Cluster vom Dtree
  - Evidenz abhandeln: Absorption in jedem Cluster, welches direkt von Evidenz betroffen ist
  - Nachrichten versenden: Cluster durch Zwei-Phasen-Nachrichtenversand unabhängig machen, lokales Modell und Nachrichten anderer Nachbarn in eine Nachricht über den Separator mittels VE verarbeiten
  - Anfragen beantworten: Cluster bzw. Subtree über die Anfragevariablen finden und deren lokalen Modelle und Nachrichten von außerhalb als Grundlage für VE nutzen
- Komplexität: Nachrichtenversand entsprecht VE Anfragenbeantwortung,
  JT Anfragenbeantwortung unabhängig von n_T

Pearl‘s Probability Propagation

Geschichtsstunde

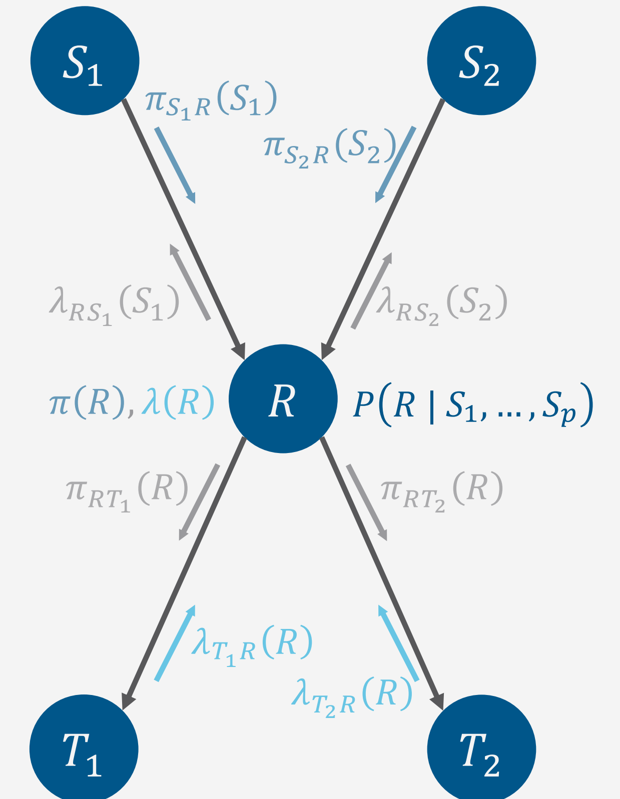
Geschichtsstunde: Pearl‘s Probability Propagation (PP)

- PP erster Inferenzalgorithmus
  - Für *Polytree* BNs
  - Auch *belief propagation* (BP) genannt
- Berechnet $P(R \mid \boldsymbol{e})$ für jedes $R \in \boldsymbol{R}$ durch Versand von Nachrichten entlang der Kanten im BN
  - Zwei Typen von Nachrichten: an Eltern bzw. Kinder
  - Initialisierung setzt Evidenz in Nachrichten
    - Evidenz wird Schritt um Schritt durch das BN propagiert
  - Nachrichten versendet, bis sich keine Änderungen ergeben
  - Danach hat jeder Knoten seine eigene Marginalverteilung als normalisiertes Produkt $\mathrm{BEL}(R)$ der finalen Nachrichten und der lokalen CPT



PP: Nachrichten

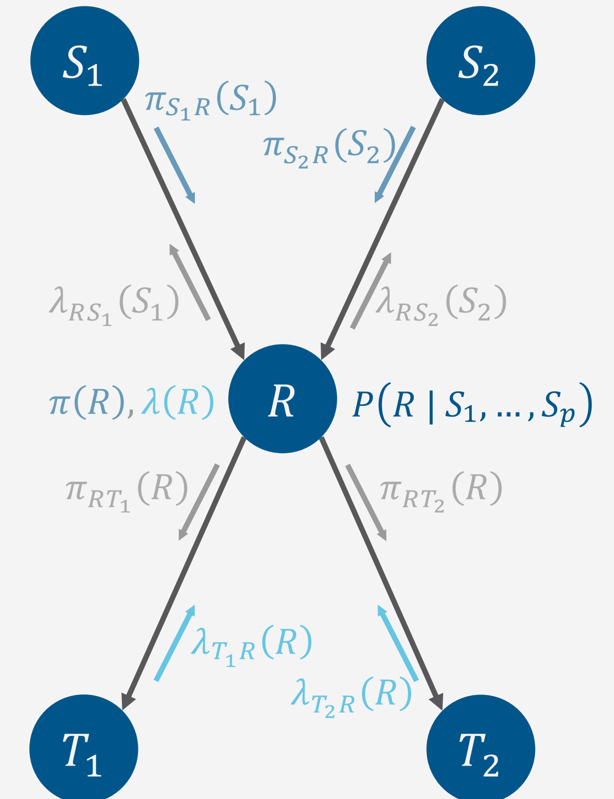
- Knoten R mit Eltern $S_1, \dots, S_p$ und Kindern $T_1, \dots, T_c$
- Versand von Knoten R an einen Knoten
  - Wenn „alle anderen“ Nachrichten schon angekommen sind
- Berechnung:
  - Multiplizieren der CPD von R mit den erhaltenen Nachrichten, anschließendes Aussummieren aller Zufallsvariablen im Produkt, die Empfänger nicht in seiner CPD hat
- *Vergleich Jtree*
  - *Versand an einen Nachbarknoten, wenn Nachrichten von allen anderen Nachbarn da sind*
  - *Kombination des lokalen Modells und aller anderen Nachrichten, anschließendes Aussummieren des Separators*



PP: Nachrichten

- Versand Nachricht π_{RT} an Kind T:
  - Kind T mit CPD $P(T \mid R_1, \dots, R, \dots, R_t)$
  - Wenn alle Nachrichten der Eltern und der anderen Kinder bei R angekommen sind
    - Unterschiedliche Kindernachrichten einzuberechnen
- Berechnung
  - Gemeinsame Variable: R
  - Für π_{RT} aussummieren von allen Eltern $S_1, \dots, S_p$
- Lokales π zur Wiederverwendung / Zusammenfassung

$$\pi(R) = \sum_{s_1,\dots,s_p \in \mathrm{Val}(\mathrm{Pa}(R))} P(R \mid s_1, \dots, s_p) \prod_{S \in \mathrm{Pa}(R)} \pi_{SR}(S)$$

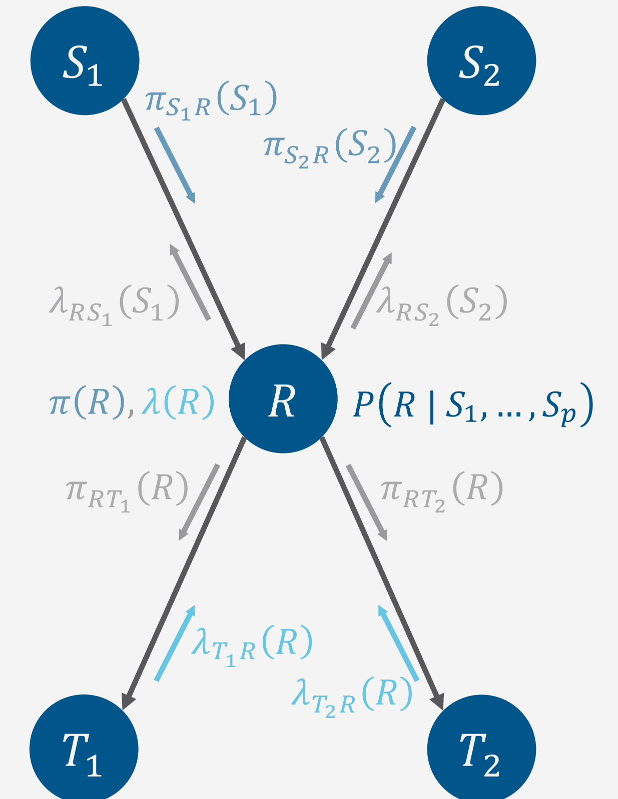


PP: Nachrichten

- Berechnung der Nachricht $\pi_{RT}(R)$ an Kind T:

$$\pi_{RT}(R) = \alpha \cdot \pi(R) \prod_{T' \in \mathrm{Ch}(R), T' \neq T} \lambda_{T'R}(R)$$

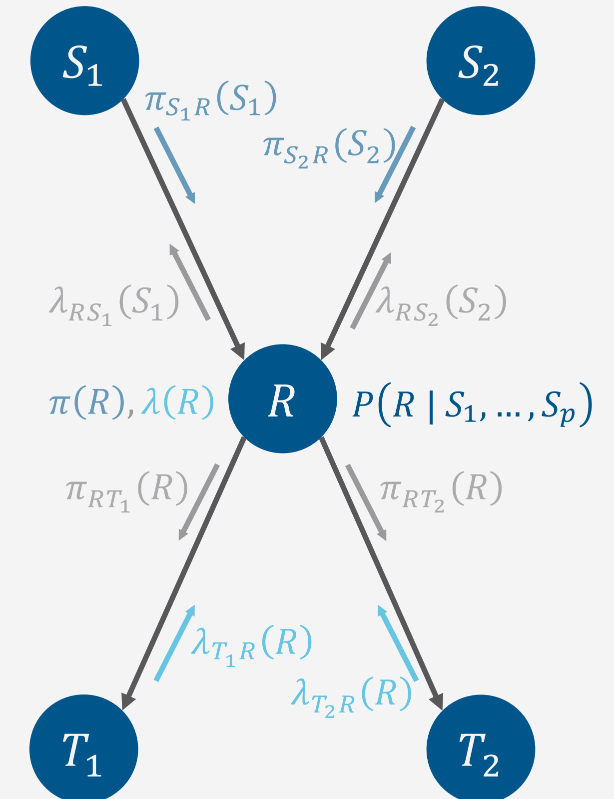
  - $\pi(R)$: alle Elternnachrichten und CPD
    - Beinhaltet Aussummierung der Eltern
  - Großes Produkt: Nachrichten $\lambda_{T'R}$ der anderen Kinder T'
  - α Normalisierung



PP: Nachrichten

- Versand Nachricht λ_{RS} an Elter S:
  - Elter S mit CPD $P(S \mid U_1, \dots, U_s)$
  - Wenn alle Nachrichten der Kinder und der anderen Eltern bei R angekommen sind
    - Unterschiedliche Elternnachrichten einzuberechnen
- Berechnung
  - Gemeinsame Variable (unterschiedlich pro Elter): S
  - Für λ_{RS} aussummieren von R, Eltern $S_1, \dots, S_p \setminus S$
- Lokales λ zur Wiederverwendung / Zusammenfassung

$$\lambda(R) = \prod_{T \in \mathrm{Ch}(R)} \lambda_{TR}(R)$$

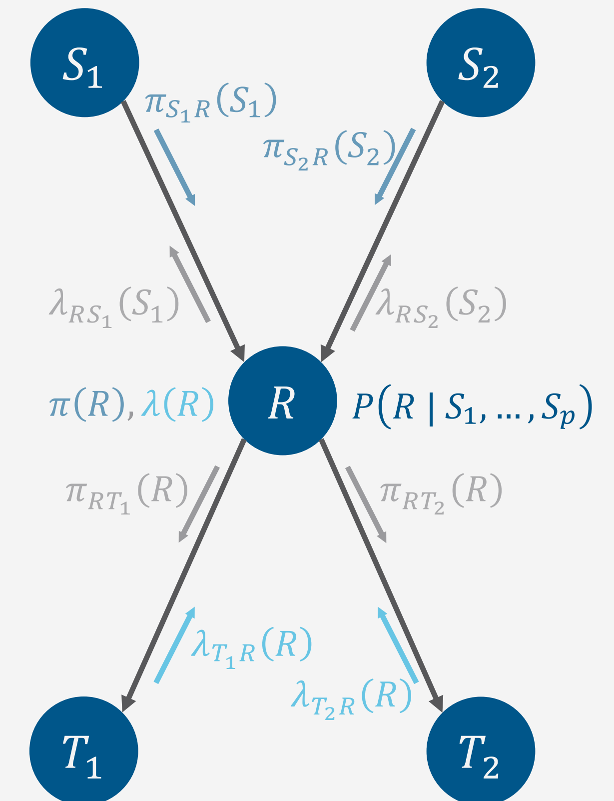


PP: Nachrichten

- Berechnung der Nachricht $\lambda_{RS}(S)$ an Elter S:

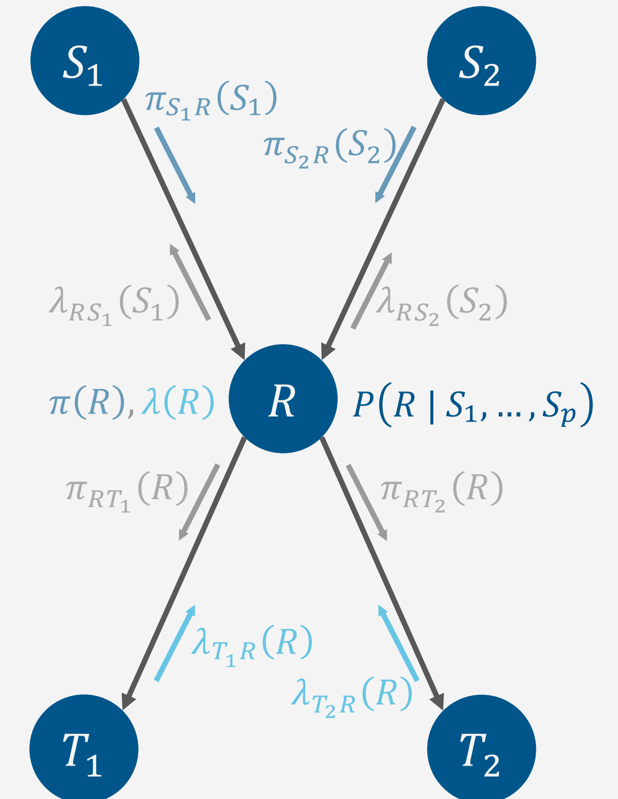
$$\lambda_{RS}(S) = \sum_{\substack{r \in \\ \mathrm{Val}(R)}} \lambda(r) \sum_{\substack{s_1,\dots,s_q \in \\ \mathrm{Val}(\mathrm{Pa}(R) \setminus \{S\})}} P(r \mid s_1, \dots, s_q) \prod_{i=1}^{q} \pi_{S_i R}(s_i)$$

  - $\lambda(r)$: alle Nachrichten der Kinder
  - Inneres Produkt:
    alle Nachrichten $\pi_{S_i R}(s_i)$ der anderen Eltern $\mathrm{Pa}(R) \setminus \{S\}$
  - Innere Summe: Aussummierung der anderen Eltern $\mathrm{Pa}(R) \setminus \{S\}$
  - Äußere Summe: Aussummierung der Variable R selbst



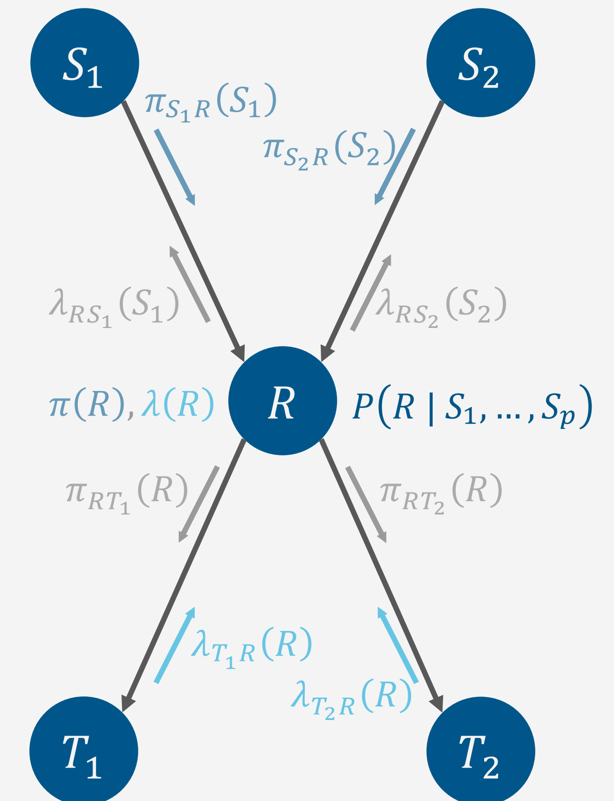
PP: Algorithmus – Initialisierung

- Für alle $E = e$ in $\boldsymbol{E} = \boldsymbol{e}$:
  - $\pi(r) = 1$ wenn $r = e$; 0 sonst
  - $\lambda(r) = 1$ wenn $r = e$; 0 sonst
- Für alle Knoten ohne Eltern und nicht in $\mathrm{rv}(\boldsymbol{E})$ do
  - $\pi(r) = P(r)$ (Apriori-Wahrscheinlichkeiten)
- Für alle Knoten ohne Kinder
  - $\lambda(r) = 1$ (uniform; wird am Ende normalisiert)



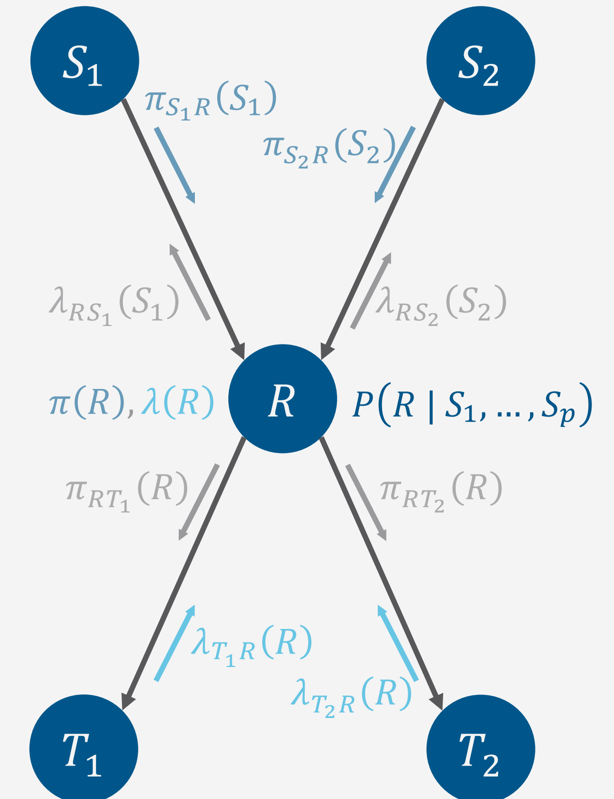
PP: Algorithmus – Prozedere

- Iteriere bis keine Änderungen mehr passieren
  - Für jeden Knoten R
    - Wenn R alle π Nachrichten seiner Eltern erhalten hat, berechne $\pi(R)$
    - Wenn R alle λ Nachrichten seiner Kinder erhalten hat, berechne $\lambda(R)$
  - Für jeden Knoten R
    - Wenn $\pi(R)$ berechnet ist und R alle λ Nachrichten seiner Kinder außer von T erhalten hat, berechne $\pi_{RT}(R)$ und sende es an T
    - Wenn $\lambda(R)$ berechnet ist und R alle π Nachrichten seiner Eltern außer von S erhalten hat, berechne $\lambda_{RS}(S)$ und sende es an S
- Für jeden Knoten R
  - Berechne $\mathrm{BEL}(R) = \pi(R) \cdot \lambda(R)$ und normalisiere
    - $\mathrm{BEL}(R)$ entspricht $P(R \mid \boldsymbol{e})$



PP: Analyse

- Komplexität von PP
  - Konvergiert in Zeit proportional zum Durchmesser des Polytrees
    - Durchmesser = größter Abstand zwischen zwei Knoten, max. n
  - Arbeit pro Knoten ist proportional zur Größe der CPD
  - Laufzeitkomplexität: $O\left(n \cdot r^{\deg(B)+1}\right)$
- PP funktioniert nur, weil das BN *azyklisch* ist
  - Bei Zyklen würde Information doppelt ankommen
  - Idee: Zyklen in allgemeinen BNs in Cluster zusammenfassen um einen azyklischen Graphen zu erhalten → Jtree
  - PP auf beliebigen Graphen → Loopy belief propagation
    - Korrekt in azyklischen Graphen, beliebig falsch sonst

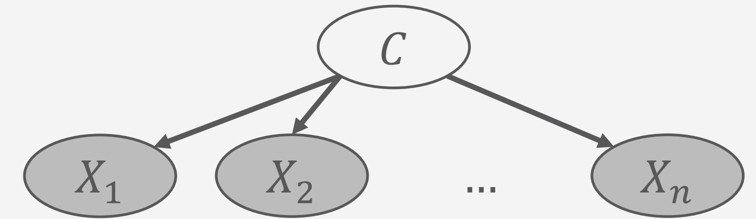


Zwischenzusammenfassung und Vergleich

- PP: Vorläufer zu JT
  - Anfragenbeantwortung auf Polytree BNs
    - Möglich für Faktormodelle mit entsprechender Struktur zu adaptieren
  - Beantwortet immer Anfragen an alle Zufallsvariablen im BN gegeben der gleichen Evidenz
  - Komplexität: Baumweite = maximale Anzahl an Eltern
- Vergleich JT
  - Anfragenbeantwortung auf generellen Faktormodellen / BNs
  - Berechnet Anfragen an Menge von Zufallsvariablen gegeben der gleichen Evidenz auf kleineren Modellen als das Ursprungsmodell (im Gegensatz zu VE)
    - Bei Polytree BNs tun JT und PP in etwa das gleiche
  - Komplexität: Baumweite von unten durch die maximale Anzahl an Nachbarn / Eltern begrenzt

Anwendungen

- Naive Bayes Klassifizierer: $P(C \mid \boldsymbol{x})$
  - Nichts auszusummieren: $\boldsymbol{S} = \{C\}, \boldsymbol{T} = \mathrm{rv}(\boldsymbol{x})$ und damit
    $$\boldsymbol{U} = \boldsymbol{R} \setminus \boldsymbol{S} \setminus \boldsymbol{T} = \emptyset$$



  - Eigentlich will man $\arg\max_{c \in \mathrm{Val}(C)} P(C \mid \boldsymbol{x})$ haben
    - Zustandsanfrage gegeben Evidenz:
      „Was ist der wahrscheinlichste Zustand der Zufallsvariablen, der die Evidenz hervorgerufen haben könnte?“
    - Variablen ausmaximieren anstatt aussummieren

Überblick: 3. Exakte Inferenz in episodischen PGMs

A. *Einzelanfragen: Variableneliminierung (VE)*
- Algorithmus, Operatoren für Wahrscheinlichkeitsanfragen
- Dekompositionsbäume, Komplexität

B. *Multi-Anfragen: Junction Tree (Cliquen-Bäume) Algorithmus (JT)*
- Cliquen, Junction Tree als Hilfsstruktur, Vorverarbeitung und Anfragebeantwortung
- Zusammenhang mit VE, Komplexität
- *Geschichtsstunde:* *Pearl‘s Probability Propagation (PP)* auf Polytree BNs

C. ***Inferenzproblem Zustandsanfragen***
- Ausprägungen: Most probable explanation (MPE) / maximum a posteriori Anfragen (MAP)
- Semantik: Ausmaximieren statt aussummieren; max-out Operator in VE
- Auswirkungen auf die Komplexität