Logic and Probability
The Computational Connection

Adnan Darwiche
UCLA

Deduction at Scale Seminar, March, 2011
Inference

• Probabilistic Graphical Models:
  Marginal and conditional probabilities
  Most likely instantiations...

• Propositional Knowledge Bases:
  Logical entailment
  Existential quantification
  Model counting...
Two Main Themes

• **Exact inference as:**
  Enforcing decomposability and determinism on propositional knowledge bases

• **Approximate inference as:**
  Relaxing, compensating for, and recovering equivalence constraints (equalities)
Knowledge Compilation

(¬A v B v C)
(A v ¬D v E)
(B v C v ¬F)
(¬B v A v F)
...

Queries

Compiled Structure

Evaluator (Polytime)
Knowledge Compilation

(¬A ∨ B ∨ C)
(A ∨ ¬D ∨ E)
(B ∨ C ∨ ¬F)
(¬B ∨ A ∨ F)
...

Compiler

Queries

Subsets of NNF

Evaluator (Polytime)
Negation Normal Form

\[
\begin{align*}
\neg A & \quad B & \quad \neg B & \quad A \\
& \quad \text{and} \quad \text{or} & \quad \text{and} \quad \text{or} & \quad \text{and} \\
& \quad \text{or} & \quad \text{or} & \quad \text{or} & \quad \text{or} \\
C & \quad \neg D & \quad D & \quad \neg C \\
& \quad \text{and} \quad \text{and} \quad \text{and} \quad \text{and} \quad \text{and} \quad \text{and} \\
& \quad \text{rooted DAG (Circuit)}
\end{align*}
\]
Decomposability (DNNF)

No two children of AND share a variable
Determinism (d-DNNF)

Every pair of children of or-node are inconsistent (mutually exclusive)
OBDD: d-DNNF + Additional Properties

OBDD (traditional form)

OBDD (NNF)
Queries and Transformations

• Queries
  SAT, MAXSAT, logical entailment, equivalence testing, model counting,…

• Transformations:
  Existential quantification, conjunction, disjunction, negation,…

• More properties imply more polytime queries and transformations, but less succinctness
Counting Models (d-DNNF)
Counting Graph

\[ \neg A \quad B \quad \neg B \quad A \quad C \quad \neg D \quad D \quad \neg C \]
Counting Graph

\( S = \{A, \neg B\} \)
### Probabilistic Inference by Weighted Model Counting

#### Diagram

- **A** → **B** → **C**
  - $\theta_A$
  - $\theta_B | A$
  - $\theta_C | A$

#### Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$\text{Pr}(.)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$\theta_A \theta_B</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$\theta_A \theta_B</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$\theta_A \theta_{\neg B</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$\theta_A \theta_{\neg B</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$\theta_{\neg A} \theta_B</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$\theta_{\neg A} \theta_{\neg B</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$\theta_{\neg A} \theta_{\neg B</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$\theta_{\neg A} \theta_{\neg B</td>
</tr>
</tbody>
</table>
### Probabilistic Inference by Weighted Model Counting

**Pr(α) = \text{wmc}(Δ \land α)**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Pr(\cdot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>(\theta_A \theta_B \theta_C)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>(\theta_A \theta_B \theta_C)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>(\theta_A \theta_B \theta_C)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>(\theta_A \theta_B \theta_C)</td>
</tr>
</tbody>
</table>

\[A \land C \leftrightarrow \theta_{C|A}\]
\[A \land \neg C \leftrightarrow \theta_{\neg C|A}\]
\[A \land B \leftrightarrow \theta_{B|A}\]
\[A \leftrightarrow \theta_A\]
\[\neg A \leftrightarrow \theta_{\neg A}\]
Weighted Model Counting
(Arithmetic Circuits)
Why Logic?

• Encoding local structure is easy:
  
  – Zero-parameters encoded by adding clauses:

  \[
  \theta_{C|A} = 0 \quad \neg A \lor \neg C
  \]

  – Context-specific independence encoded by collapsing variables:

  \[
  \theta_{C|AB} = \theta_{C|A \neg B}
  \]
• Relational networks (251 networks)
  • Average clique size is 50
Alchemy - Open Source AI

Welcome to the Alchemy system! Alchemy is a software package providing a series of algorithms for statistical relational learning and probabilistic logic inference, based on the Markov logic representation. Alchemy allows you to easily develop a wide range of AI applications, including:

- Collective classification
- Link prediction
- Entity resolution
- Social network modeling
- Information extraction

If you are not already familiar with Markov logic, we recommend that you first read the paper Unifying Logical and Statistical AI.

Alchemy is a software package providing a series of algorithms for statistical relational learning and probabilistic logic inference, based on Markov logic representations.

- Generative weight learning
- Structure learning
- MAP/MPE inference (including memory efficient)
- Probabilistic inference: MC-SAT, Gibbs Sampling, Simulated Tempering, Belief Propagation (including lifted)
- Support for native and linked-in functions
- Block inference and learning over variables with mutually exclusive and exhaustive values
- EM (to handle ground atoms with unknown truth values during learning)
- Specification of indivisible formulas (i.e. formulas that should not be broken up into separate clauses)
- Support of continuous features and domains
- Online inference
- Decision Theory

In the next release we plan to include:

- Online learning
- Exact inference for small domains
Current Challenges

• **Incremental compilation:**
  – What? Current compilers monolithic: c2d (UCLA) and DSharp (Toronto)
  – Need:
    • Logic: planning and verification applications
    • Probability: approximate inference
  – Main insight:
    • Structured decomposability & vtrees (AAAI-08, AAAI-10)

• **Guarantees and Complexity results:**
  – Upper & lower bounds on size of compilation (AAAI-10, ECAI-10)
  – Main insights:
    • The notion of a decomposition (AAAI-10)
    • The notion of an interaction function (ECAI-10)
Structured Decomposability

vtree T

Full binary tree with leaves corresponding to variables

DNNF respects T
OBDD: DNNF that Respects Linear vtrees
Decomposition of Boolean Functions (AAAI-10)

- Examples: \( f = (X_1 \lor X_2) \land (Y_1 \lor X_2) \land (X_1 \lor Y_2) \land (Y_1 \lor Y_2) \lor (X_2 \land Y_3) \)
  - \( X = \{X_1, X_2\}, Y = \{Y_1, Y_2, Y_3\} \):

\[
f(X,Y) = g(X) \land h(Y)
\]

\[
f(X,Y) = f_1 \lor f_2 \lor f_3 \lor \ldots \lor f_m
\]

\[
\begin{align*}
g_1(X) & \land h_1(Y) \\
g_2(X) & \land h_2(Y) \\
g_3(X) & \land h_3(Y) \\
g_m(X) & \land h_m(Y)
\end{align*}
\]

\((X,Y)\)-decomposition of \( f \)
Lower Bounds (AAAI-10)

\[(\neg E \lor \neg F) \land (\neg A \land \neg B) \lor (D \land E) \land (F \land G) \land ((A \land (B \lor C)) \lor \neg A) \lor (F \land G) \land (\neg A \lor (\neg B \land \neg C))\]

\[(X, Y)\text{-decomposition of the function represented by DNNF}\]
The Interaction Function (ECAI-10)

\[ f(X,Y) = g(X) \land h(Y) \land I(X,Y) \]

Captures precisely knowledge about variables in \( X \)
Captures precisely knowledge about variables in \( Y \)
Captures precisely interaction between variables \( X \) and \( Y \)

\[ \exists Y f \]
\[ \exists X f \]
\[ f \lor \neg (\exists X f) \lor \neg (\exists Y f) \]
The Interaction Function (ECAI-10)

\[ f(X,Y) = g(X) \land h(Y) \land l(X,Y) \]

- Captures precisely knowledge about variables in \( X \)
- Captures precisely knowledge about variables in \( Y \)
- Captures precisely interaction between variables \( X \) and \( Y \)

\[ \exists Y f \quad \exists X f \quad (A \Rightarrow (C \Rightarrow B)) \Rightarrow (B \Rightarrow C) \]

(\( A \Rightarrow B \) \( \neg A \Rightarrow C \) \( X=\{A\} \) \( Y=\{B,C\} \))
Current Research

• Searching for good vtrees (on-going)
• Characterizing and searching for optimal decompositions
• Upper and lower bounds on size of DNNF

• Key objective: incremental compiler for DNNF and d-DNNF

• ???
Two Main Themes

• **Exact inference as:**
  Enforcing decomposability and determinism on propositional knowledge bases

• **Approximate inference as:**
  Relaxing, compensating for, and recovering equivalence constraints (equalities)
Try ACE - a companion system for networks exhibiting local structure: determinism and CSI

Samlam is a comprehensive tool for modeling and reasoning with Bayesian networks, developed in Java by the Automated Reasoning Group of Professor Adnan Darwiche at UCLA.

Samlam includes two main components: a graphical user interface and a reasoning engine. The graphical interface allows users to develop Bayesian network models and to save them in a variety of formats. The reasoning engine supports many tasks including: classical inference; parameter estimation; time-space tradeoffs; sensitivity analysis; and explanation-generation based on MAP and MPE.
Two Main Themes

• **Exact inference as:**
  Enforcing decomposability and determinism on propositional knowledge bases

• **Approximate inference as:**
  Relaxing, compensating for, and recovering equivalence constraints (equalities)
Treewidth

**CNF**

\[(A \lor B \lor \neg C) \land (A \lor \neg B \lor C) \land (C \lor \neg D \lor E) \land (B \lor D) \land (D \lor E)\]

**Constraint Graph**

![Graph](image-url)
Treewidth

$O(n \exp\{w\})$
Treewidth

CNF

\( (A \lor B \lor \neg C) \land \\
(\neg A \lor B \lor C) \land \\
(\neg C \lor D \lor E) \land \\
(\neg B \lor D) \land \\
(\neg D \lor E) \land \)

Constraint Graph

- A
- B
- C
- D
- E
Treewidth

CNF

$$\begin{align*}
(A \lor B \lor \neg C) \land \\
(A' \lor \neg B \lor C) \land \\
(C \lor \neg D \lor E) \land \\
(B \lor D) \land \\
(D \lor E) \land \\
(A = A')
\end{align*}$$

Constraint Graph
(A ∨ B ∨ ¬C) ∧
(A' ∨ ¬B ∨ C) ∧
(C ∨ ¬D ∨ E) ∧
(B ∨ D) ∧
(D ∨ E)
Equivalence Constraints

\[ \psi_{eq}(X_i = x_i, X_j = x_j) = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases} \]

\[ \Pr(x_1, \ldots, x_n) = \frac{1}{Z} \psi(x_1, x_5) \ldots \psi(x_2, x_4) \ldots \theta(x_1) \ldots \theta(x_n) \]
Relaxing Equivalence Constraints

- Model $\mathcal{M}$
Relaxing Equivalence Constraints

- Model + Eq.
Relaxing Equivalence Constraints

- Relaxed
- Treewidth 1
Relaxing Equivalence Constraints

• Model $\mathcal{M}$
Relaxing Equivalence Constraints

- Model + Eq.
Relaxing Equivalence Constraints

- Relaxed
- Decomposed
Model + Eq

Intractable model, augmented with equivalence constraints

Relax

Simplify network structure:
Relax equivalence constraints

Compensate

Compensate for relaxation:
Restore a weaker equivalence

Recover

Recover structure, identify an improved approximation
Model + Eq

Intractable model, augmented with equivalence constraints

Relax

Usually gives upper/lower bounds: mini-buckets, MAXSAT

Compensate

Compensate for relaxation: Restore a weaker equivalence

Recover

Recover structure, identify an improved approximation
Compensating for an Equivalence

\[
\begin{align*}
\theta(X_i) & \quad \theta(X_j) \\
| & |
\begin{array}{c|c}
X_i & \theta(X_i) \\
T & .4789 \\
F & .5211 \\
X_j & \theta(X_j) \\
T & .8273 \\
F & .1727
\end{array}
\end{align*}
\]
Compensating for an Equivalence

$\Pr(X_i = x) = \Pr(X_j = x)$
Compensating for an Equivalence

\[
\theta(X_i) = \alpha \frac{\partial Z}{\partial \theta(X_j)} \\
\theta(X_j) = \alpha \frac{\partial Z}{\partial \theta(X_i)}
\]

\[
\Pr(X_i = x) = \Pr(X_j = x)
\]
Parametrizing Edges Iteratively

Iteration $t = 0$

Initialization
Parametrizing Edges Iteratively

Iteration $t = 1$
Parametrizing Edges Iteratively

Iteration $t = 2$
Parametrizing Edges Iteratively

\[ \theta(X_i) = \alpha \frac{\partial Z}{\partial \theta(X_j)} \]

\[ \theta(X_j) = \alpha \frac{\partial Z}{\partial \theta(X_i)} \]
Characterizing Loopy Belief Propagation

\[
\theta(X_i) = \alpha \frac{\partial Z}{\partial \theta(X_j)} \quad \theta(X_j) = \alpha \frac{\partial Z}{\partial \theta(X_i)}
\]

Iteration $t$
Which Edges to Delete?
Edge Recovery

loopy BP

recover edges

recover edges

exact
Edge Recovery

Recover edges with largest $MI(X_i;X_j)$
## Evaluation Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>PR</th>
<th>MAR</th>
<th>MPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSP</td>
<td>8</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>Grids</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Image Alignment</td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Medical Diagnosis</td>
<td>26</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Object Detection</td>
<td>96</td>
<td>96</td>
<td>92</td>
</tr>
<tr>
<td>Pedigree</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Protein Folding</td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>Protein-Protein Interaction</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Segmentation</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>204</td>
<td>204</td>
<td>287</td>
</tr>
</tbody>
</table>
## Overall Results

### PR Task: 20 Seconds

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>edbr</td>
<td>1.7146</td>
</tr>
<tr>
<td>vgogate</td>
<td>2.1620</td>
</tr>
<tr>
<td>libDai</td>
<td>2.2775</td>
</tr>
</tbody>
</table>

### MAR Task: 20 Seconds

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>edbq</td>
<td>0.2390</td>
</tr>
<tr>
<td>libDai2</td>
<td>0.3064</td>
</tr>
<tr>
<td>vgogate</td>
<td>0.4409</td>
</tr>
</tbody>
</table>
Ideally...

• Exact inference based on compiling CNFs

• Edge recovery using incremental compilation:
  – conjoin recovered equivalence constraint with current compilation

• Not there yet: more engineering needed!
Key Ideas

- **Approximate inference**: formulated as exact inference in an approximate model
- **Approximate models**: obtained by relaxing and compensating for equivalence constraints
- **Anytime inference**: selective recovery of equivalence constraints
- **Exact inference**: formulated in terms of enforcing decomposability and determinism of propositional knowledge bases
Adnan Darwiche

MODELING AND REASONING with

BAYESIAN NETWORKS

Cambridge
Try ACE - a companion system for networks exhibiting local structure: determinism and CSI

Samiam is a comprehensive tool for modeling and reasoning with Bayesian networks, developed in Java by the Automated Reasoning Group of Professor Adnan Darwiche at UCLA.

Samiam includes two main components: a graphical user interface and a reasoning engine. The graphical interface allows users to develop Bayesian network models and to save them in a variety of formats. The reasoning engine supports many tasks including: classical inference; parameter estimation; time-space tradeoffs; sensitivity analysis; and explanation-generation based on MAP and MPE.