Libra in a Nutshell

Libra is a command-line toolkit for:

Learning and
Inference in
Bayesian networks,
Random fields*, and
Arithmetic circuits.

* -- Support for random fields (MRFs, CRFs) is coming soon!
Libra in a Nutshell

- **Functionality**: Structure and weight learning, exact and approximate inference, etc.

- **Strengths**: Libra uses local structure and arithmetic circuits to perform efficient, exact inference, even in models with high treewidth.

- **Language**: OCaml

- **License**: Modified BSD
Outline

• Background
  – Graphical models
  – Arithmetic circuits
• Algorithms
  – Structure learning
  – Inference
• Implementation
• Demo
• Conclusion
Probabilistic Graphical Models

**Bayesian Networks**
Factors: Conditional probability distributions

\[ P(A,B,C,D) = P(A)P(B)P(C \mid A,B)P(D \mid B) \]

Graph: Directed, acyclic

**Markov Networks**
Factors: Arbitrary non-negative functions

\[ P(A,B,C,D) = \frac{1}{Z} \phi_1(A,B,C) \phi_2(B,D) \]

Graph: Undirected
Probabilistic Graphical Models

Probabilistic graphical models represent probability distributions as products of factors, e.g.:

$$P(A, B, C, D) = P(A) \times P(B) \times P(C|A, B) \times P(D|B)$$

...etc...
Local Structure: Decision Trees

\[ P(C|A,B) = \begin{array}{ccc}
A & B & P(C|A,B) \\
T & T & 0.1 \\
T & F & 0.1 \\
F & T & 0.4 \\
F & F & 0.6 \\
\end{array} \]

Now we can handle factors with over 100 variables.
Local Structure: Features

\[ \phi(A, B) = \begin{cases} 
4.48 & \text{if } A \land \neg B \\
1 & \text{otherwise}
\end{cases} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>\phi(A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>4.48</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>

Log-linear model:

\[ P(X = x) = \frac{1}{Z} \exp \left( \sum_i w_i f_i(x) \right) \]
Arithmetic Circuits [Darwiche, 2003]

- Directed, acyclic graph with single root
  - Leaf nodes are inputs
  - Interior nodes are addition or multiplication
  - Can represent any distribution
- Inference is linear in circuit size!
  - Never larger than junction tree
  - Can exploit local structure to save time/space
Outline

• Background
  – Graphical models
  – Arithmetic circuits

• Algorithms
  – Structure learning
  – Inference

• Implementation

• Demo

• Conclusion
Structure Learning

• Chow-Liu algorithm  
  [Chow and Liu, 1968]

• Bayesian networks with tree CPDs  
  [Chickering et al., 1997]

• Dependency networks with tree CPDs  
  [Heckerman et al., 2000]

• Arithmetic circuits (BN with tree CPDs)  
  [Lowd and Domingos, 2008]

  Can learn models with 100+ parents

  Can learn models with 100+ parents, exact inference in 100 ms.
Exact Inference

• AC Variable Elimination
  [Chavira and Darwiche, 2007]
  – Uses ADDs to compile a BN into a compact AC
  – Can exploit local structure in table CPDs

• AC exact inference
  [Darwiche, 2003]
  – Compute any conjunctive query in linear time
  – Compute all marginals in linear time
Approximate Inference

• Gibbs sampling
  – Compute marginals or arbitrary conjunctions
  – Answer queries with the same evidence in parallel
  – Rao-Blackwellization

• Loopy belief propagation
  – Both tree CPDs and table CPDs
  – For tree CPDs, running time is linear in the number of leaves

• Mean field
What’s Missing?

• Markov network and CRF support (coming soon...)
• Time series (DBNs) and relational models (PRMs, MLNs) (cf: BNT, Alchemy)
• Many advanced approximate inference methods (cf: libDAI, FastInf)
• Many alternate structure learning methods

Libra’s emphasis is on local structure and arithmetic circuits, not on reimplementing every possible algorithm.
Outline

• Background
  – Graphical models
  – Arithmetic circuits

• Algorithms
  – Structure learning
  – Inference

• Implementation

• Demo

• Conclusion
Implementation

Collection of standalone command-line executables:

– aclearnstruct
– acquery
– acopt
– acscore
– acve
– bnsample
– bnscore
– bp
– cl
– gibbs
– mf
Implementation

Shared code wrapped into libraries:

- **Ext**: Common definitions and utility methods
- **Data**: Input and output of instances
- **Lbfgs**: L-BFGS optimization (wrapped C library)
- **Expat**: XML parsing (wrapped C library)
- **Bn**: Bayesian network representation and I/O
- **Circuit**: AC representation, methods, and I/O

Everything is subject to renaming and refactoring!
Why OCaml?

OCaml is (almost) as fast as C++ and (almost) as fun as Python. This leads to faster development, refactoring, and execution.

Language Features:
- Supports functional, procedural, and object-oriented programming
- Garbage collection
- Strong, static typing (“If it compiles, it runs”)
- Automatic type inference
- Native-code compilation supports Linux, OS X, Cygwin, etc.
- Easy to link to C code
Example: Return true iff an instance \( x \) satisfies a list of conditions \( \text{condl} \), each condition being a variable equality or inequality, e.g.,
\[
(X_2 = 1) \land (X_4 = 0) \land (X_5 \neq 2)
\]

**C++ (583 characters):**

```cpp
class AtomicCondition {
public:
    AtomicCondition(bool sense, int var, int value)
        : _sense(sense), _var(var), _value(value) { }

    bool matches(const std::vector<int>& x) const {
        return (_sense == (x[_var] == _value));
    }

private:
    bool _sense;
    int _var;
    int _value;
};

bool cond_match(const std::list<const AtomicCondition*>& &condl,
        const std::vector<int>& x)
{
    std::list<const AtomicCondition*>::const_iterator i;
    for (i = condl.begin(); i != condl.end(); i++) {
        if (!(*i)->matches(x)) {
            return false;
        }
    }
    return true;
}
```

**OCaml (98 characters):**

```ocaml
let cond_match condl x =
    List.for_all (fun (sense,var,value) -> (x.(var) = value) = sense) condl
```
DEMO
Lessons Learned

• Research code requires frequent refactoring. Some languages are better at this!

• Automated tests are awesome.
Conclusion

• Libra offers fast implementations of classic and cutting-edge learning and inference algorithms

• Libra makes it easy to exploit context-specific independence for efficient representation, learning, and inference

• More functionality coming soon...

http://libra.cs.uoregon.edu
Exact Inference

Bayesian network:

\[ P(A,B,C) = P(A)P(B)P(C \mid A,B) \]

Network polynomial:

\[
\lambda_A \lambda_B \lambda_C \theta_A \theta_B \theta_C | AB + \lambda_{\neg A} \lambda_B \lambda_C \theta_{\neg A} \theta_B \theta_C | \neg AB + \\
\lambda_A \lambda_{\neg B} \lambda_C \theta_A \theta_{\neg B} \theta_C | A \neg B + \lambda_{\neg A} \lambda_{\neg B} \lambda_C \theta_{\neg A} \theta_{\neg B} \theta_C | \neg A \neg B + \\
\lambda_A \lambda_B \lambda_{\neg C} \theta_A \theta_B \theta_{\neg C} | A B + \lambda_{\neg A} \lambda_B \lambda_{\neg C} \theta_{\neg A} \theta_B \theta_{\neg C} | \neg AB + \\
\lambda_A \lambda_{\neg B} \lambda_{\neg C} \theta_A \theta_{\neg B} \theta_{\neg C} | A \neg B + \lambda_{\neg A} \lambda_{\neg B} \lambda_{\neg C} \theta_{\neg A} \theta_{\neg B} \theta_{\neg C} | \neg A \neg B
\]

Can answer arbitrary marginal queries by evaluating network polynomial.