Intelligent Agents: Web-mining Agents

Probabilistic Graphical Models

Lifted Inference

Tanya Braun
Probabilistic Graphical Models (PGMs)

1. Recap: Propositional modelling
   - Factor model, Bayesian network, Markov network
   - Semantics, inference tasks + algorithms + complexity

2. Probabilistic relational models (PRMs)
   - Parameterised models, Markov logic networks
   - Semantics, inference tasks

3. Lifted inference
   - LVE, LJT, FOKC
   - Theoretical analysis

4. Lifted learning
   - Recap: propositional learning
   - From ground to lifted models
   - Direct lifted learning

5. Approximate Inference: Sampling
   - Importance sampling
   - MCMC methods

6. Sequential models & inference
   - Dynamic PRMs
   - Semantics, inference tasks + algorithms + complexity, learning

7. Decision making
   - (Dynamic) Decision PRMs
   - Semantics, inference tasks + algorithms, learning

8. Continuous Space
   - Gaussian distributions and Bayesian networks
   - Probabilistic soft logic
Problem: Many Queries

• Set of queries
  • $P(Travel(eve))$
  • $P(Sick(bob))$
  • $P(Treat(eve, m_1))$
  • $P(Epid)$
  • $P(Nat(flood))$
  • $P(Man(virus))$
  • Combinations of variables

• Under evidence
  • $Sick(X') = true$
  • $X' \in \{alice, eve\}$

• LVE restarts with initial model for each query
Outline: 3. Lifted Inference

A. Lifted variable elimination (LVE)
   • Operators
   • Algorithm
   • Complexity (including first-order dtrees), completeness, tractability
   • Variants

B. Lifted junction tree algorithm (LJT)
   • First-order junction trees (FO jtrees)
   • Algorithm
   • Complexity, completeness
   • Variants

C. First-order knowledge compilation (FOKC)
   • Normal form, circuits
   • Algorithm
   • Complexity, completeness

D. Most probable assignment queries
   • Distribution vs. assignment queries
   • Most probable explanation (MPE), Maximum-a-posteriori (MAP) assignments
   • Changes in LVE, LJT, FOKC
   • Complexity, completeness
Clustering of Models

• Idea: Find subsets (clusters) of PRVs that are “enough” for certain queries
  • E.g.,
    • For queries about instances of $Nat(D)$, $Man(W)$, $Epid$
      • $Nat(D)$, $Man(W)$, $Epid$ are enough
    • For queries about instances of $Travel(X)$, $Sick(X)$, $Epid$
      • $Travel(X)$, $Sick(X)$, $Epid$ are enough
    • For queries about instances of $Treat(X, M)$, $Sick(X)$, $Epid$
      • $Treat(X, M)$, $Sick(X)$, $Epid$ are enough
Clustering of Models

• But: If only parfactors used that contain the PRVs of a cluster, information stored in all other parfactors ignored
  • E.g.,
    • $Nat(D), Man(W), Epid: g_1 \rightarrow misses g_2, g_3$
    • $Travel(X), Sick(X), Epid: g_2 \rightarrow misses g_1, g_3$
    • $Treat(X, M), Sick(X), Epid: g_3 \rightarrow misses g_1, g_2$

• Only correct if clusters are independent from each other
  • How can we achieve independence?
Clustering of Models

• Remember: Global Markov Property
  • Any two subsets of variables are conditionally independent given a separating subset
  • E.g.,
  • \( Nat(D), Man(W), Epid: g_1 \)
    \( \rightarrow \) independent of the rest given \( Epid \)
  • \( Travel(X), Sick(X), Epid: g_2 \)
    \( \rightarrow \) independent of the rest given \( Epid, Sick(X) \)
  • \( Treat(X, M), Sick(X), Epid: g_3 \)
    \( \rightarrow \) independent of the rest given \( Epid, Sick(X) \)
Clustering of Models

- Put clusters and their separators into a graph structure where
  - Nodes are clusters with parfactors assigned containing the cluster PRVs \textit{(local model)}
  - Edges are labelled with the separator between neighbouring nodes
  - If two nodes contain the same PRV, every node on the path between the two nodes contain the PRV \textit{(running intersection property)}
Clustering of Models

• Next: Make clusters actually independent of each other
  • Each cluster $i$ asks its neighbours $j \in nbs(i)$ for information about the separator $S_{ij}$ between them
    • Other clusters have to collect all the information from the model that lies behind the separator on its part, eliminate the non-separator PRVs from that information using LVE, and send the result in a message $m_{ji}$, i.e., a set of parfactors, back
  • Having the information on the separators to all neighbours makes a cluster independent from its neighbours and therefore all other parts of the model
    • Ensures that each cluster of PRVs has all model information needed available for query answering on instances of its cluster PRVs
Clustering of Models

• Next: Make clusters actually independent of each other
  • E.g., $C_3: g_3 \rightarrow$ independent of the rest given $Epid, Sick(X)$
    • Asks neighbour $C_2$ for information on $Epid, Sick(X)$
      • $C_2$ asks neighbour $C_1$ for information on $Epid$
        • $C_1$ sends information on $Epid$ in a message $m_{12}$
          • Eliminates $Nat(D), Man(M)$ from $g_1$ for $m_{12}$
        • $C_2$ sends information on $Epid, Sick(X)$ to $C_3$ in a message $m_{23}$
          • Eliminates $Travel(X)$ from $g_2$ and $m_{12}$ for $m_{23}$
    • With $m_{23}$, $C_3$ is independent from its neighbour $C_2$ and therefore also from $C_1$
      • As $C_2$ is independent given $m_{12}$ from $C_1$

The same has to be done for $C_2$ and $C_1$
Clustering of Models

- With each cluster $i$ independent of the rest, each $i$ can answer queries about instances of its PRVs based on its local model and the messages received
  - Query terms: grounded instances or parameterised versions of its PRVs
  - Conjunctive queries if terms only concern the cluster PRVs
  - E.g., $C_3: g_3 \rightarrow \text{independent of the rest given } Epid, Sick(X)$
    - Based on $g_3$ and $m_{23}$, $C_3$ can answer queries about $Epid, Sick(X), Treat(X, M)$ such as $P(Sick(X)), P(Treat(eve, m_2)), P(Epid, Sick(alice))$
    - Cannot answer any queries about $Nat(D), Man(W), Travel(X)$ but $C_1$ and $C_2$ can
Clustering of Models

• Problem left: If each cluster asks for information on separators, some messages are sent multiple times
  • E.g.,
    • $C_3$ asks $C_2$, which asks $C_1$
      • Messages calculated and sent: $m_{12}, m_{23}$
    • $C_2$ asks $C_1$ and $C_3$
      • Messages calculated and sent: $m_{12}, m_{32}$
    • $C_1$ asks $C_2$, which asks $C_3$
      • Messages calculated and sent: $m_{32}, m_{21}$

Organise in way that messages are calculated only once
Clustering of Models

• Use dynamic programming to organise the order of asking or rather sending information in messages:
  → If a node $i$ has received all information from neighbours but one, $j$, node $i$ sends a message with its information on the separator $S_{ij}$ to $j$
  → If a node $i$ has received all messages, then it sends messages to all neighbours $j$ that have not received a message yet

• When computing the message, $i$ takes into consideration its local model as well as the messages received from all other neighbours but the receiving neighbour $j$
Clustering of Models

• Observations:

→ If a node $i$ has received all information from neighbours but one, $j$, node $i$ sends a message with its information on the separator $S_{ij}$ to $j$
  • Trivially true at leaf nodes (periphery), can start sending immediately to its only neighbour (in parallel!)
    • From periphery inbound, new nodes trigger this first condition
→ If a node $i$ has received all messages, then it sends messages to all neighbours $j$ that have not received a message yet
  • As messages are sent further inwards, a first node at the centre will have received all messages and will start sending messages outbound, leading to new nodes triggering this second condition

These two passes from periphery inbound and back suffice to distribute all information and make the clusters independent from each other*

Foundations of Clustering

• History in propositional probabilistic inference:
  • Based on probability propagation introduced by Pearl (1988)

• If a BN is a polytree, i.e., the underlying undirected graph has no trivial cycles, then
  • Treat each node in a BN as a cluster with the random variables (randvars) of the accompanying CPT as the cluster randvars
  • Send messages along the edges (to parents and children), eliminating randvars not occurring in the parent or child nodes
Foundations of Clustering

• History in propositional probabilistic inference:
  • If no polytree, the cycles mess up the message passing along the edges (information arrives multiple times)
    • Send messages nonetheless (exact if polytree, approximate otherwise): called belief propagation as an algorithm for approximate inference
  • Exact inference required → put the cycles into one cluster
  • Graph formed called a junction tree (jtree)
    • A first-order version of a jtree was induced on the previous slides
    • Also known as clique tree (since the cycles often form cliques in the model graph) or join tree
    • Propositional version introduced by Lauritzen and Spiegelhalter (1988)
    • Shenoy and Shafer (1989) introduce three axioms of local computations to show correctness of doing computations locally

First-order Jtree (FO Jtree)

• As seen on the earlier slides
  • Acyclic graph
  • Nodes contain PRVs, which form clusters
  • Edges are based on the separators between the clusters
  • Nodes have parfactors assigned

• Next slides:
  • Formal definition
  • Construction
    • Get an *acyclic* structure with valid *separators* and each parfactor of a model assigned to a *local model*
Parameterised Clusters

• Node of an FO jtree:
  Set of PRVs called parameterised cluster (parcluster)

• Let \( X \) be a set of logvars, \( A \) a set of PRVs with \( \text{lv}(A) \subseteq X \), and \((X, C_X)\) a constraint on \( X \) with \( X \) being a sequence of the logvars of \( X \)

• Then, a parcluster \( C \) is given by
  \[
  \forall x \in C_X : A_{|(x, C_X)}
  \]
  \( A_{|(x, C_X)} \) for short
  Again, \((X, C_X)\) can be omitted if \( \top \) constraint encoded
  Depicted as a round shape containing \( A \) or just \( A \)
  Again, constraint usually not depicted

• E.g., parcluster \( C_2 \)
  \[
  \forall x \in D(X) : \{\text{Epid, Sick}(x), \text{Travel}(x)\}_{|(x, D(X))}
  \]
  \[
  = \{\text{Epid, Sick}(X), \text{Travel}(X)\}_{|(x, D(X))}
  \]
  \[
  = \{\text{Epid, Sick}(X), \text{Travel}(X)\}
  \]
FO Jtree

• An FO jtree $J$ for a model $G$ is a cycle-free graph $(V, E)$
  • Set of nodes $V \subseteq 2^{rv(G)}$
    • I.e., nodes are sets of PRVs (parclusters)
    • $2^{rv(G)}$ denotes the power set of $rv(G)$
  • Set of edges $E \subseteq \{\{i, j\} | i, j \in V, i \neq j\}$,
    • Has to be cycle free, which includes no self-loops
  • E.g., as depicted on the left
    • But at this point in the definition, could be any subsets of PRVs

\[
\begin{array}{c}
\text{Epid Nat}(D) \\
\text{Man}(W) \quad C_1
\end{array}
\quad
\begin{array}{c}
\text{Epid Sick}(X) \\
\text{Travel}(X) \quad C_2
\end{array}
\quad
\begin{array}{c}
\text{Epid Sick}(X) \\
\text{Treat}(X, M) \quad C_3
\end{array}
\quad
\begin{array}{c}
\text{Nat}(D) \\
\text{Man}(M)
\end{array}
\quad
\begin{array}{c}
\text{Epid} \\
\text{Treat}(X, P)
\end{array}
\quad
\begin{array}{c}
\text{Sick}(X) \\
\text{Travel}(X)
\end{array}
\quad
\begin{array}{c}
g_1 \\
g_2 \\
g_3
\end{array}
\]
FO Jtree

• An FO jtree $J$ for a model $G$ is a cycle-free graph $(V, E)$
  • Has to satisfy three properties:
    1. $\forall C \in V : C \subseteq \text{rv}(G)$
    2. $\forall g \in G : \exists C \in V : \text{rv}(g) \subseteq C$
    3. If $\exists A \in \text{rv}(G) : A \in C_i \land A \in C_j$ with $C_i, C_j \in V$, then $\forall C_k \in V$ on the path between $C_i, C_j : A \in C_k$ (running intersection property)

• E.g., as depicted on the left
  • Only the following and one with $C_3$ at the centre are valid
An FO jtree $J$ for a model $G$ is a cycle-free graph $(V, E)$
- Is minimal if by removing a PRV from a parcluster, the FO jtree ceases to be an FO jtree, i.e., no longer fulfils at least one property
- E.g., depicted on the left
  - Cannot remove any PRV from any parcluster
    - Otherwise, a parfactor would no longer have its arguments in one parcluster
An FO jtree $J$ for a model $G$ is a cycle-free graph $(V, E)$

- Set $S_{ij}$ called separator of edge $\{i, j\} \in E$, defined by
  $$S_{ij} = C_i \cap C_j$$

- Term $\text{nbs}(i)$ refers to the neighbours of $C_i$, defined by
  $$\text{nbs}(i) = \{j \mid \{i, j\} \in E\}$$

- Each $C_i$ has a local model $G_i$ and $\forall g \in G_i : rv(g) \subseteq C_i$
  - Local models $G_i$ partition $G$, i.e., $G = \bigcup_{i \in V} G_i$
Construction

• Where do we get the FO jtree from s.t. the jtree
  • is acyclic
  • fulfils the three FO jtree properties
  • has the model parfactors automatically assigned to fitting parclusters?

→ Clusters of an FO dtree
  + undirected dtree edges
  + minimisation
  = FO jtree
Clusters → Parclusters

• Given an FO dtree $T$ for a model $G$ with clusters for each node

• Given a cluster $\{A_1, \ldots, A_n\}$ of a DPG node $(X, x, C)$
  • Resulting parcluster $C_j = \{A_1, \ldots, A_n\}_C$
  • Local model $G_j = \emptyset$

• Given a cluster $\{A_1, \ldots, A_n\}$ of a VE node
  • Resulting parcluster $C_j = \{A_1, \ldots, A_n\}_T$
  • Local model $G_j = \emptyset$

• Given a cluster $\{A_1, \ldots, A_n\}$ from a leaf node with parfactor $g_i$
  • Resulting parcluster $C_j = \{A_1, \ldots, A_n\}_T$
  • Local model $G_j = \{g_i\}$

Let’s carry the constraint around for a bit to make it explicit
FO Dtree → FO Jtree

• Forming an FO jtree $J$ from an FO dtree $T$ of a model $G$

• Nodes of $J$
  • Parclusters resulting from clusters of $T$ as shown on previous slide
    • Each parcluster has a source node in $T$

• Edges of $J$
  • Add an edge between two parclusters whenever there is an edge between the source nodes of the two parclusters in $T$
FO Dtree $\rightarrow$ FO Jtree

- Result after transformation
  - Fulfils the three jtree properties
  - But is not minimal
FO Dtreet $\rightarrow$ FO Jtree

- Result after transformation fulfils the three jtree properties
- Hold by construction
  1. Parclusters can only contain model PRVs
  2. Each parfactor occurs at a dtree leaf, which is turned into a parcluster
  3. Based on how cutset/context are calculated*
  - E.g., $Sick(X)$


1. $\forall C \in V: C \subseteq rv(G)$
2. $\forall g \in G: \exists C \in V: rv(g) \subseteq C$
3. If $\exists A \in rv(G): A \in C_i \land A \in C_j$ with $C_i, C_j \in V$, then $\forall C_k \in V$ on the path between $C_i, C_j : A \in C_k$

![Diagram of FO Jtree transformation](image)
FO Dtreet $\rightarrow$ FO Jtree

- Result after transformation **not minimal**
  - Can remove complete parclusters and still have an FO jtree
    - Even if we keep parclusters that carry constraint information that we would otherwise lose
  - E.g.,
    - Parclusters **marked**

- **Observation**
  - Parclusters are subsets of other parclusters
    - Use for minimisation
Minimisation

• Merge parclusters $C_i$ and $C_j$ with local models $G_i$ and $G_j$ iff

\[ \text{gr}(C_i) \subseteq \text{gr}(C_j) \lor \text{gr}(C_j) \subseteq \text{gr}(C_i) \]

• Assuming T constraints and same logvar names if the same domain is referenced (from normal form of FO dtree), then the following suffices:

\[ C_i \subseteq C_j \lor C_j \subseteq C_i \]

• Checking on a PRV and logvar level instead of a grounded level
Minimisation

• Pre-processing necessary:
  • Parclusters may contain a logvar $X$ or a representative $x$

• For each source DPG node $T_X$
  • Apply the inverse substitution $\theta^{-1}$ to the one applied during FO dtree construction to all parclusters that come from descendants of $T_X$:
    \[
    \theta^{-1} = \{X \rightarrow x\}^{-1}
    \]
    \[
    = \{x \rightarrow X\}
    \]
Minimisation

• Merging parclusters $C_i$ and $C_j$ into parcluster $C_k$
  • $C_k = C_i \cup C_j$
  • $G_k = G_i \cup G_j$

• Changes in FO jtree $(V, E)$
  • $V = V \setminus \{C_i, C_j\} \cup C_k$
  • $E = E \setminus \{{i, l} \mid l \in nbs(i)\} \setminus \{{j, l} \mid l \in nbs(j)\}$
    $\cup \{{k, l} \mid l \in nbs(i) \lor l \in nbs(j), l \neq i, l \neq j\}$
Minimisation

- **Possible merging strategy**
  - Start at the leaves and merge **inbound**
  - Until no further merging is possible
    - No parcluster is a subset of another

- **After merging, the resulting FO jtree is minimal**
  - E.g.,
    - Start at leaves with
      - local model \(\{g_1\}\)
      - local model \(\{g_2\}\)
      - local model \(\{g_3\}\)

\[
\begin{tikzpicture}[level distance=1.5cm,
  level 1/.style={sibling distance=3.5cm},
  level 2/.style={sibling distance=2.5cm},
  level 3/.style={sibling distance=1.5cm},
  every node/.style={draw, rounded corners}]

  \node (root) {Epid}
  \node (Tw) [below of=root] {Epid, Nat(D)\vert_T} child {node {Epid, Nat(D), Man(W)}}
  \node (Td) [below of=Tw] {Epid, Man(W), Nat(D)\vert_T}
  \node (Td) [right of=Tw] {Epid, Man(W), Nat(D)\vert_T}
  \node (g1) [below of=Td] {Epid, Nat(D), Man(W)}
  \node (g2) [right of=Td] {Epid, Sick(X)\vert_T}
  \node (g3) [right of=Td] {Epid, Sick(X)\vert_T}
  \node (gx) [below of=g2] {Epid, Sick(X), Treat(X, M)}
  \node (g1) [below of=gx] {Epid, Sick(X), Treat(X, M)}
  \node (g2) [below of=gx] {Epid, Sick(X), Travel(X)}

\end{tikzpicture}
\]
Minimisation: Example Continued

- Consider leaf parcluster with local model \( \{g_1\} \)
  - Let us call it \( C_1 \)
  - Merge inbound
    - \( C_1 \) and \( T_d \) parcluster identical → merge (call result \( C_1 \) again)
Minimisation: Example Continued

• Consider leaf parcluster with local model \{g_1\}
  • Let us call it \(C_1\)
  • Merge inbound
    • \(C_1\) and \(T_d\) parcluster identical \(\rightarrow\) merge (call result \(C_1\) again)
    • \(C_1\) and \(T_D\) parcluster identical \(\rightarrow\) merge
Minimisation: Example Continued

• Consider leaf parcluster with local model \( \{g_1\} \)
  • Let us call it \( C_1 \)
  • Merge inbound
    • \( C_1 \) and \( T_d \) parcluster identical \( \rightarrow \) merge (call result \( C_1 \) again)
    • \( C_1 \) and \( T_D \) parcluster identical \( \rightarrow \) merge
    • \( C_1 \) and \( T_W \) parcluster identical \( \rightarrow \) merge
Minimisation: Example Continued

• Consider leaf parcluster with local model \{g_1\}
  • Let us call it \(C_1\)
  • Merge inbound
    • \(C_1\) and \(T_d\) parcluster identical \(\rightarrow\) merge (call result \(C_1\) again)
    • \(C_1\) and \(T_D\) parcluster identical \(\rightarrow\) merge
    • \(C_1\) and \(T_W\) parcluster identical \(\rightarrow\) merge
    • \(T_W\) parcluster subset of \(C_1\) \(\rightarrow\) merge
Minimisation: Example Continued

• Consider leaf parcluster with local model \( \{g_1\} \)
  • Let us call it \( C_1 \)
  • Merge **inbound**
    • \( C_1 \) and \( T_d \) parcluster identical \( \rightarrow \) merge (call result \( C_1 \) again)
    • \( C_1 \) and \( T_D \) parcluster identical \( \rightarrow \) merge
    • \( C_1 \) and \( T_W \) parcluster identical \( \rightarrow \) merge
    • \( T_W \) parcluster subset of \( C_1 \) \( \rightarrow \) merge
    • Root parcluster subset of \( C_1 \) \( \rightarrow \) merge
Minimisation: Example Continued

• Consider leaf parcluster with local model \( \{g_1\} \)
  • Let us call it \( C_1 \)
  • Merge inbound
    • \( C_1 \) and \( T_d \) parcluster identical → merge (call result \( C_1 \) again)
    • \( C_1 \) and \( T_D \) parcluster identical → merge
    • \( C_1 \) and \( T_W \) parcluster identical → merge
    • \( T_W \) parcluster subset of \( C_1 \) → merge
    • Root parcluster subset of \( C_1 \) → merge
Minimisation: Example Continued

• Consider leaf parcluster with local model \( \{g_1\} \)
  • Let us call it \( C_1 \)
  • At this point, we have reached the former root and cannot merge further inbound
    • Also: the \( T_X \) parcluster contains logvar \( X \), which is not a subset or superset of the logvars of \( C_1 (D, W) \)
  • Merging stops
Minimisation: Example Continued

- Consider leaf parcluster with local model \{g_2\}
  - Let us call it \(C_2\)
  - Merge inbound
    - \(C_2\) and neighbouring parcluster identical \(\rightarrow\) merge
Minimisation: Example Continued

- Consider leaf parcluster with local model \( \{ g_2 \} \)
  - Let us call it \( C_2 \)
  - Merge **inbound**
    - \( C_2 \) and neighbouring parcluster identical $\rightarrow$ merge
    - \( T_x \) parcluster is a subset of \( C_2 \) $\rightarrow$ merge
Minimisation: Example Continued

- Consider leaf parcluster with local model \{g_2\}
  - Let us call it \(C_2\)
  - Merge inbound
    - \(C_2\) and neighbouring parcluster identical → merge
    - \(T_X\) parcluster is a subset of \(C_2\) → merge
    - \(T_X\) parcluster is a subset of \(C_2\) → merge
Minimisation: Example Continued

- Consider leaf parcluster with local model \( \{g_2\} \)
  - Let us call it \( C_2 \)
  - Merge inbound
    - \( C_2 \) and neighbouring parcluster identical \( \rightarrow \) merge
    - \( T_X \) parcluster is a subset of \( C_2 \) \( \rightarrow \) merge
    - \( T_X \) parcluster is a subset of \( C_2 \) \( \rightarrow \) merge
  - Merging cannot move further inbound
    - \( C_1 \) is neither a subset nor a superset of \( C_2 \)
    - Merging stops
Minimisation: Example Continued

- Consider leaf parcluster with local model \( \{g_3\} \)
  - Let us call it \( C_3 \)
  - Merge inbound
    - \( C_3 \) and \( T_m \) parcluster identical → merge

![Diagram showing the minimisation process](attachment:diagram.png)
Minimisation: Example Continued

- Consider leaf parcluster with local model \( \{g_3\} \)
  - Let us call it \( C_3 \)
  - Merge **inbound**
    - \( C_3 \) and \( T_m \) parcluster identical
      - \( \rightarrow \) merge
    - \( T_m \) parcluster is a subset of \( C_3 \)
      - \( \rightarrow \) merge
Minimisation: Example Continued

• Consider leaf parcluster with local model \( \{g_3\} \)
  • Let us call it \( C_3 \)
  • Merge inbound
    • \( C_3 \) and \( T_m \) parcluster identical
      \( \rightarrow \) merge
    • \( T_m \) parcluster is a subset of \( C_3 \)
      \( \rightarrow \) merge
  • Merging cannot move further inbound
    • \( C_3 \) is neither a subset nor a superset of \( C_2 \)
    • Merging stops
Minimisation: Example Continued

- Resulting FO jtree $J$ from FO dtree $T$ given model $G$
  - If we had started merging from leaf with $g_3$ inbound before merging from leaf with $g_2$, $C_2$ and $C_3$ would be switched
FO Jtree Construction

• Given a model $G$, the following steps are necessary
  1. Bring $G$ into the required normal form for FO dtree construction
  2. Construct an FO dtree $T$ for $G$
  3. Translate $T$ into an FO jtree $J$
  4. Apply inverse substitutions to parclusters of descendants of DPG nodes in $J$
  5. Minimise $J$

• Next?
• FO jtrees for query answering
  • Messages need to be passed to ensure independence
  • What about evidence?
Message Passing in FO Jtrees

- Ensure independence between parclusters
- Send messages based on two conditions
  - If a node $i$ has received all messages from neighbours but one, $j$, node $i$ calculates and sends a message to $j$
  - If a node $i$ has received all messages, then it calculates and sends messages to all neighbours $j$ that have not received a message yet
Message Passing in FO Jtrees

• Message $m_{ij}$ from sender $C_i$ to receiver $C_j$
  • Set of parfactors $\{g_l\}_{l=1}^n$ with $rv(g_l) \subseteq S_{ij}$
  • To calculate
    • Collect necessary information from local model and received messages:
      $$G_{ij} = G_i \cup \bigcup_{k \in nbs(i), k \neq j} m_{ki}$$
    • Ignore the message that came from $C_j$ (if it already exists)
    • Call slightly modified LVE with $G_{ij}$ as input model, $S_{ij}$ as query, and no evidence: $LVE^*(G_{ij}, S_{ij}, \emptyset)$
    • Specification of $LVE^*$: next slide
LVE\textsuperscript{*}(G, Q, \{g_e\}_{e=1}^m)

G ← Shatter G on Q, \{g_e\}_{e=1}^m, and on itself
G ← Absorb \{g_e\}_{e=1}^m in G

while G contains non-query terms do
    if a PRV A fulfils the preconditions of sum-out then
        G ← Apply sum-out to A in G
    else
        G ← Apply an enabling operator (multiply, count–convert, expand, count–normalise, split, ground) on some parfactors in G

    g ← Multiply all parfactors in G into one parfactor
    g ← Normalise the potentials in g

return g

return G
Message Passing in FO Jtrees

• E.g.,
  • Message $m_{12}$ from $C_1$ to $C_2$
    • Collect $G_{12} = \{g_1\} \cup \emptyset$
      • No further neighbours except $C_2$
    • Call $\text{LVE}^* (\{g_1\}, \{\text{Epid}\}, \emptyset)$
      • $\text{LVE}^*$ eliminates $\text{Nat}(D)$, $\text{Man}(W)$ from $\{g_1\}$
        • Count-converting $\text{Nat}(D)$ into $\#_D [\text{Nat}(D)]$
        • Summing out $\text{Man}(W)$ and then $\#_D [\text{Nat}(D)]$
      • Returning $\{g'_1\}$
    • Send $\{g'_1\}$ as $m_{12}$ to $C_2$
Message Passing in FO Jtrees

• E.g.,
  • Message $m_{32}$ from $C_3$ to $C_2$
    • Collect $G_{32} = \{g_3\} \cup \emptyset$
      • No further neighbours except $C_2$
    • Call LVE* ($\{g_3\}, \{\text{Epid, Sick}(X)\}, \emptyset$)
      • LVE* eliminates $\text{Treat}(X, M)$ from $\{g_3\}$
        • Summing out $\text{Treat}(X, M)$
        • Returning $\{g_3'\}$
    • Send $\{g_3'\}$ as $m_{32}$ to $C_2$
Message Passing in FO Jtrees

- E.g.,
  - Message $m_{21}$ from $C_2$ to $C_1$
    - Collect $G_{21} = \{g_2\} \cup m_{32}$
      - Further neighbour: $C_3$, sent message $m_{32} = \{g'_3\}$
    - Call $LVE^*(\{g_2, g'_3\}, \{Epid\}, \emptyset)$
      - $LVE^*$ eliminates $Travel(X), Sick(X)$ from $\{g_2, g'_3\}$
        - Summing out $Travel(X)$ from $g_2$, yielding $g'_2$
        - Summing out $Sick(X)$ from product of $g'_2$ and $g'_3$, yielding $g'_{23}$
      - Returning $\{g'_{23}\}$
    - Send $\{g'_{23}\}$ as $m_{21}$ to $C_1$
Message Passing in FO Jtrees

- E.g.,
  - Message \( m_{23} \) from \( C_2 \) to \( C_3 \)
    - Collect \( G_{23} = \{g_2\} \cup m_{12} \)
      - Further neighbour: \( C_1 \), sent message \( m_{12} = \{g'_1\} \)
    - Call \( \text{LVE}^* (\{g_2, g'_1\}, \{\text{Epid}, \text{Sick}(X)\}, \emptyset) \)
      - \( \text{LVE}^* \) eliminates \( \text{Travel}(X) \) from \( \{g_2, g'_1\} \)
        - Summing out \( \text{Travel}(X) \) from \( g_2 \), yielding \( g'_2 \)
        - Returning \( \{g'_2, g'_1\} \)
    - Send \( \{g'_2, g'_1\} \) as \( m_{23} \) to \( C_3 \)
Message Passing: Overview

• Given an FO jtree $J$, send messages if one of the two conditions is true
  → If a node $i$ has received all messages from neighbours but one, $j$, node $i$ calculates and sends a message to $j$
  → If a node $i$ has received all messages, then it calculates and sends messages to all neighbours $j$ that have not received a message yet

• To calculate a message:
  • Collect necessary information from local model and received messages:
    $$G_{ij} = G_i \cup \bigcup_{k \in \text{nebs}(i), k \neq j} m_{ki}$$
  • Call $\text{LVE}^* \left( G_{ij}, S_{ij}, \emptyset \right)$
Query Answering in FO Jtrees

- After message passing, the parclusters are independent from each other given the messages
  - Prepared for query answering

For each query with query term $Q$

- Find parcluster $C_i$ s.t. $Q \in C_i$
- Collect information from local model and messages, i.e.,
  $$G_Q = G_i \cup \bigcup_{j \in nbs(i)} m_{ji}$$
- Call $LVE(G_Q, Q, \emptyset)$ and return or store result of the call
Query Answering in FO Jtrees

- E.g., $P(Epid)$
  - All parclusters contain $Epid$, choose one at random, e.g., $c_2$
  - Collect $G_{Epid} = \{g_2\} \cup m_{12} \cup m_{32} = \{g_2, g'_1, g'_3\}$
  - Call LVE($\{g_2, g'_1, g'_3\}, Epid, \emptyset$), yielding a parfactor $g$ containing the probability distribution over $Epid$

- What about evidence?
Evidence in FO Jtrees

- Evidence applies to PRVs in some parclusters
  - Changes the distributions in local models
  - Information sent in messages might change
    - Even if summed out and therefore hidden from the other parclusters

- Therefore, handle evidence before sending messages

- Given a set of evidence parfactors \( \{ \phi_e(R(X)) \mid C_e \} \) \( e=1 \)
  - For each \( \phi_e(R(X)) \)
    - For each parcluster \( C_i \) where \( R(X) \in C_i \)
      - Shatter \( G_i \) on \( R(X) \mid C_e \)
      - Absorb \( \phi_e(R(X)) \mid C_e \) in \( G_i \)

- Only, then send messages
Evidence in FO Jtrees

• E.g., given $\text{Sick}(\text{eve}) = \text{true}$ as evidence in $g_e$
  • In $C_2$
    • Shatter $g_2 = \{g_2\}$ on $\text{Sick}(\text{eve})$, yielding $\{g_e^2, g_2'\}$
    • Absorb $g_e$ in $g_e^2$, yielding $g_e^2'$
    • Result: $G_2 = \{g_e^2', g_2'\}$
  • In $C_3$
    • Shatter $g_3 = \{g_3\}$ on $\text{Sick}(\text{eve})$, yielding $\{g_e^3, g_3'\}$
    • Absorb $g_e$ in $g_e^3$, yielding $g_e^3'$
    • Result: $G_3 = \{g_e^3', g_3'\}$
• Then, send messages based on the local models that have absorbed the evidence
Evidence in FO Jtrees

• E.g., given $\text{Sick}(\text{eve}) = true$ as evidence in $g_e$
  • Message $m_{12}$ does not change compared to previous example
  • Message $m_{32}$ calculated based on $\{g_3^{e'}, g_3'\}$
    • Call $\text{LVE}^* (\{g_3^{e'}, g_3'\}, \{\text{Epid}, \text{Sick}(X)\}, \emptyset)$, yielding $\{g_3^{e''}, g_3''\}$
  • Message $m_{23}$ calculated based on $\{g_2^{e'}, g_2'\} \cup m_{12}$
    • Call $\text{LVE}^* (\{g_2^{e'}, g_2', g_1'\}, \{\text{Epid}, \text{Sick}(X)\}, \emptyset)$, yielding $\{g_2^{e''}, g_2'', g_1'\}$
  • Message $m_{21}$ calculated based on $\{g_2^{e'}, g_2'\} \cup m_{32}$
    • Call $\text{LVE}^* (\{g_2^{e'}, g_2', g_3^{e''}, g_3''\}, \{\text{Epid}\}, \emptyset)$, yielding $\{g_2^{e''}, g_2'', g_3^{e''}, g_3''\}$
Evidence and Queries in FO Jtrees

• After evidence handling
  • All queries are answered in an FO jtree with handled evidence \{g_e\}_{e=1}^m yield results conditional on \{g_e\}_{e=1}^m
  • So, given evidence \{g_e\}_{e=1}^m and query terms \{Q_i\}_{i=1}^n for a model \(G\)
    • The posed queries are \(P(Q_i | \{g_e\}_{e=1}^m), 1 \leq i \leq n, \text{w.r.t. } P_G\)

• FO jtree constructed without specific evidence
  • Reuse for different evidence sets
    • As long as model stays the same
  • Reset the local models before entering new evidence
LJT: Algorithm

\[ \text{LJT}(G, \{Q_i\}_{i=1}^{n}, \{g_e\}_{e=1}^{m}) \]

Construct an FO jtree \( J \) for \( G \)
Enter evidence \( \{g_e\}_{e=1}^{m} \) into \( J \)
Pass message in \( J \)
Answer queries with query terms \( \{Q_i\}_{i=1}^{n} \) in \( J \)

• Look for blue boxes on the previous slides to find the descriptions of each step

• \textit{Constant} overhead for FO jtree construction

• Payoff if given multiple queries
Comparison to Ground Inference

- Propositional Junction Tree Algorithm (JT)
  - Same algorithm, only with propositional model
  - E.g., gr(G)
Junction Tree: Messages

- From *periphery* to *centre* and back
Junction Tree: Symmetry → Inefficiency

- Identical messages incoming
- Information already present
- Calculating identical messages + sending information partially present

\[ m_{eve}: \text{Eliminate } Travel.eve, Sick.eve \]
from \( f_2^2, m_{p1}, m_{p2} \)

\[ m_{m1}: \text{Eliminate } Treat.eve.m_1 \]
from \( f_3^1 \)

\[ m_{m1, back} \]

\[ f_3^2 \]

\[ f_3^5 \]

\[ m_{m2, back} \]

\[ m_{m2}: \text{Eliminate } Treat.eve.m_2 \]
from \( f_3^2 \)
Compact Encoding of Jtrees

Epid Man. war Dis. fire Dis. flood

Epid Man. virus Dis. fire Dis. flood

Epid Sick. alice Travel. alice

Epid Sick. eve Travel. eve

Epid Sick. bob Travel. bob

Epid Sick. alice Treat. alice.m₁

Epid Sick. alice Treat. alice.m₂

Epid Sick. eve Treat. eve.m₁

Epid Sick. eve Treat. eve.m₂

Epid Sick. bob Treat. bob.m₁

Epid Sick. bob Treat. bob.m₂

Epid Nat(D) Man(W) C₁

Epid Sick(X) Travel(X) C₂

Epid Sick(X) Treat(X, M) C₃
Message Calculation Strategies

• Message calculation strategy seen so far
  • Eliminate all non-separator PRVs from all messages but the one that came from receiver and the local model
  • Called *Shafer-Shenoy* architecture after the two researchers who first presented the scheme

• Another strategy for JT exists, called *Hugin* architecture
  • Multiply the factors of local model $G_i$ into one factor $g_i$
  • Multiply each incoming message $m_{ji}$ into $g_i$
    • Store $m_{ji}$ as well
    • Each message consists of only one factor (no longer a set)
  • When sending message $m_{ij}$
    • Eliminate all non-separator randvars from $\frac{g_i}{m_{ji}}$
      • I.e., divide $g_i$ by $m_{ji}$ first
      • If $m_{ji}$ does not exist, then divide by a symbolic 1 (or no division)
Message Calculation Strategies

• *Hugin* architecture continued

  • May enlarge the factors at each node to the worst-case size of each node
    • E.g., $G_i = \{\phi(R, S), \phi(S, T)\} \rightarrow g_i = \phi(R, S, T)$
  • May lead to more involved multiplications
    • E.g., multiplying message $m_{ij} = \phi(R)$ into $g_i = \phi(R, S, T)$
      more involved then multiplying $\phi(R)$ into an intermediate result $\phi'(R)$
  • Pays off if the nodes of the jtree have a high degree
    • Many duplicate multiplications during message calculation
      • E.g., $m_{3i}, m_{4i}, m_{5i}, m_{6i}$ and $g_i$ have to be multiplied for both $m_{1i}, m_{2i}$
    • Requires a division operator for factors

What about a Lifted Hugin?
Message Calculation Strategies

• Lifted Hugin?
  • Arguments pro and con also apply to lifted version
    • May enlarge the factors at each node to the worst-case size of each node
    • May lead to more involved multiplications
    • Pays off if the nodes of the jtree have a high degree
    • Requires a division operator for factors
  • Main obstacle: So far, no lifted division operator
    • We are working on it @Moritz
  • Also, CAUTION: In general, parfactors may be multiplied with different logvars such that previously unnecessary count conversions might become necessary
In terms of Lifting: Is it that simple?

• Algorithm-induced **groundings** due to message passing
  • For message calculation, non-separator PRVs are eliminated with separator PRVs as the query terms containing logvars
    • Non-separator PRVs have to fulfil sum—out preconditions
      1. \( \forall B \in rv(G \setminus \{g\}): \text{gr}(B|_{C}) \cap \text{gr}(A|_{(x,c_x)}) = \emptyset \)
      2. \( \forall X \in \{X \mid |\pi_X(C_X)| > 1\}: X \in lv(A) \)
      3. \( X^{\text{excl}} = lv(A) \setminus (X \setminus lv(A)) \) count-normalised w.r.t. \( X^{\text{com}} = lv(A) \cap X \) in \( C \), with \( X \) the set of \( X \)
  • Preconditions 1 + 3 fulfilled by construction
  • Precondition 2 may not be fulfilled \( \rightarrow \) can cause groundings
  • E.g., logvar \( E \) added to PRVs \( \text{Epid}, \text{Sick}(X), \text{Treat}(X, M) \)
    • When calculating \( m_{23} \), one has to eliminate \( \text{Travel}(X) \)
      • But: it does not contain both \( X \) and \( E \) and a count conversion does not apply as \( E \) occurs in two PRVs \( \rightarrow E \) gets **grounded**
Conditions on Groundings

• For a lifted calculation of message $m_{ij}$, it necessarily has to hold that
  • for each PRV $A \in (C_i \setminus S_{ij})$, i.e., $A$ has to be eliminated:
    • for each separator PRV $S \in S_{ij}$: $lv(S) \subseteq lv(A)$ (Cond. 1)
  • If Cond. 1 does not hold, i.e., $lv(S) \not\subseteq lv(A)$, one may induce Cond. 1 by count conversion
    • If $lv(S) \setminus lv(A)$ are countable in $G_{ij}$ (Cond. 2)
Conditions on Groundings

• Problem with induced Cond. 1 using count conversions on the logvars in $lv(S) \setminus lv(A)$:
  • Logvars that were previously not counted are now counted
  • All receiving parclusters need to be able to handle the counted versions, which needs to be checked
    • If a newly counted logvar arrives at a parcluster $C_k$, it has to be countable in $G_k$ as well (Cond. 3)
  • For further calculations, since they refer to the same set of randvars, they have to occur in the same form, i.e., at one point the logvar has to be counted in $G_k$ as well
Fusion

• Extra step at end of construction called fusion
  • Test each possible message $m_{ij}$ for each PRV $A$ to eliminate and each separator PRV $S$ based on the three conditions
    • If Cond. 1 holds: no groundings for $A$ and $S$; continue
    • Otherwise:
      • If Cond. 2 holds: check Cond. 3
        • If Cond. 3 holds: no groundings for $A$ and $S$; continue
        • Otherwise: groundings; mark $m_{ij}$; continue with next message
      • Otherwise: groundings; mark $m_{ij}$; continue with next message
  • For each message $m_{ij}$ marked:
    • Merge parclusters $C_i, C_j$ (as in minimisation)
  • E.g.,
    • Testing marks $m_{23}$ → merge $C_2, C_3$

\[\begin{array}{ccc}
\text{Epid}(E) \text{ Nat}(D) & \text{Man}(W) & \text{Epid}(E) \text{ Sick}(X, E) \\
C_1 & & C_2 \\
\text{Epid}(E) \text{ Sick}(X, E) & \text{Travel}(X) & \text{Treat}(X, M, E) \\
C_2' & & C_3 \\
\end{array}\]
LJT: Complexity

- Uses also the notion of lifted width $w_T = (w_g, w_\#)$
  - $w_g$ largest ground width
  - $w_\#$ largest counting width
- As FO jtree constructed from FO dtree, $w_T$ identical between LVE and LJT
  - Fusion may change $w_T$ in terms of the FO jtree
    - But in terms of the LVE calculations in the merged parcluster, $w_T$ is still the same with multiple nodes being combined into one
    - For simplicity, let us consider models that all fulfil Cond. 1 in fusion such that $w_T$ is identical for both LJT and LVE
LJT: Complexity

- LJT complexity based on complexity of LVE:
  \[ O(n_T \cdot \log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}) \]

- Complexity of individual steps
  - Construction: linear in number of nodes, no calculations; negligible compared to later steps
  - Evidence entering: \( O(n_J \cdot \log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}) \)
    - Absorbing evidence complexity: \( O\left(\log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}\right) \)
      - Visits \( \frac{1}{r} \cdot r^{wg} \cdot n^{r\#w\#} \) lines, possibly exponentiates the potentials
  - At each node \( \rightarrow n_J \cdot O\left(\log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}\right) \)
    - \( n_J \) number of nodes in FO jtree \( J \)
  - For each \( e \) evidence parfactors \( \rightarrow e \cdot O\left(n_J \cdot \log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}\right) \)
    - Assuming \( e \ll n_J \rightarrow O\left(n_J \cdot \log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}\right) \)
  - First two steps accumulated: \( O\left(n_J \cdot \log_2(n) \cdot r^{wg-1} \cdot n^{r\#w\#}\right) \)
LJT: Complexity

• Complexity of individual steps
  • First two steps accumulated: $O(n_J \cdot \log_2(n) \cdot r^{w_g-1} \cdot n^{r\#w\#})$
  • Message passing: $O(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#})$
    • Calculating one message = answering one query on a parcluster
      • Worst-case parfactor size at parcluster: $r^{w_g} \cdot n^{r\#w\#}$
      • Elimination of $|C_i \setminus S_{ij}|$ PRVs goes through each line, potentials may be exponentiated $\rightarrow O(\log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#})$
    • Two messages per edge, $n_J - 1$ edges in $J \rightarrow n_J \cdot O(\log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#})$
  • Query answering: $O(m \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#})$
    • Each query answered in one parcluster $\rightarrow O(\log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#})$
    • With $m$ query terms $\rightarrow m \cdot O(\log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#})$
  • All four steps accumulated:
    $$O \left( (n_J + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r\#w\#} \right)$$
Comparison to LVE

• For both holds: $w_g$ bounded from below by $\max_{\phi(A_1,\ldots,A_k) \in G} k$

• LVE complexity of one query
  = LJT complexity of message passing
  
  • $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^{w_g}})$ vs. $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^{w_g}})$
  
  • Actual number of calculations:
    • In LVE: $c_{LVE}$
    • For message pass: $2 \cdot c_{LVE}$

• For $m$ queries
  
  • LVE: $O(m \cdot n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^{w_g}})$
  
  • LJT: $O((n_j + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^{w_g}})$
  
  • Difference in $m \cdot n_T$ vs. $(n_j + m)$
    • LVE has complexity of $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^{w_g}})$ for one query
    • LJT only complexity of $O(\log_2(n) \cdot r^{w_g} \cdot n^{r^{w_g}})$ for one query

LJT only pays off if $m > 1$, most likely, starting with third query (two queries in LVE = one message pass)
LJT: Completeness

• Completeness results from LVE also hold for LJT
  • Proof using the FO jtree properties and the fusion conditions on a case basis regarding separators:
    • Separators containing only propositional randvars
      • Do not interfere with elimination order for sum–out
    • Separators additionally containing one-logvar PRVs
      • Do not interfere with elimination order for sum–out
        • Two-logvar PRVs within a parcluster eliminable
        • One logvar PRVs within a parcluster eliminable
          • Given all available counting versions
          • (Along the lines of completeness proof for \(M^{1 prv}\))
    • Separators additionally containing two-logvar PRVs
      • All two-logvar PRVs eliminable
        • If inequality constraint between them \(\rightarrow\) same parcluster, eliminable within one parcluster using group inversion
      • Because of fusion, PRVs with less logvars also part of separator or eliminable (may it be through extra count conversion)
        • Else parclusters would have been merged
LJT: Implementation

• Available at:
  • https://www.ifis.uni-luebeck.de/index.php?id=518&L=2
  • Based on the LVE implementation by Taghipour
    • Available at:
      • https://dtai.cs.kuleuven.be/software/gcfove
    • Includes an implementation of the propositional junction tree algorithm for comparison

• Input: BLOG files
  • Based on Bayesian Logic Programming Language
    • https://bayesianlogic.github.io
Runtimes: Increasing Domain Sizes

• Example model
  • All domain sizes $\in \{2, 4, \ldots, 20, 30, \ldots, 100, 200, \ldots, 1000\}$
  • No evidence
  • Queries:
    • $P(Travel(x_1))$
    • $P(Sick(x_1))$
    • $P(Treat(x_1, m_1))$
    • $P(Nat(d_1))$
    • $P(Man(w_1))$
    • $P(Epid)$
  • Test trade-off (overhead vs. faster inference)

• Test increasing
  • Ground width $w_g$
    • Default: 3
  • Counting width $w_\#$
    • Default: 1
  • Number of nodes $n_J$
    • Default: 3
  • Domain size $n$
    • Default: 1000
  • Based on $O(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_\#w_\#})$
Step-wise $O(n_j \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^w_{#w}})$

Runtimes in milliseconds
Default: $n = 1000, n_j = 3, w_g = 3, w_{#w} = 1$
Queries answering

FOKC: see next lecture
compile: all overhead time

Runtimes in milliseconds
Default: \( n = 1000, n_f = 3, w_g = 3, w_\# = 1 \)
Trade-off Evaluation: Criteria

- For multi-query algorithms
  - Overhead to set off (model is compiled into a helper structure) vs.
  - Shorter individual query answering time

- With
  - $t_{q,cpl}$ runtime for answering single query with an algorithm that uses compilation
  - $t_{q,uncpl}$ runtime for answering single query with an algorithm without compilation
  - $t_{c,cpl}$ runtime for compilation with an algorithm that uses compilation
  - What is the ratio between individual query answering times?
    \[ \alpha = \frac{t_{q,cpl}}{t_{q,uncpl}} \]
  - How many queries does it take to offset the overhead?
    \[ \beta = \frac{t_{c,cpl}}{t_{q,uncpl} - t_{q,cpl}} \]
  - Makes only sense if $\alpha > 1$
Trade-off

Default: \( n_f = 3, w_g = 3, w_\# = 1 \)
Beyond Standard LJT

• LJT is basically a framework for query answering that is independent of
  • Specific function encoding $\rightarrow$ calculating algorithm has to work with the encoding
    • Such as lists, tables, ADDs, etc.
  • Concrete query language $\rightarrow$ whatever the calculating algorithm can handle, LJT can (within parclusters)
    • E.g., with LVE, queries with
      • Uncertain evidence
      • Parameterised query terms
    • One exception: conjunctive queries!

⇒ Could use any other query answering algorithm for calculations as long as the query answering algorithm can handle message calculations
LJT for Conjunctive Queries

• Problem if query terms occur outside of one parcluster
  • E.g., with the FO jtree below
    • $P(Epid, Sick(eve))$ ✓
    • $P(Travel(eve), Treat(eve, m_1))$ X

• Solution:
  Temporarily merge parclusters such that the query terms occur in one parcluster
Parcluster Merging for Queries

- Find a subgraph \( J' \) of the FO jtree \( J \) for the query terms \( Q \) such that \( J' \) such that \( Q \subseteq rv(J') \)
  - For query answering, use a set \( G_Q \) that consists of local models in \( J' \) and messages from outside \( J' \)

- Why subgraph?
  - Allows for ignoring messages within \( J' \) and including messages from outside \( J' \) into parclusters of \( J' \)
    - No duplicate information used
    - Messages reused as much as possible
  - E.g., consider subgraph of \( C_i, C_k, C_j \) for query on \( R_1, T_1 \)
    - Take all outside messages and local models
    - Ignore inside messages \( m_{ki}, m_{ik}, m_{jk}, m_{kj} \)
Parcluster Merging for Queries

- Find a subgraph $J'$ of the FO jtree $J$ for the query terms $Q$ such that $J'$ such that $Q \subseteq rv(J')$

- Subgraph should be minimal for optimal performance, i.e., a minimisation problem to solve:

$$\arg \min_{J'} |rv(J')|$$

subject to $Q \subseteq rv(J')$

- Trade-off between finding a subgraph fast and finding a minimal one
  - It is not about the number of parclusters but the number of PRVs in the parclusters!
Parcluster Merging for Queries

• Find a subgraph $J'$ of the FO jtree $J$ for the query terms $Q$ such that $J'$ such that $Q \subseteq rv(J')$

• Subgraph should be minimal for optimal performance, i.e., a minimisation problem to solve:

$$\text{argmin}_{J'} |rv(J')|$$

s.t. $Q \subseteq rv(J')$

• Simple heuristics (without guarantees on optimality):
  • Start with one parcluster that fulfils $Q \cap C_i \neq \emptyset$ as $J'$
    • $Q' = Q \setminus C_i$ remains as not covered by $J'$
  • Perform a breadth-first search starting at $C_i$
    • Whenever a newly visited parcluster $C_j$ fulfils $Q' \cap C_j \neq \emptyset$, add all parclusters on the path between $C_i$ and $C_j$ to $J'$ (if not already part of $J'$) and set $Q' = Q' \setminus C_i$
  • until $Q' = \emptyset$
Query Answering in FO jtrees

• Given an FO jtree $J$ with messages passed
  • Prepared for query answering

• For each query with query terms $Q$
  • Find subgraph $J'$ s.t. $Q \subseteq rv(J')$
  • Collect information from local models and outside messages, i.e,
    \[ G_Q = \bigcup_{i \in J'} G_i \cup \bigcup_{j \in nbs(i) \atop i \in J', j \notin J'} m_{ji} \]
  • Call $LVE(G_Q, Q, \emptyset)$ and return or store result of the call
Example

• E.g., \( P(\text{Travel(eve)}, \text{Treat(eve, } m_1)) \)
  • Subgraph: \( c_2, c_3 \)
  • Submodel for query answering: \( G_Q = (g_2, g_3, m_{12}) \)

\[
\begin{array}{c}
\text{Epid} \\
g'_1 \\
\text{Travel}(X) \\
g_2 \\
\text{Sick}(X) \\
g_3 \\
\text{Treat}(X, M)
\end{array}
\]

• Call LVE with \( G_Q \) and \( Q = \{\text{Travel(eve)}, \text{Treat(eve, } m_1)\} \)
  • Split off query terms
  • Eliminate all non-query terms
  • Normalise the result
Complexity & Runtimes

• With conjunctive queries, complexity for answering a single query depends on the size of the subtree $n_j$.
  • $O\left(n_j \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r^w#}\right)$
  • Assumption is that query terms occur close together and therefore $n_j$ hopefully small

• Runtime behaviour observable in implementation
  • Increasing $n_j$ on x-axis
  • Runtimes in milliseconds
  • More parclusters needed, runtimes increase
    • Closer to compile time
    • Closer to LVE time

![Graph showing runtimes for different parcluster sizes and LVE times](image)
Interim Summary

• Motivation
  • Find clusters that are enough for query answering

• FO jtree
  • From FO dtree clusters to FO jtree parclusters

• LJT algorithm
  • Propagation/message passing: Dynamic programming

• Complexity
  • Compared to LVE
    • Overhead for construction, message passing
    • Savings during query answering

• Completeness
  • Results for LVE hold as well

• Implementation

• Conjunctive queries
  • Find subgraph covering the query terms