Logic and Bayesian Networks
Part 3: Knowledge Compilation

Jinbo Huang
Negation Normal Form (NNF)

AND/OR circuit over literals
Any Boolean function can be represented in NNF (completeness)

Impose conditions/properties over NNF

Obtain subsets/restrictions of NNF

Consider only complete subsets
Decomposability
Decomposable NNF (DNNF)
Determinism

\begin{align*}
\text{or} & \quad \text{and} \\
\text{or} & \quad \text{and} \\
\text{or} & \quad \text{and} \\
\neg A & \quad B & \neg B & \quad A \\
C & \quad \neg D & \quad D & \quad \neg C
\end{align*}
Smoothness

\[ \text{or} \quad \text{and} \quad \text{or} \quad \text{and} \quad \text{and} \]

\[ \text{or} \quad \text{and} \quad \text{and} \quad \text{and} \quad \text{and} \]

\[-A \quad B \quad \neg B \quad A \]

\[C \quad \neg D \quad D \quad \neg C\]
Deterministic DNNF (d-DNNF)
Flatness

or

and

X

and

Y

and

Z

and

-X

and

-Y

and

-Z
Simple Disjunction

\[ \text{and} \]

\[ \text{or} \quad \text{or} \quad \text{or} \]

\[ X \quad Y \quad Z \quad -X \]
Simple Conjunction

[Diagram of a logical expression with nodes and edges labeled with variables and negations.]
CNF, DNF

Diagram:

- d-NNF
- s-NNF
- NNF
- DNNF
- d-DNNF
- sd-DNNF
- CO, CE, ME
- VA, IP, CT
- EQ?
- DNF
- CNF
- f-NNF
Prime Implicates (PI)

Conjunction of clauses

Includes every entailed clause

No clause subsumes another
Conjunction of clauses

Includes every entailed clause

No clause subsumes another

Can obtain by running resolution to saturation, removing subsumed clauses
Prime Implicants (IP)

- Disjunction of terms
- Includes every entailing term
- No term subsumes another
Prime Implicants (IP)

- Disjunction of terms
- Includes every entailing term
- No term subsumes another
- Can obtain by process dual to resolution
Decision node: **true**, **false**, or following fragment where $\alpha, \beta$ are decision nodes

```
  or
    and      and
      X     $\alpha$    $\neg X$    $\beta$
```

Decision Drawn Compactly

\[
\begin{align*}
\text{or} & \\
\text{and} & \\
X & \alpha & \neg X & \beta \\
\text{and} & \\
\alpha & \beta
\end{align*}
\]
Binary Decision Diagrams (BDD)
Test-once
Free BDDs (FBDD)
Ordering
Ordered BDDs (OBDD)
Top-down approach

- Based on exhaustive search

Bottom-up approach

- Compiling subformulas, combining results
DPLL Search

\[
\begin{align*}
x & \lor y \\
\neg x & \lor \neg z \\
\neg w & \lor z \\
\neg v & \lor w \\
v & \lor w & \lor z
\end{align*}
\]

SAT?
DPLL Search

v = true

SAT?
DPLL Search

SAT?

\( v = \text{true} \)

\( x \lor y \)
\( \neg x \lor \neg z \)
\( w \lor z \)
\( \neg v \lor w \lor z \)

SAT?

\( v = \text{false} \)

\( x \lor y \)
\( \neg x \lor \neg z \)
\( w \lor z \)

SAT?
DPLL Search

Terminating condition?
Exhaustive DPLL

Search the space of all models
- Count models
- Generate representation of all models
Exhaustive DPLL

Search the space of all models

- Count models
- Generate representation of all models

Amount of work not necessarily proportional to number of models
The Language of Search

Exhaustive DPLL → Record Trace → Knowledge Compiler
Variations ↔ Languages
Trace of DPLL

\[
X \lor Y \\
X \lor \neg Y \lor \neg Z \\
\neg X \lor Y \lor \neg Z
\]
Trace of DPLL

Run to exhaustion
Trace of DPLL

\[
\begin{align*}
X &
\quad \text{unsat} &
\quad \text{sat} \\
Y &
\quad \text{unsat} &
\quad \text{sat} \\
Z &
\quad \text{unsat} &
\quad \text{sat} \\
\end{align*}
\]

\[
\begin{align*}
0 &
\quad \neg Y &
\quad Y &
\quad \neg Z &
\quad \neg Z &
\quad \neg Y &
\quad Y &
\quad 1 \\
\end{align*}
\]

\[
\begin{align*}
\text{and} &
\quad \text{and} \\
\text{or} &
\quad \neg X &
\quad X &
\quad \text{or} \\
\end{align*}
\]
Trace of DPLL

or
and and
or ¬X X or
and and and
and
0 ¬Y Y ¬Z ¬Z ¬Y Y 1
Trace of DPLL

Equivalent to original CNF

Tractable
(satisfies some properties)
Dealing with Redundancy

- Do not record redundant portions of trace.
- Try not to solve equivalent subproblems.

Jinbo Huang
Dealing with Redundancy

Do not record redundant portions of trace
Dealing with Redundancy

Do not record redundant portions of trace

Try not to solve equivalent subproblems
Dealing with Redundancy

![Graph with nodes labeled X, Y, Z, unsat, and sat connected by dotted lines.]

- Node X connected to Y and Z.
- Node Y connected to unsat.
- Node Z connected to unsat and another unsat.
- The graph concludes with sat as the final node.
Dealing with Redundancy
This is an OBDD!
Top-down OBDD Compiler

Exhaustive DPLL,
Fixed variable order,
Unique nodes

Compile

\[ X \lor Y \]
\[ X \lor \neg Y \lor \neg Z \]
\[ \neg X \lor Y \lor \neg Z \]
FBDD Compiler

Exhaustive DPLL, Dynamic variable order, Unique nodes

Compile

\[
\begin{align*}
X \lor Y \\
X \lor \neg Y \lor \neg Z \\
\neg X \lor Y \lor \neg Z
\end{align*}
\]
Dealing with Redundancy

Level 1: Unique nodes (done)

Level 2: Avoiding redundant compilation/search
Redundant Compilation

\[ x_5 \lor x_6 \]
\[ x_4 \lor \neg x_5 \lor x_6 \]
\[ x_1 \lor x_3 \lor x_4 \lor x_5 \]
\[ x_2 \lor x_3 \]
\[ x_1 \lor x_2 \lor \neg x_3 \]
Caching

\[ \text{OBDD}(\Delta) = v_1 \]

\[ \text{OBDD}(\Delta|_{v_1=0}) \quad \text{OBDD}(\Delta|_{v_1=1}) \]

\[ \lll \quad \lll \]

\[ v_2 \]

\[ \text{OBDD}(\Delta|_{v_1=0,v_2=0}) \quad \text{OBDD}(\Delta|_{v_1=1,v_2=1}) \]

...
Caching

Recursive calls may be made on equivalent subformulas
Caching

\[
\begin{align*}
\nu_5 + \nu_6 \\
\nu_4 + \neg \nu_5 + \nu_6 \\
\nu_1 + \nu_3 + \nu_4 + \nu_5 \\
\nu_2 + \nu_3 \\
\nu_1 + \nu_2 + \neg \nu_3 
\end{align*}
\]

<table>
<thead>
<tr>
<th>(\nu_1 \nu_2 \nu_3)</th>
<th>(\Delta')</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>contradiction</td>
</tr>
<tr>
<td>001</td>
<td>contradiction</td>
</tr>
<tr>
<td>010</td>
<td>(\nu_5 + \nu_6, \nu_4 + \neg \nu_5 + \nu_6, \nu_4 + \nu_5)</td>
</tr>
<tr>
<td>011</td>
<td>(\nu_5 + \nu_6, \nu_4 + \neg \nu_5 + \nu_6)</td>
</tr>
<tr>
<td>100</td>
<td>contradiction</td>
</tr>
<tr>
<td>101</td>
<td>(\nu_5 + \nu_6, \nu_4 + \neg \nu_5 + \nu_6)</td>
</tr>
<tr>
<td>110</td>
<td>(\nu_5 + \nu_6, \nu_4 + \neg \nu_5 + \nu_6)</td>
</tr>
<tr>
<td>111</td>
<td>(\nu_5 + \nu_6, \nu_4 + \neg \nu_5 + \nu_6)</td>
</tr>
</tbody>
</table>
After instantiation of $v_1, v_2, v_3$, $\Delta'$ is either contradictory, or determined by clause $c_3$ alone. $c_3$ can only be in one of two states: satisfied or shrunk to $v_4 + v_5$.
After instantiation of $v_1 v_2 v_3$, $\Delta'$ is either contradictory, or determined by clause $c_3$ alone.

- $c_3$ can only be in one of two states: satisfied or shrunk to $v_4 + v_5$.
$Cutset_i$: clauses having variable $\leq v_i$ and one $> v_i$
Caching

Cutset\(_i\): clauses having variable \(\leq v_i\) and one \(> v_i\)

After instantiation of \(v_1 v_2 \ldots v_i\), \(\Delta'\) is either contradictory, or determined by states of clauses Cutset\(_i\).
Caching

*Cutset*$_i$: clauses having variable $\leq v_i$ and one $> v_i$

After instantiation of $v_1 v_2 \ldots v_i$, $\Delta'$ is either contradictory, or determined by states of clauses $Cutset_i$;

$\#$ of distinct $\Delta' \leq 2|Cutset_i| + 1$
Caching

\textit{Cutset}_i: clauses having variable \( \leq v_i \) and one \( > v_i \)

After instantiation of \( v_1 v_2 \ldots v_i \), \( \Delta' \) is either contradictory, or determined by states of clauses \textit{Cutset}_i

\# of distinct \( \Delta' \leq 2|\textit{Cutset}_i| + 1 \)

Maintain a cache for each \( i \), use value of \textit{Cutset}_i (bit vector) as key
Complexity

For each $i$, $2^{|Cutset_i|}$ bounds
- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
Complexity

For each $i$, $2^{|\text{Cutset}_i|}$ bounds
- # of recursive calls OBDD($\Delta, i + 1$)
- # of entries in cache
For each $i$, $2^{|\text{Cutset}_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
- # of OBDD nodes labeled with $v_{i+1}$
For each $i$, $2^{|Cutset_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
- # of OBDD nodes labeled with $v_{i+1}$

Complexity bound on

- Time complexity of compilation
- Space complexity of compilation
- Size of OBDD
Complexity

For each $i$, $2^{|Cutset_i|}$ bounds

- # of recursive calls $\text{OBDD}(\Delta, i + 1)$
- # of entries in cache
- # of OBDD nodes labeled with $v_{i+1}$

Complexity bound on

- Time complexity of compilation
- Space complexity of compilation
- Size of OBDD

Linear in # of variables, exponential in cutwidth

- Size of largest cutset of variable order
Beyond BDDs

Plain DPLL $\Rightarrow$ FBDD

Fixed variable ordering $\Rightarrow$ OBDD
Beyond BDDs

Plain DPLL $\Rightarrow$ FBDD

Fixed variable ordering $\Rightarrow$ OBDD

Decomposition $\Rightarrow$ d-DNNF
Decomposition

\[
\begin{align*}
A \lor B \lor C & \quad \neg A \lor \neg B \lor C \\
A \lor D \lor E & \quad \neg A \lor \neg D \lor E
\end{align*}
\]
Decomposition
Decomposition Methods

Dynamically detect disjoint components
  - Most effective, but very expensive
Decomposition Methods

Dynamically detect disjoint components
  ▶ Most effective, but very expensive

Static structural analysis
  ▶ Constructs a decomposition tree (dtree)
  ▶ Does not detect all decompositions
  ▶ Low cost
Fixed variable ordering $\Rightarrow$ OBDD

Plain DPLL $\Rightarrow$ FBDD

Decomposition $\Rightarrow$ d-DNNF