Problem Solving by Inference

Basic Principles

• Inference by Variable Elimination
• Inference by Factor Elimination (Tree-clustering, Jointree)
• Inference by Recursive Conditioning (Decomposition)
The Inference Problem

- Answer queries about a function which is given in factored form.

- Logic/Constraints: Boolean functions
- Probability: Probability functions

- Techniques apply to other types of inference: belief functions, penalty logics, etc.

Bayesian Networks

Bayesian Network

| B | H | Pr(H|B) |
|---|---|--------|
| t | t | .4     |
| t | f | .6     |
| . | . | .      |

Probability Function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>Pr(.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>.004</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>f</td>
<td>.001</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>.020</td>
</tr>
</tbody>
</table>
**Probabilistic Inference**

**Bayesian Network**

**Probability Function**

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G & H & \text{Pr(.)} \\
t & t & t & t & t & t & t & .004 \\
t & t & t & t & t & t & f & .001 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
f & f & f & f & f & f & f & .020 \\
\end{array}
\]

- \( \text{Pr}(A=t \text{ or } H=f) \)
- MPE: row \( r \) with max \( \text{Pr}(r) \)
- MAP: partial instantiation \( i \) with max \( \text{Pr}(i) \)

---

**Propositional Knowledge Bases**

**Propositional KB**

\[
\begin{align*}
A \& \text{ok}_X & \Rightarrow \neg B \\
\neg A \& \text{ok}_X & \Rightarrow B \\
B \& \text{ok}_Y & \Rightarrow \neg C \\
\neg B \& \text{ok}_Y & \Rightarrow C
\end{align*}
\]

**Boolean Function**

\[
\begin{array}{cccccccc}
A & B & C & \text{OK}_X & \text{OK}_Y & F(\cdot) \\
T & T & T & T & T & 0 \\
T & T & T & T & F & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
F & F & F & F & F & 1 \\
\end{array}
\]

- OK\(_Y\) B C F(\(_\cdot\))
  - T T T 1
  - T F F 0
  - . . . .
  - F F F 1

---

A. Darwiche
Propositional Knowledge Bases

Propositional KB

| A & ok_X => ¬B | B & ok_Y => ¬C |
| ¬A & ok_X => B | ¬B & ok_Y => C |

Boolean Function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>OK_X</th>
<th>OK_Y</th>
<th>F(…)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Is there a satisfying assignment?

How many satisfying assignments

Are two KBs equivalent?

A. Darwiche

Constraint Satisfaction

Example: map coloring

Variables - countries (A,B,C, etc.)
Values - colors (e.g., red, green, yellow)
Constraints:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>red</td>
<td>0</td>
</tr>
<tr>
<td>red</td>
<td>green</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

A. Darwiche
Constraint Satisfaction

Example: map coloring
Variables - countries (A, B, C, etc.)
Values - colors (e.g., red, green, yellow)
Constraints:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>red</td>
<td>red</td>
<td>red</td>
<td>F()</td>
</tr>
<tr>
<td>1</td>
<td>red</td>
<td>red</td>
<td>green</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>yellow</td>
<td>0</td>
</tr>
</tbody>
</table>

Are the constraints consistent?
Find a solution, find all solutions
Count all solutions

Inference by Variable Elimination

A. Darwiche
Basic Principle

**Reduce**: A query about a function 
\[ f = f_1 f_2 \ldots f_m \]
over \( n \) variables into a query about a function \( f' \) over \( n-1 \) variables.

**Eliminate** a variable \( X \) from \( f \), while keeping the result, \( f' \), as factored as possible.

If variable \( X \) appears in only one factor \( f_1 \) of \( f \), all we have to do is replace \( f_1 \) with \( \text{elm}(f_1, X) \):
\[ f' = \text{elm}(f_1, X) f_2 \ldots f_m \]

If variable \( X \) appears in more than one factor, say, \( f_1 \) and \( f_2 \), we must combine (multiply) them first before eliminating \( X \):
\[ f' = \text{elm}(f_1 f_2, X) f_3 \ldots f_m \]

**Notes:**
- The more factors we have to combine, the less factored the result is.
- The order in which we eliminate variables matters only computationally.
### Elimination in Probability

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>.1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>.2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>.2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>.5</td>
</tr>
</tbody>
</table>

Eliminate B
Project on A

<table>
<thead>
<tr>
<th>A</th>
<th>F(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.3</td>
</tr>
<tr>
<td>F</td>
<td>.7</td>
</tr>
</tbody>
</table>

### Elimination in Probability

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>.1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>.2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>.2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>.5</td>
</tr>
</tbody>
</table>

Eliminate B
Project on A

<table>
<thead>
<tr>
<th>A</th>
<th>F(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.3</td>
</tr>
<tr>
<td>F</td>
<td>.7</td>
</tr>
</tbody>
</table>
Elimination in Probability

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>.1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>.2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>.2</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>.5</td>
</tr>
</tbody>
</table>

Eliminate B
Project on A

\[ F(A) = \sum_B F(AB) \]

Different notions of elimination: sum out, max out
Each preserves ability to answer a different type of query
Sum out: compute probabilities  Max out: compute MPE
Sum then Max: compute MAP

A. Darwiche
Variable Elimination Example

Multiply factors
To eliminate A

Variable Elimination Example

Eliminate A
To eliminate A
Variable Elimination Example

To eliminate B

B | f_5(B)
---|---
T  | .62
F  | .38

Multiply factors

B  C  | f_5(BC)
---  ---  |---
T  T  | .3
T  F  | .7
F  T  | .5
F  F  | .5

A. Darwiche
Variable Elimination Example

A → B → C

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>( f_6(BC) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>.186</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>.434</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>.190</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>.190</td>
</tr>
</tbody>
</table>

Eliminate B

Variable Elimination Example

A → B → C

<table>
<thead>
<tr>
<th>C</th>
<th>( f_7(C) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.376</td>
</tr>
<tr>
<td>F</td>
<td>.624</td>
</tr>
</tbody>
</table>
Elimination in Logic

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F(ABC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

A => B, B => C

Eliminate B

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>F(AC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

A => C

Project on A C

A. Darwiche
Elimination in Logic

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F(ABC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

A \Rightarrow B, B \Rightarrow C

Eliminate B

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>F(AC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

A \Rightarrow C

Elimination in Logic

A \Rightarrow B, B \Rightarrow C

\[ \Delta \]

\[ \exists B. \Delta \]

\~A or B

\~B or C

Resolve on B

\~A or B

\~B or C

\~A or C

Throw out B clauses

A \Rightarrow B, B \Rightarrow C

\[ \Delta \]

\[ \exists B. \Delta \]

\~A or C

\[ \exists B. \Delta \]

A \Rightarrow C

A. Darwiche
Elimination in Logic

• Different notions of elimination:
  – Existential elimination (quantification)
  – Universal elimination (quantification)
• Preserve ability to answer different queries
  – E.g. Existential preserves SAT
• Different types of eliminations can be mixed to solve more sophisticated problems: diagnosis and planning

Interaction Graphs

• Can simulate variable elimination on connectivity graph:
  – Node for every variable
  – Edge between two nodes iff appear in same factor

\[ f(ABCDEF) = f_1(AB) f_2(BCD) f_3(CF) f_4(EF) \]
Interaction Graph: Example

\[
\begin{align*}
A & \land \text{ok}_X \Rightarrow \neg B \\
\neg A & \land \text{ok}_X \Rightarrow B \\
B & \land \text{ok}_Y \Rightarrow \neg C \\
\neg B & \land \text{ok}_Y \Rightarrow C
\end{align*}
\]

Also known as primal graph in constraint satisfaction
Other types of graphs can be defined: dual & hyper

Interaction Graphs

- Eliminate F:
  \[
  f(ABCDEF) = f_1(AB) \cdot f_2(BCD) \cdot f_3(CF) \cdot f_4(EF)
  \]

  \[
  f'(ABCDE) = f_1(AB) \cdot f_2(BCD) \cdot f_{34}(CE)
  \]

\[\text{Interaction graph for } f \quad \text{Interaction graph for } f'\]
Interaction Graphs

• Eliminate C:

\[ f'(ABCDEF) = f_1(AB) f_2(BCD) f_{34}(CE) \]

\[ f''(ABCDEF) = f_1(AB) f_{234}(BDE) \]

Interaction graph for \( f' \)

Connect neighbors of C
Remove node C

Interaction graph for \( f'' \)

Induced Width / Treewidth

• The factor constructed upon eliminating variable X will be over X’s neighbors in interaction graph.

• If w is largest number of neighbors encountered when eliminating variables according to order O, then w is called the induced width of elimination order O.

• The complexity of variable elimination is \( O(n \exp(w)) \), where n is number of variables

• The induced width of a graph is width of its best elimination order

• This is also known as treewidth in graph theoretic literature
Treewidth

Higher treewidth

Bucket Elimination
A mechanism for managing variable elimination

\[ f(ABCDEF) = f_1(AB) \cdot f_2(BCD) \cdot f_3(CF) \cdot f_4(EF) \]

<table>
<thead>
<tr>
<th>Var</th>
<th>Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>( f_1(AB) )</td>
</tr>
<tr>
<td>B</td>
<td>( f_2(BCD) )</td>
</tr>
<tr>
<td>C</td>
<td>( f_3(CF) )</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>( f_4(EF) )</td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Var</th>
<th>Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>( f_2(BCD) ), ( f_5(B) )</td>
</tr>
<tr>
<td>C</td>
<td>( f_3(CF) )</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>( f_4(EF) )</td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
**Bucket Elimination**

A mechanism for managing variable elimination

\[ f(ABCDEF) = f_1(AB) \cdot f_2(BCD) \cdot f_3(CF) \cdot f_4(EF) \]

<table>
<thead>
<tr>
<th>Var</th>
<th>Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>( f_2(BCD) \cdot f_1(B) )</td>
</tr>
<tr>
<td>C</td>
<td>( f_3(CF) )</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>( f_4(EF) )</td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

Elm B

<table>
<thead>
<tr>
<th>Var</th>
<th>Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>( f_3(CF) \cdot f_6(CD) )</td>
</tr>
<tr>
<td>D</td>
<td>( f_4(EF) )</td>
</tr>
<tr>
<td>E</td>
<td>( f_4(EF) )</td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

**Another View of Treewidth**

Interaction graph

Tree decomposition

A. Darwiche
Tree Decomposition

- A tree of clusters

- Every pair of variables connected by an edge in interaction graph must appear together in some cluster

- If a variable appears in two clusters, it must appear in all clusters on the path between them

Tree Decomposition

- A tree of clusters

- The variables of every factor must appear in some cluster

- If a variable appears in two clusters, it must appear in all clusters on the path between them
Tree Decomposition

- Width of tree decomposition is size of its largest cluster - 1
- Width of best tree decomposition is treewidth of connectivity graph
- Treewidth quantifies the resemblance of a graph to a tree structure
- Tree decompositions correspond to jointrees (junction trees)

Inference by Factor Elimination

(Jointree algorithm)
(Tree clustering algorithm)
Basic Principle

**Reduce**: A query about a function
\[ f = f_1 f_2 \ldots f_m \]
into a query about a function \( f' \) over \( m-1 \) factors.

*Eliminate* factor \( f_i \) while keeping result, \( f' \), as factored as possible

Allows \( n \) queries to be answered in \( O(n \exp(w)) \) time instead of \( O(n^2 \exp(w)) \) [Standard VE]

Basic Principles

- **To eliminate a factor** \( f_1 \):
  - Eliminate all variables appearing in \( f_1 \) but not in other factors \( f_2, \ldots, f_m \)
  - Multiply resulting factor into one of \( f_2, \ldots, f_m \)
- **To control elimination**:
  - Choose a factor to eliminate
  - Decide on which factor to multiply into
- **Factor elimination is controlled by an elimination tree** (instead of elimination order)
Elimination Trees

- Spanning tree of factors
- Any spanning tree will do, some lead to more work than others

Factor Elimination

Eliminate factor $f_5$
Factor Elimination

Eliminate factor $f_2$

Factor Elimination

Eliminate factor $f_1$
Factor Elimination

\[ f(AC) = \sum_{BD} f(ABCD) \]

Eliminate factor \( f_4 \)

Resulting factor represents projection of original factors on variables AC

\[
f(AC) = \sum_{BDE} f(ABCDE) = \sum_{BDE} f(AB)f(A)f(AC)f(BCF)f(CE)
\]
Separators

\[ f(A) \]

\[ f(AB) \]

\[ f(AC) \]

\[ f(ACE) \]

\[ f(ABCD) \]

\[ f(BCD) \]

\[ f(CE) \]
Separators

1. $f(A)$
2. $f(AB)$
3. $f(AC)$
4. $f(BCD)$
5. $f(CE)$

Messages over Separators

1. $f(A)$
2. $f(AB)$
3. $f(AC)$
4. $f(BCD)$
5. $f(CE)$

$m(AB)$
$m(AC)$
$m(C) = \sum_{E} f(CE)$
Clusters

Cluster of node: variables of its factor union neighboring separators

Width of elm tree: size of largest cluster - 1
Complexity

- To project on variables of a cluster, choose cluster as root
- Number of messages passed is twice number of edges
- Sending a message from a cluster is exponential in size of cluster
- With appropriate elim tree, all messages passed in $O(n \exp(w))$

Tree Decomposition $\Leftrightarrow$ Elm Tree

Every tree decomposition of width $w$
embeds
an elimination tree of width $w$
Tree Decomposition $\mapsto$ Elm Tree

Every elimination tree of width $w$ induces a tree decomposition of width $w$
Elm Tree $\leftrightarrow$ Tree Decomposition

Interaction graph

Elimination tree

Tree decomposition

Inference by Recursive Conditioning

A. Darwiche
**Basic Principle**

Reduce: A query about a function
\[ f = f_1 f_2 \ldots f_m \]
into queries about a decomposition
\[ f_L = f_1 \ldots f_i \quad f_R = f_{i+1} \ldots f_m \]

Must condition on variables shared by \( f_L \) and \( f_R \)

Allows inference in \( O(n \exp(w)) \) time
Facilitates time-space tradeoffs

**Decomposition**
Case Analysis

Case I

Case II

LP * RP

A. Darwiche
Case Analysis

LP * RP + LP * RP

Case Analysis

LP * RP + LP * RP
• Decomposition and Case Analysis can answer any query
• Non-Deterministic!
Decomposition Tree

A. Darwiche
Decomposition Tree

A → B → C → D → E

B: \( LP \times RP \)

Decomposition Tree

A → B → C → D → E

A → B → C → D

B: \( LP \times RP + LP \times RP \)

A. Darwiche
Computational Complexity

Ancestoral Cutset

Decomposition Tree
Decomposition Tree

\[ \text{Context}(N) = \text{A-Cutset}(N) \& \text{Vars}(N) \]

A. Darwiche
Computational Complexity

- Given
  - DAG with \( n \) nodes
  - elimination order of width \( w \)

- Can construct a dtree, such that
  - time complexity: \( O(n \exp(w \log n)) \)
    space complexity \( O(n) \) [no caching]

- Can construct a dtree, such that
  - time complexity: \( O(n \exp(w)) \)
    space complexity: \( O(n \exp(w)) \) [full caching]
Recursive Conditioning in Logic/CSP

Graphical Models

Variable Elimination
Elimination Order
Factor Elimination
Jointree/Elm trees

Dtreet
Recursive Conditioning

Width preserving transformations
Elimination Orders to dtrees

F E A B C D \( W=2 \)

\[ \begin{align*}
A & \quad AB & \quad AC & \quad ABE & \quad BCD & \quad DF \\
A & \quad BC & \quad D & \quad & \quad & \quad
\end{align*} \]

Cutset \( \leq w+1 \)

Elimination Orders to dtrees

F E A B C D \( W=2 \)

\[ \begin{align*}
A & \quad AB & \quad AC & \quad ABE & \quad BCD & \quad DF \\
A & \quad BC & \quad D & \quad & \quad & \quad
\end{align*} \]

A. Darwiche
Elimination Orders to dtrees

F E A B C D W=2

Elimination Orders to dtrees

F E A B C D W=2
Elimination Orders to dtrees

F E A B C D  W=2

Dtrees to Jointree

F E A B C D  W=2
The problem of hypergraph partitioning is well-studied in VLSI design. ....and is alive!

A hypergraph is a generalization of a graph, such that an edge is permitted to connect an arbitrary number of vertices, rather than exactly two.

The task of hypergraph partitioning is to find a way to split the vertices of a hypergraph into k approximately equal parts, such that the number of hyperedges connecting vertices in different parts is minimized.

A. Darwiche
Hypergraph Partitioning => dtrees

For DAG:

dtrees to Elimination Orders

A. Darwiche
Graphical Models

Variable Elimination
Elimination Order

Factor Elimination
Jointree/Elm trees

Dtree
Recursive Conditioning

Width preserving transformations

Summary

• Inference by variable elimination:
  Elimination orders
• Inference by factor elimination:
  Elimination trees / tree decompositions / cluster trees / jointrees
• Inference by recursive conditioning:
  Decomposition trees (dtrees)
• Treewidth guarantees shared between all methods
Knowledge Compilation

- **Representational factorization**: Compact representation/specification of a function (Bayesian network, CNF, CSP, …)

- **Computational factorization**: A representation of a function on which (some) inference can be performed in polytime

- **Knowledge Compilation**: Convert a representational factorization into a (smallest) computational factorization
KC in Logic: Boolean Circuits
(NNFs / AND-OR Graphs)

Negation Normal Form
Decomposability
Determinism
Smootheness
Flatness
Decision
Ordering

Polytime Operations
Consistency (CO)
Validity (VA)
Clausal entailment (CE)
Sentential entailment (SE)
Implicant testing (IP)
Equivalence testing (EQ)
Model Counting (CT)
Model enumeration (ME)

Projection (existential quantification)
Conditioning
Conjoin, Disjoin, Negate

Succinctness

Example from Diagnosis

\[ KB = \begin{cases} 
A \land \text{ok}_X \Rightarrow \neg B \\
\neg A \land \text{ok}_X \Rightarrow B \\
B \land \text{ok}_Y \Rightarrow \neg C \\
\neg B \land \text{ok}_Y \Rightarrow C 
\end{cases} \]

Is A, \( \neg C \) a normal device behavior?
Is A, ¬C a normal device behavior?

Example from Diagnosis

Satisfiability Algorithm

Example from Diagnosis

Satisfiability Algorithm

A. Darwiche
Example from Diagnosis

\[
\begin{align*}
A & \& ok_X \Rightarrow \neg B \\
\neg A & \& ok_X \Rightarrow B \\
B & \& ok_Y \Rightarrow \neg C \\
\neg B & \& ok_Y \Rightarrow C \\
A, \neg C, ok_X, ok_Y
\end{align*}
\]

How many satisfying assignments?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>OK_X</th>
<th>OK_Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Counting Algorithm

Knowledge Compilation

\[
\begin{align*}
A & \& ok_X \Rightarrow \neg B \\
\neg A & \& ok_X \Rightarrow B \\
B & \& ok_Y \Rightarrow \neg C \\
\neg B & \& ok_Y \Rightarrow C
\end{align*}
\]

Compiler

Compiled Structure

Queries

Evaluator

A. Darwiche
Knowledge Compilation

A & ok_X => ¬B
¬A & ok_X => B
B & ok_Y => ¬C
¬B & ok_Y => C

Compiler

Queries

Evaluator

A. Darwiche

Knowledge Compilation

A & ok_X => ¬B
¬A & ok_X => B
B & ok_Y => ¬C
¬B & ok_Y => C

Compiler

Prime Implicates
OBDD

Queries

Evaluator

A. Darwiche
Knowledge Compilation

Negation Normal Form
- Decomposability
- Determinism
- Smoothness
- Flatness
- Decision
- Ordering

Polytime Operations
- Consistency (CO)
- Validity (VA)
- Clausal entailment (CE)
- Sentential entailment (SE)
- Implicant testing (IP)
- Equivalence testing (EQ)
- Model Counting (CT)
- Model enumeration (ME)
- Projection (existential quantification)
- Conditioning
- Conjoin, Disjoin, Negate

Succinctness

Applications

- Diagnosis
  - Is this a normal behavior?
  - What are the possible faults?
- Planning
  - Can this goal be achieved?
  - Generate a set of plans
- Probabilistic reasoning
  - What is the probability of X given Y
- Non-monotonic reasoning (penalty logics)
  - Does X follow preferentially from Y
- Formal verification / CAD:
  - Is it possible that the design will exhibit behavior X?
  - Are two designs equivalent?
Negation Normal Form (NNF)

\[-A \lor B \land \neg B \land A\]
\[-C \lor D \land \neg D \land \neg C\]

rooted DAG

Negation Normal Form

Decomposability
Determinism
Smoothness
Flatness
Decision
Ordering

A. Darwiche
Nested vs Flat languages

\[(X \land Y \land Z) \lor (Z \land \neg X \land \neg Y) \lor (Y \land Z \land \neg X) \lor (\neg X \land \neg Y \land \neg Z)\]
NNF Subsets

NNF

- d-NNF
- s-NNF
- DNNF
- f-NNF

d-DNNF

sd-DNNF

VA, IP, CT

CO, CE, ME

EQ?

Counting

Given A, ¬B
Simple Conjunction

\[ \text{or} \]
\[ \text{and} \]
\[ \text{and} \]
\[ \text{and} \]
\[ \text{and} \]
\[ X \]
\[ Y \]
\[ Z \]
\[-X \]
\[-Y \]
\[-Z \]

Implies decomposability

Simple Disjunction

\[ \text{and} \]
\[ \text{or} \]
\[ \text{or} \]
\[ \text{or} \]
\[ X \]
\[ Y \]
\[ Z \]
\[-X \]
Decision implies determinism
NNF Subsets

- d-NNF
- s-NNF
- f-NNF
- BDD
- sd-DNNF
- DNNF
- CO, CE, ME
- VA, IP, CT
- EQ?
- DNF
- CNF
- VA, IP, SE, EQ
- CO, CE, ME, VA, IP, SE, EQ
- IP
- PI

Decision Nodes

- or
- and
- and
- X
- α
- ¬X
- β
- X
- α
- β
Binary Decision Diagram

$$(X_2 \land X_3) \lor (X_1 \land \neg X_2 \land \neg X_3)$$

NNF Subsets

A. Darwiche
Free Binary Decision Diagram

In the context of decision: Test-once = decomposability
Language Succinctness

L1 at least as succinct as L2

\[ L_1 \leq L_2 \]

Size \( p(n) \) \quad Size \( n \)

L1 is more succinct than L2

\[ L_1 < L_2 \]

\[ \text{sd-DNNF} = \text{d-DNNF} \]

\[ \text{DNF} \leq \text{OBDD} \]

\[ \text{OBDD} \leq \text{DNF} \]

\[ \text{PI} \leq \text{DNNF} \]

A. Darwiche
Tractability vs Succinctness

Propositional Transformations

- Project (Forget)
- Condition
- Conjoin
- Disjoin
- Negate
Projection

\[ KB = \begin{cases} 
A \land \text{ok}(X) \Rightarrow \neg B \\
\neg A \land \text{ok}(X) \Rightarrow B \\
B \land \text{ok}(Y) \Rightarrow \neg C \\
\neg B \land \text{ok}(Y) \Rightarrow C 
\end{cases} \]

A. Darwiche

Projection

\[ KB = \begin{cases} 
A \land \text{ok}(X) \Rightarrow \neg B \\
\neg A \land \text{ok}(X) \Rightarrow B 
\end{cases} \]

A. Darwiche
**Projection**

\[
\text{A} \lor \neg \text{ok}(X) \implies \neg \text{B} \\
\neg \text{A} \lor \neg \text{ok}(X) \implies \text{B}
\]
Polytime Transformations

- Closure under conjunction: none!
  Bounded conjunction: OBDD, DNF, IP, MODs
- Closure under disjunction: DNF, DNNF
  Bounded disjunction: OBBD, PI
- Closure under negation: FBDD, OBDD
- Conditioning: all!
- Forgetting multiple variables: DNNF, DNF, PI, MODS
- Forgetting single variable: OBDD
A Knowledge Compiler: DPLL with a Trace

DPLL with fixed order and caching \( \Rightarrow \) OBDD compiler
DPLL with caching \( \Rightarrow \) FBDD compiler
**A Knowledge Compiler:**

DPLL with a Trace

DPLL with fixed order and caching → OBDD compiler
DPLL with caching → FBDD compiler
DPLL with caching and decomposition → d-DNNF compiler*

---

**Knowledge Compilation in Probability:**

Arithmetic Circuits

ADD (Algebraic Decision Diagrams)
*BMD (Binary Moment Diagrams)
PDGs (Probabilistic Decision Graphs)…
Conclusion

- Knowledge compilation converts a representational factorization into a computational factorization:
  - Target language (succinctness, tractability)

- Knowledge compilers can be constructed by keeping the trace of various search algorithms

- Another view of knowledge compilation is based on deductive closure

Knowledge Compilation as Deductive Closure: Two Examples

1. Eliminate all variables $X_1, \ldots, X_n$, while keeping all intermediate CNFs
   - The intermediate CNFs represent projections of initial CNF on subsets $X_2, \ldots, X_n$, $X_3, \ldots, X_n$, $X_{n-1}X_n$, $X_n$
   - The result can be used to answer some hard queries in polytime

2. Add enough clauses to make CNF complete under unit resolution
References

- Darwiche and Marquis. A knowledge compilation map. JAIR, 17, 229-264, 2002.