UAI 2006

report from the
1st Evaluation of Probabilistic Inference
July 14th, 2006
What is this presentation about?

• Goal: The purpose of this evaluation is to compare the performance of a variety of different software systems on a single set of Bayesian network (BN) problems.
• By creating a friendly evaluation (as is often done in other communities such as SAT, and also in speech recognition and machine translation with their DARPA evaluations), we hope to foster new research in fast inference methods for performing a variety of queries in graphical models.
• Over the past few months, the 1st such an evaluation took place at UAI.
• This presentation summarizes the outcome of this evaluation.
Who we are

• Evaluators
  – Jeff Bilmes – University of Washington, Seattle
  – Rina Dechter – University of California, Irvine

• Graduate Student Assistance
  – Chris Bartels – University of Washington, Seattle
  – Radu Marinescu – University of CA, Irvine
  – Karim Filali – University of Washington, Seattle

• Advisory Council
  – Dan Geiger -- Technion - Israel Institute of Technology
  – Faheim Bacchus – University of Toronto
  – Kevin Murphy – University of British Columbia
Outline

• Background, goals.
• Scope (rational)
• Final chosen queries
• The UAI 2006 BN benchmark evaluation corpus
• Scoring strategies
• Participants and team members
• Results for PE and MPE
• Team presentations
  – team1 (UCLA), team2 (IET), team3 (UBC), team4 (U. Pitt/DSL), team 5 (UCI)
• Conclusion/Open discussion
Acknowledgements: Graduate Student Help

Chris Bartels, University of Washington
Radu Marinescu, University of CA, Irvine
Karim Filali, University of Washington

Also, thanks to another U. Washington Student, Mukund Narasimhan (now at MSR)
Background

- Early 2005: Rina Dechter & Dan Geiger decide there should be some form of UAI inference evaluation (like in the SAT community) and discuss the idea (by email) with Adnan Darwiche, Faheim Bacchus, Hector Geffner, Nir Friedman, Thomas Richardson.
- I (Jeff Bilmes) take on the task to run it this first time.
  - Speech recognition and DARPA evaluations
    - evaluation of ASR systems using error rate as a metric.
Scope

• Many “queries” could be evaluated including:
  – MAP – maximal a posteriori hypothesis
  – MPE – most probable explanation (also called Viterbi assignment)
  – PE – probability of evidence
  – N-best – compute the N-best of the above

• Many algorithmic variants
  – Exact inference
  – Enforced limited time-bounds and/or space bounds
  – Approximate inference, and tradeoffs between time/space/accuracy

• Classes of models
  – Static BNs with a generic description (list of CPTs)
  – More complex description language (e.g., context specific indep.)
  – Static models vs. Dynamic models (e.g., Dynamic Bayesian Networks, and DGMs) vs. relational models
Decisions for this first evaluation.

- Emphasis: Keep things simple.
- Focus on exact inference
  - exact inference can still be useful.
  - “Exact inference is NP-complete, so we perform approximate inference” is often seen in the literature
  - With smart algorithms, and for fixed (but real-world) problem sizes, exact is quite doable and can be better for applications.
- Focus on small number of queries:
- Original plan: PE, MPE, and MAP for both static and dynamic models
- From final participants list, narrowed this down to: PE and MPE on static Bayesian networks
Query: Probability of Evidence (PE)

Given a distribution $p(x_U)$, compute:

$$\ln p(\bar{x}_E) = \ln \sum_{x_H} p(\bar{x}_E, x_H)$$

Where $X_U$ is a universe of random variables, partition $U = (E, H)$, and $E$ is evidence index set, and $H = U \setminus E$ are the variables to be marginalized away.
Query: Most Probable Explanation (MPE)

Compute

\[ p(\bar{x}_E, x^*_H) = \max_{x_H} p(\bar{x}_E, x_H) \]

where

\[ x^*_H \in \text{argmax}_{x_H} p(\bar{x}_E, x_H) \]

- \( X_U \) is a universe of random variables, and \( E \) is evidence index set, and \( H = U \setminus E \) are the variables to be marginalized away.
- We accepted any such \( x_H^* \) (verification checked only that the score was correct, not the variable assignment).
The UAI06 BN Evaluation Corpus

- J=78 BNs used for PE, and J=57 BNs used for MPE queries. The BNs were not exactly the same.
- BNs were the following (more details will appear on web page):
  - random mutations of the burglar alarm graph
  - diagnosis network (Druzdzel)
  - DBNs from speech recognition that were unrolled a fixed amount.
  - Variations on the Water DBN
  - Various forms of grids
  - Variations on the ISCAS 85 electrical circuit
  - Variations on the ISCAS 89 electrical circuit
  - Various genetic linkage graphs (Geiger)
  - BNs from computer-based patient care system (cpcs)
  - Various randomly generated graphs (F. Cozman’s alg).
  - Various known-tree-width random k-trees, with determinism (k=24)
  - Various known-tree-width random positive k-trees, (k=24)
  - Various linear block coding graphs.
- While some of these have been seen before, BNs were “anonymized” before being distributed.
- BNs distributed in xbif format (basically XML)
Timing Platform and Limits

- Timing machines: dual-CPU 3.8GHz Pentium Xeons with 8Gb of RAM each, with hyper-threading turned on.
- Single threaded performance only in this evaluation.
- Each team had 4 days of dedicated machine usage to complete their timings (other than this, there was no upper time bound).
- No-one asked for more time than these 4 days -- after timing the BNs, teams could use rest of 4 days as they wish for further tuning. After final numbers were sent to me, no further adjustment of timing numbers have taken place (say based on seeing other’s results).
- Each timing number was the result of running a query 10 times, and then reporting the fastest (lowest) time.
The Teams

• Thanks to every member of every team: Each member of every team was crucial to making this a successful event!!
Team 1: UCLA

• David Allen (now at HRL Labs, CA)
• Mark Chavira (graduate student)
• Adnan Darwiche

• Keith Cascio
• Arthur Choi (graduate student)
• Jinbo Huang (now at NICTA, Australia)
Team 2: IET

From right to left in photo:
- Masami Takikawa
- Hans Dettmar
- Francis Fung
- Rick Kissh

Other team members:
- Stephen Cannon
- Chad Bisk
- Brandon Goldfedder

Other key contributors:
- Bruce D'Ambrosio
- Kathy Laskey
- Ed Wright
- Suzanne Mahoney
- Charles Twardy
- Tod Levitt
Team 3: UBC

Jacek Kisynski, University of British Columbia

David Poole, University of British Columbia

Michael Chiang, University of British Columbia
Team 4: U. Pittsburgh, DSL

Tomasz Sowinski,
University of Pittsburgh, DSL

Marek J. Druzdzel,
University of Pittsburgh, DSL
Team 5: UCI

Robert Mateescu, University of CA, Irvine

Radu Marinescu, University of CA, Irvine

Rina Dechter, University of CA, Irvine
The Results
To check for correctness, we used relative score.

$$100 \times \frac{|a - b|}{|a|} < \tau$$

where $a$ is the reference score (log probability), $b$ is a candidate score, and $\tau$ is a threshold for acceptance (set to 0.1 at UAI06).

Every team met this threshold on each graph (except when they failed, next slide or so). Most of the differences were $10^{-2}$ to $10^{-5}$. 
Definition of “FAIL”

• Each team had 4 days to complete the evaluation
• No time-limit placed on any particular BN.
• A “FAILED” score meant that either the system failed to complete the query, or that the system underflowed their own numeric precision.
  – some of the networks were designed not to fit within IEEE 64-bit double precision, so either scaling or log-arithmetic needed to be used (which is a speed hit for PE).
• Teams had the option to submit multiple separate submissions, none did.
• Systems were allowed to “backoff” from no-scaling to, say, a log-arithmetic mode (but that was included in the time charge)
Definition of Average Speedup

- Problem: We have a vector of times (in seconds) for each team for each BN, say $C_i$, goal is to compare teams with each other using various metrics. $C$ is matrix of times $C_{ij}$ is team $i$’s time on BN$_j$.
- Let $m_j = \min_i C_{ij}$ is the shortest (best) time reported for BN$_j$ out of all teams that scored on that BN, and $C_{ij}$ is the time for a team $i$ on BN$_j$.

Avg. Speedup of best over team $i = \frac{1}{J} \sum_{j=1}^{J} \frac{C_{ij}}{m_j}$

- Therefore, for a particular team, a lower number is better (best possible number is 1, meaning always equal to the min).
Results – PE and MPE

78 PE BNs, and 57 MPE BNs

1. Failure rates – number of networks that each team failed to produce a score or a correct score (including underflow)

2. Speedup results on categorized BNs
   - a BN is categorized based on how many teams failed to produce a score (so 0-5 for PE, or 0-3 for MPE)
   - Average speedup of the best performance over a particular team’s performance on each category.

3. Rank scores
   - Number of times a particular team was rank n for various n

4. Workload scores
   - In what workload region (if any) is a particular team optimal.
PE Results
PE Failure Rate Results (low is best)

Reminder: 78 BNs total for PE
Avg. Speedups: BNs that ran on all 5 systems

61 out of 78 BNs ran on all systems.

Remember: lower is better!
Avg. Speedups: BNs that ran on all systems
Avg. Speedups: BNs that ran on (3-4)/5 systems

8 out of 78 BNs ran on only 3 or 4 systems.
Avg. Speedups: BNs that ran on (3-4)/5 systems

Speedup stats

Team 1  Team 2  Team 3  Team 4  Team 5
Average  Std  Min  Max

0 10 20 30 40 50 60
9 out of 78 BNs ran on only 1 or 2 systems.

Only 2 teams (team 1 and 2) had systems that could run this category BN.

These include the genetic linkage BNs.
Rank Proportions (how often was each team a particular rank, rank 1 is best)
<table>
<thead>
<tr>
<th>BN_0</th>
<th>Alarm graph</th>
<th>Team 1</th>
<th>Team 2</th>
<th>Team 3</th>
<th>Team 4</th>
<th>Team 5</th>
<th>Team 1</th>
<th>Team 2</th>
<th>Team 3</th>
<th>Team 4</th>
<th>Team 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.476</td>
<td>1.680</td>
<td>0.400</td>
<td>0.010</td>
<td>0.020</td>
<td>0.1770</td>
<td>0.0700</td>
<td>0.1700</td>
<td>0.00065</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.548</td>
<td>2.130</td>
<td>0.520</td>
<td>0.010</td>
<td>0.010</td>
<td>0.2550</td>
<td>0.0770</td>
<td>0.2900</td>
<td>0.00133</td>
<td>0.0300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.507</td>
<td>2.160</td>
<td>0.500</td>
<td>0.010</td>
<td>0.010</td>
<td>0.1750</td>
<td>0.0680</td>
<td>0.2200</td>
<td>0.00065</td>
<td>0.0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.484</td>
<td>3.020</td>
<td>0.490</td>
<td>0.010</td>
<td>0.020</td>
<td>0.1990</td>
<td>0.0950</td>
<td>0.2100</td>
<td>0.00144</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.623</td>
<td>2.240</td>
<td>0.690</td>
<td>0.010</td>
<td>0.020</td>
<td>0.2670</td>
<td>0.1060</td>
<td>0.3600</td>
<td>0.00097</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.785</td>
<td>2.970</td>
<td>0.820</td>
<td>0.010</td>
<td>0.020</td>
<td>0.3640</td>
<td>0.1600</td>
<td>0.4100</td>
<td>0.00241</td>
<td>0.0600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.781</td>
<td>3.000</td>
<td>0.810</td>
<td>0.010</td>
<td>0.140</td>
<td>0.3640</td>
<td>0.1110</td>
<td>0.5300</td>
<td>0.00392</td>
<td>0.1300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.675</td>
<td>1.780</td>
<td>0.540</td>
<td>0.010</td>
<td>0.080</td>
<td>0.3660</td>
<td>0.0900</td>
<td>0.2800</td>
<td>0.00278</td>
<td>0.0800</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.876</td>
<td>3.200</td>
<td>0.530</td>
<td>0.010</td>
<td>0.170</td>
<td>0.5680</td>
<td>0.1000</td>
<td>0.2700</td>
<td>0.00568</td>
<td>0.1700</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.916</td>
<td>2.070</td>
<td>0.690</td>
<td>0.010</td>
<td>0.220</td>
<td>0.3740</td>
<td>0.0910</td>
<td>0.2900</td>
<td>0.00200</td>
<td>0.0600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.636</td>
<td>2.540</td>
<td>0.780</td>
<td>0.010</td>
<td>0.200</td>
<td>0.5570</td>
<td>0.1460</td>
<td>0.3700</td>
<td>0.00808</td>
<td>0.2100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.628</td>
<td>2.240</td>
<td>0.790</td>
<td>0.020</td>
<td>0.120</td>
<td>0.3010</td>
<td>0.1100</td>
<td>0.5100</td>
<td>0.00438</td>
<td>0.2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.979</td>
<td>3.580</td>
<td>1.780</td>
<td>0.100</td>
<td>1.790</td>
<td>0.2880</td>
<td>0.1170</td>
<td>0.4700</td>
<td>0.01415</td>
<td>0.1100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.908</td>
<td>3.270</td>
<td>0.920</td>
<td>0.060</td>
<td>1.270</td>
<td>0.6470</td>
<td>0.1980</td>
<td>1.4500</td>
<td>0.09400</td>
<td>1.7900</td>
</tr>
<tr>
<td></td>
<td>Diagnosis</td>
<td>1.581</td>
<td>4.430</td>
<td>1.110</td>
<td>0.070</td>
<td>0.170</td>
<td>0.4360</td>
<td>0.3230</td>
<td>0.1600</td>
<td>0.00225</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.583</td>
<td>3.590</td>
<td>1.050</td>
<td>0.070</td>
<td>0.170</td>
<td>0.4580</td>
<td>0.2960</td>
<td>0.1200</td>
<td>0.00229</td>
<td>0.0200</td>
</tr>
<tr>
<td></td>
<td>Speech</td>
<td>66.103</td>
<td>28.300</td>
<td>56.910</td>
<td>FAIL</td>
<td>467.440</td>
<td>60.0760</td>
<td>20.5390</td>
<td>46.8100</td>
<td>FAIL</td>
<td>463.2400</td>
</tr>
<tr>
<td></td>
<td>recognition</td>
<td>8.070</td>
<td>11.110</td>
<td>11.710</td>
<td>1.960</td>
<td>13.640</td>
<td>2.7210</td>
<td>5.4040</td>
<td>4.0700</td>
<td>0.72258</td>
<td>10.4400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>52.795</td>
<td>16.630</td>
<td>19.850</td>
<td>FAIL</td>
<td>123.120</td>
<td>47.5610</td>
<td>8.3500</td>
<td>11.8400</td>
<td>FAIL</td>
<td>119.2300</td>
</tr>
<tr>
<td></td>
<td>Water DBN</td>
<td>3.723</td>
<td>3.620</td>
<td>3.630</td>
<td>0.720</td>
<td>1.620</td>
<td>0.2240</td>
<td>0.0940</td>
<td>0.4100</td>
<td>0.02208</td>
<td>0.2100</td>
</tr>
<tr>
<td></td>
<td>Grids</td>
<td>2.744</td>
<td>4.630</td>
<td>FAIL</td>
<td>0.420</td>
<td>FAIL</td>
<td>1.8670</td>
<td>2.2170</td>
<td>FAIL</td>
<td>0.39248</td>
<td>FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.192</td>
<td>6.350</td>
<td>FAIL</td>
<td>0.520</td>
<td>FAIL</td>
<td>2.2500</td>
<td>3.9940</td>
<td>FAIL</td>
<td>0.47857</td>
<td>FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.158</td>
<td>30.190</td>
<td>FAIL</td>
<td>0.490</td>
<td>FAIL</td>
<td>2.2100</td>
<td>26.7250</td>
<td>FAIL</td>
<td>0.45130</td>
<td>FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.124</td>
<td>5.470</td>
<td>FAIL</td>
<td>0.100</td>
<td>FAIL</td>
<td>2.1450</td>
<td>3.4750</td>
<td>FAIL</td>
<td>0.39153</td>
<td>FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.042</td>
<td>6.850</td>
<td>FAIL</td>
<td>0.590</td>
<td>FAIL</td>
<td>2.0660</td>
<td>4.2930</td>
<td>FAIL</td>
<td>0.55135</td>
<td>FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.074</td>
<td>5.810</td>
<td>FAIL</td>
<td>0.320</td>
<td>FAIL</td>
<td>2.0720</td>
<td>3.3670</td>
<td>FAIL</td>
<td>0.28734</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

Another look at PE:
<p>| Team |
|------|-----------------------------------|
| BN_42 | icnas85 | Team 1 | Team 2 | Team 3 | Team 4 | Team 5 |
|      |         | 1.296  | 2.960  | 0.950  | 0.020  | 0.130  |
| BN_43 |         | 1.212  | 3.040  | 0.990  | 0.030  | 0.100  |
| BN_44 |         | 1.608  | 2.890  | 1.060  | 0.130  | 0.300  |
| BN_45 |         | 0.981  | 3.030  | 0.680  | 0.020  | 0.050  |
| BN_46 |         | 1.192  | 2.880  | 1.210  | 0.160  | 1.370  |
| BN_47 | icnas89 | 1.373  | 3.480  | 2.170  | 0.190  | 0.750  |
| BN_49 |         | 1.325  | 3.410  | 9.120  | 0.420  | 0.650  |
| BN_51 |         | 1.056  | 2.960  | 1.240  | 0.030  | 0.200  |
| BN_53 |         | 0.834  | 2.740  | 0.630  | 0.010  | 0.040  |
| BN_55 |         | 1.161  | 2.180  | 1.450  | 0.140  | 0.720  |
| BN_57 |         | 0.817  | 2.950  | 0.630  | 0.020  | 0.020  |
| BN_59 |         | 0.748  | 2.470  | 0.580  | 0.010  | 0.020  |
| BN_61 |         | 1.100  | 2.700  | 1.390  | 0.100  | 0.100  |
| BN_63 |         | 0.974  | 2.330  | 1.010  | 0.030  | 0.260  |
| BN_65 |         | 0.739  | 2.490  | 0.630  | 0.010  | 0.030  |
| BN_67 |         | 0.938  | 2.630  | 0.910  | 0.040  | 0.370  |
| BN_69 | icnas89 | 5.103  | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_70 |         | 8.069  | 75.300 | FAIL   | FAIL   | FAIL   |
| BN_71 |         | 8.844  | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_72 |         | 16.581 | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_73 |         | 16.634 | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_74 |         | 10.532 | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_75 |         | 25.656 | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_76 |         | 31.216 | FAIL   | FAIL   | FAIL   | FAIL   |
| BN_77 |         | 59.385 | FAIL   | FAIL   | FAIL   | FAIL   |</p>
<table>
<thead>
<tr>
<th>Team</th>
<th>WALL TIME</th>
<th>INFECTION TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BN_78</td>
<td>Team 1</td>
</tr>
<tr>
<td></td>
<td>0.409</td>
<td>0.14800</td>
</tr>
<tr>
<td></td>
<td>0.903</td>
<td>0.19200</td>
</tr>
<tr>
<td></td>
<td>0.910</td>
<td>0.25200</td>
</tr>
<tr>
<td></td>
<td>1.004</td>
<td>0.29500</td>
</tr>
<tr>
<td></td>
<td>2.397</td>
<td>0.47400</td>
</tr>
<tr>
<td></td>
<td>2.590</td>
<td>0.47000</td>
</tr>
<tr>
<td></td>
<td>2.621</td>
<td>0.49300</td>
</tr>
<tr>
<td></td>
<td>2.691</td>
<td>0.52900</td>
</tr>
<tr>
<td></td>
<td>0.754</td>
<td>0.11500</td>
</tr>
<tr>
<td></td>
<td>0.603</td>
<td>0.19200</td>
</tr>
<tr>
<td></td>
<td>0.771</td>
<td>0.19300</td>
</tr>
<tr>
<td></td>
<td>1.089</td>
<td>0.49200</td>
</tr>
<tr>
<td></td>
<td>1.748</td>
<td>0.62000</td>
</tr>
<tr>
<td></td>
<td>BN_94</td>
<td>BN_96</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>graphs</td>
</tr>
<tr>
<td></td>
<td>15.708</td>
<td>11.097</td>
</tr>
<tr>
<td></td>
<td>4.020</td>
<td>3.650</td>
</tr>
<tr>
<td></td>
<td>5.930</td>
<td>6.880</td>
</tr>
<tr>
<td></td>
<td>Team 1</td>
<td>Team 2</td>
</tr>
<tr>
<td></td>
<td>0.34800</td>
<td>0.54900</td>
</tr>
<tr>
<td></td>
<td>0.86300</td>
<td>0.87300</td>
</tr>
<tr>
<td></td>
<td>0.19000</td>
<td>8.53000</td>
</tr>
<tr>
<td></td>
<td>0.00699</td>
<td>0.38662</td>
</tr>
<tr>
<td></td>
<td>0.34000</td>
<td>0.19000</td>
</tr>
<tr>
<td></td>
<td>BN_104</td>
<td>BN_106</td>
</tr>
<tr>
<td></td>
<td>k-trees</td>
<td>(k=24)</td>
</tr>
<tr>
<td></td>
<td>5.345</td>
<td>8.592</td>
</tr>
<tr>
<td></td>
<td>1.880</td>
<td>3.140</td>
</tr>
<tr>
<td></td>
<td>6.480</td>
<td>5.790</td>
</tr>
<tr>
<td></td>
<td>Team 1</td>
<td>Team 2</td>
</tr>
<tr>
<td></td>
<td>1.04400</td>
<td>0.63700</td>
</tr>
<tr>
<td></td>
<td>0.49500</td>
<td>0.60200</td>
</tr>
<tr>
<td></td>
<td>1.96000</td>
<td>0.38000</td>
</tr>
<tr>
<td></td>
<td>1.4516</td>
<td>0.1697</td>
</tr>
<tr>
<td></td>
<td>3.50000</td>
<td>0.58000</td>
</tr>
<tr>
<td></td>
<td>1.54000</td>
<td>1.54000</td>
</tr>
</tbody>
</table>
MPE Results

• Only three teams participated:
  – Team 1
  – Team 2
  – Team 5

• 57 BNs, not the same ones, but some are variations of the same original BN.
MPE Failure Rate Results

Failure Rate (percent)

- Team 1: 0%
- Team 2: 38.6%
- Team 5: 7.01%

57 BNs total
MPE Avg. Speedups: BNs that ran on all 3 systems

31 out of 57 BNs ran on all systems.
MPE Avg. Speedups: BNs that ran on all 3 systems

31 out of 57 BNs ran on all systems.
26 out of 57 BNs ran on 2 systems.
MPE Avg. Speedups: BNs that ran on 2/3 systems
Rank Proportions (how often was each team a particular rank, rank 1 is best)
| BN_17 | 46.193 | FAIL | 0.79 | 44.988 | FAIL | 0.75 |
| BN_19 | 47.436 | FAIL | 0.72 | 46.259 | FAIL | 0.68 |
| BN_21 | 34.689 | 24.82 | u/flow | 28.182 | 17.66 | u/flow |
| BN_25 | 8.284 | 8.47 | u/flow | 3.765 | 2.951 | u/flow |
| BN_27 | 10.963 | 16.76 | u/flow | 5.688 | 8.843 | u/flow |
| BN_29 | 3.324 | 4.07 | 1.54 | 0.4 | 0.183 | 0.17 |
| BN_31 | 2.951 | 3.31 | 74.47 | 2.073 | 0.835 | 74.46 |
| BN_33 | 3.253 | 21.98 | 38.76 | 2.261 | 19.551 | 38.76 |
| BN_35 | 3.192 | 5.1 | 41.81 | 2.203 | 2.243 | 41.8 |
| BN_37 | 3.263 | 4.73 | 12.35 | 2.296 | 2.896 | 12.35 |
| BN_39 | 3.254 | 7.84 | 137.24 | 2.265 | 5.158 | 137.23 |
| BN_41 | 3.08 | 4.58 | 21.7 | 2.097 | 2.29 | 21.68 |
| BN_48 | 0.818 | FAIL | 0.78 | 0.488 | FAIL | 0.84 |
| BN_50 | 1.36 | FAIL | 0.7 | 0.488 | FAIL | 0.84 |
| BN_52 | 1.133 | FAIL | 0.75 | 0.488 | FAIL | 0.84 |
| BN_54 | 1.165 | FAIL | 0.85 | 0.488 | FAIL | 0.84 |
| BN_56 | 1.133 | FAIL | 0.75 | 0.488 | FAIL | 0.84 |
| BN_58 | 1.36 | FAIL | 0.7 | 0.488 | FAIL | 0.84 |
| BN_60 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_62 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_64 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_66 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_68 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_79 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_81 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_83 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_85 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_87 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_89 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_91 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_93 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_95 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_97 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_99 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_101 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_103 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_105 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_107 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_109 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_111 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_113 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_115 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_117 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_119 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_121 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_123 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_125 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_127 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_129 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_131 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_133 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
| BN_135 | 1.047 | FAIL | 0.43 | 0.54 | FAIL | 0.42 |
PE and MPE Results
Workload Scores: PE and MPE

- Let $\lambda$ be a workload vector, with $\lambda_j \geq 0$ and $\sum_{j=1}^{J} \lambda_j = 1$. $\lambda$ lives in the $\mathbb{R}^J$ simplex. The value $\lambda_j$ indicates proportion of times a particular user wishes to run $\text{BN}_j$.
- $C$ is matrix of timings: $C_{ij}$ is time (seconds) team $i$ ran $\text{BN}_j$.
- Then if $C\lambda = b$, $b$ vector contains costs for each team for users current workload $\lambda$. Best system for this workload is:

$$\text{Opt}(\lambda_1, \lambda_2, \ldots, \lambda_J) = \arg\max_{1 \leq i \leq J} \sum_{j=1}^{J} \lambda_j C_{ij} = \arg\max_{1 \leq i \leq J} \lambda_i \cdot C_{i}.$$ 

- So for each workload, there is an optimal system. But for each system, is there some workload (or range thereof) that makes that system optimal? Or is there no region that makes a particular team optimal?
Workload Scores and Linear Programming

- Can solve this using 5 runs of a LP solver.
- For each $i$:
  
  
  minimize $C_i \lambda$ subject to 
  
  \[
  \begin{cases} 
  C_i \lambda < C_k \lambda \text{ for all } k \neq i \\
  \sum_i \lambda_i = 1, \lambda_i \geq 0
  \end{cases}
  \]

- Repeat this for each $i \in \{1, \ldots, 5\}$, when LP solver doesn’t fail, there is at least one workload which makes $i$ optimal.
- **RESULTS:** after running PE and MPE $C$ matrices (for UAI06 evaluation), **there exists workload regions for each team where they were a clear winner** (no team was everywhere dominated by some other team)!!
- Finding the region volume itself can be done using Monte-Carlo techniques (or something more sophisticated)
Workload Scores: PE and MPE

• So each team is a winner, it depends on the workload.
• Could attempt further to rank teams based on volume of workload region where a team wins.
• Which measure, however, should we on the simplex, uniform? Why not something else.
• “A Bayesian approach to performance ranking”
  – UAI does system performance measures …
Team technical descriptions

• 5 minute for each team.
• Current plan: more details to ultimately appear on the inference evaluation web site (see main UAI page).
Team 1: UCLA Technical Description

• presented by Adnan Darwiche
Team UCLA

- **Performance summary:**
  - Solved all 78 P(e) networks in 319s: about 4s per instance
  - Solved all 57 MPE networks in 466s: about 8s per instance

- **MPE approach**
  - Prune network
  - If network has treewidth 25 or less, run RC
  - Else if network has enough local structure, run Ace
  - Else run BnB/Ace

- **P(e) approach**
  - Prune network
  - If network has genetic net characteristics, run RC_Link
  - Else if network has treewidth 25 or less, run RC
  - Else run Ace

- Approach is powerful enough to solve every network in every suite. Yet, it incurs a fixed overhead that disadvantages it on easy networks
RC and RC Link

- Recursive Conditioning
  - Conditioning/Search algorithm
  - Based on decomposing the network
  - Inference exponential in treewidth
  - VE/Jointree could have been used for this!

- RC Link
  - RC with local structure exploitation
  - Not necessarily exponential in treewidth
  - Download: http://reasoning.cs.ucla.edu/ rc_link
Ace

- Compiles BN to arithmetic circuit
- Reduce to logical inference
- **Strength:** Local Structure (determinism & CSI)
- **Strength:** online inference
- Inference not exponential in treewidth
- http://reasoning.cs.ucla.edu/ace
Branch & Bound

- Approximate network by deleting edges to provides an upper bound on MPE
- Compile the network using Ace and use to drive search
- Use belief propagation to construct
  - seed
  - a static variable order
  - for each variable, an ordering on values
<table>
<thead>
<tr>
<th>Software Title</th>
<th>Web Site</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ace</td>
<td><a href="http://reasoning.cs.ucla.edu/ace">http://reasoning.cs.ucla.edu/ace</a></td>
<td>Ace is a package that compiles a Bayesian network into an Arithmetic Circuit (AC) and then uses the AC to answer queries with respect to the network. Compilation proceeds by encoding the network into CNF, factoring the CNF, and extracting the AC from the factored logic.</td>
</tr>
<tr>
<td>c2d Compiler</td>
<td><a href="http://reasoning.cs.ucla.edu/c2d/">http://reasoning.cs.ucla.edu/c2d/</a></td>
<td>A compiler for converting CNF into d-DNNF (deterministic, decomposable negation normal form): a tractable form allowing operations such as model counting and existential quantification to be performed in linear time.</td>
</tr>
<tr>
<td>Dtree-SAT</td>
<td><a href="http://reasoning.cs.ucla.edu/dtree_sat/">http://reasoning.cs.ucla.edu/dtree_sat/</a></td>
<td>Dtree-SAT is an implementation of a variable group ordering heuristic for Propositional Satisfiability (SAT), proposed in [IJCAI03:HD], which exploits the structure of CNF formulas via their decomposition trees (dtrees). zChaff is used as the underlying SAT solver.</td>
</tr>
<tr>
<td>SamIam</td>
<td><a href="http://reasoning.cs.ucla.edu/samiam/">http://reasoning.cs.ucla.edu/samiam/</a></td>
<td>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;</td>
</tr>
</tbody>
</table>
Team 2: IET Technical Description

• presented by Masami Takikawa
Basic SPI Algorithm

- **BN**
  - Collect factors
  - Order nodes
  - Multiplication
  - Summation

**Was able to solve 59 out of 78 challenges.**

- d-separation, barren nodes removal & evidence propagation
- Minimum weight heuristic

<table>
<thead>
<tr>
<th>#BNs</th>
<th>MaxWeight</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>&lt;=100</td>
</tr>
<tr>
<td>2</td>
<td>&lt;=1,000</td>
</tr>
<tr>
<td>20</td>
<td>&lt;=10,000</td>
</tr>
<tr>
<td>16</td>
<td>&lt;=100,000</td>
</tr>
<tr>
<td>13</td>
<td>&lt;=1,000,000</td>
</tr>
<tr>
<td>4</td>
<td>&lt;=2,000,000</td>
</tr>
</tbody>
</table>

Repeat these steps until all variables are eliminated
Extensions (aka additional overhead)

- Intra-node factorization
- Collect factors
- Order nodes
- Time-slice ordering if DBN

BN

Solved additional 11 challenges.

<table>
<thead>
<tr>
<th>BN</th>
<th>Without INF</th>
<th>With INF</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN_30</td>
<td>130M (27)</td>
<td>66K (16)</td>
</tr>
<tr>
<td>BN_32</td>
<td>17G (34)</td>
<td>260K (18)</td>
</tr>
<tr>
<td>BN_34</td>
<td>1.1G (30)</td>
<td>3.1M (21)</td>
</tr>
<tr>
<td>BN_36</td>
<td>4.3G (32)</td>
<td>130K (17)</td>
</tr>
<tr>
<td>BN_38</td>
<td>4.3G (32)</td>
<td>520K (19)</td>
</tr>
<tr>
<td>BN_40</td>
<td>1.1G (30)</td>
<td>130K (17)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BN</th>
<th>Min-Weight</th>
<th>Time-slice</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN_70</td>
<td>8.3E19 (56)</td>
<td>9.0E7 (21)</td>
</tr>
<tr>
<td>BN_72</td>
<td>2.9E21 (46)</td>
<td>3.5E11 (38)</td>
</tr>
<tr>
<td>BN_73</td>
<td>6.4E22 (51)</td>
<td>4.3E9 (26)</td>
</tr>
<tr>
<td>BN_75</td>
<td>5.5E18 (43)</td>
<td>1.7E10 (34)</td>
</tr>
<tr>
<td>BN_76</td>
<td>1.9E20 (35)</td>
<td>1.7E11 (24)</td>
</tr>
</tbody>
</table>

Needed to avoid underflow for BN_20-26.

Team 3: UBC Technical Description

• presented by David Poole
Variable Elimination Code
by David Poole and Jacek Kisyński

• This is an implementation of variable elimination in Java 1.5 (without threads).

• We wanted to test how well our base VE system that we were using compared with other systems.

• We use the min-fill heuristic for the elimination orderings and 2GB of memory.
The most interesting part of the implementation is in the representation of factors:

- A factor is essentially a list of variables, and a one-dimensional array of values.

- There is a total order of all variables and a total ordering of all values, which gives a canonical order of the values in a factor.

- We can multiply factors and sum out variables without doing random access to the values, but rather using the canonical ordering to enumerate the values.
• This code was written for David Poole and Nevin Lianwen Zhang, "Exploiting contextual independence in probabilistic inference", Journal of Artificial Intelligence Research, 18, 263-313, 2003. http://www.jair.org/papers/paper1122.html

• This is also the code that is used in the CIspace belief and decision network applet. A new version of the applet will be released in July. See: http://www.cs.ubc.ca/labs/lci/CIspace/

• We plan to release the VE code as open source.
Team 4: U. Pitt/DSL Technical Description

• presented by Jeff Bilmes (Marek Druzdzel was unable to attend).
Decision Systems Laboratory
University of Pittsburgh

dsl@sis.pitt.edu
http://dsl.sis.pitt.edu/
Good theory (in addition to good implementation)

1. Clustering algorithm at the foundation of the program [Lauritzen & Spiegelhalter] (Pr(E) as the normalizing factor).
2. Relevance reasoning, based on conditional independence [Dawid 1979, Geiger 1990], as structured in [Suermondt 1992] and [Druzdzel 1992], summarized in [Druzdzel & Suermondt 1994].

Relevance steps:
1. In p(E), focusing inference on the evidence nodes
2. Removal of barren nodes
3. Evidence absorption
4. Removal of nuisance nodes
5. Reuse of valid posteriors

Full references are included in GeNIe on-line help, http://genie.sis.pitt.edu/.

Good engineering (Tomek Sowinski).
• Efficient and reliable implementation in C++ (SMILE®)
• Tested by over eight years of both academic and industrial use
Where our program spent the most time?

- bn_82 (cpcs): 38x
- bn_22 (speech dbn): 1x
- bn_94 (random): 1,046x
- bn_18 (diagnosis): ∞

Speedup due to relevance

Easy problems

Hard problems
A developer’s environment for graphical decision models (http://genie.sis.pitt.edu/).

Model developer module: **GeNIe**.
Implemented in Visual C++ in Windows environment.

Wrappers: **SMILE.NET**, **jSMILE**, Pocket SMILE
Allow SMILE to be accessed from applications other than C++ compiler

Learning and discovery module: **SMiner**

Reasoning engine: **SMILE** (Structural Modeling, Inference, and Learning Engine).
A platform independent library of C++ classes for graphical models.

Support for model building: **ImaGeNIe**

Broader context: GeNIe and SMILE

Qualitative interface: **QGeNIe**

Diagnosis: **Diagnosis**
UAI Competition: Sources of speedup

Good theory rather than engineering tricks

1. Clustering algorithm at the foundation of the program [Lauritzen & Spiegelhalter] (Pr(E) as the normalizing factor).


Full references are included in GeNIE on-line help, http://genie.sis.pitt.edu/.

Top research programmer (Tomek Sowinski).
Efficient and reliable implementation in C++ (SMILE®).
Broader Context: GeNIe and SMILE

A developer’s environment for graphical decision models ([http://genie.sis.pitt.edu/](http://genie.sis.pitt.edu/)).

Model developer module: GeNIe.
Implemented in Visual C++ in Windows environment.

Wrappers: SMILE.NET®, jSMILE®, Pocket SMILE®
Allow SMILE® to be accessed from applications other than C++ compiler

Reasoning engine: SMILE® (Structural Modeling, Inference, and Learning Engine).
A platform independent library of C++ classes for graphical models.

Support for model building: ImaGeNIe

Learning and discovery module: SMiner

Qualitative interface: QGeNIe

Diagnosis: Diagnosis

ImaGeNIe

DSL
Decision Systems Laboratory

UAI Software Competition
Team 5: UCI Technical Description

• presented by Rina Dechter
PE & MPE – AND/OR Search

Bayesian network

Pseudo tree

AND/OR search tree

Context minimal graph
Adaptive caching

context(X) = \([X_1X_2\ldots X_{k-i}X_{k-i+1}\ldots X_k]\)

i-bound < k

i-cache for X is purged for every new instantiation of X_{k-i}

i-context(X) = \([X_{k-i+1}\ldots X_k]\)

in conditioned subproblem
PE solver - implementation

• C++ implementation
• Caching based on context (table caching)
  – Adaptive caching when contexts are too large
• Switch to variable elimination for small and nondeterministic problems
• Constraint propagation
• No good learning – just caching no goods
• Dynamic range support (for very small probabilities)
MPE solver - AOMB(i,j)

Node value v(n): most probable explanation of the sub-problem rooted by n

Caching: identical sub-problems rooted at AND nodes (identified by their contexts) are solved once and the results cached

j-bound (context size) controls the memory used for caching

Heuristics: pruning is based on heuristics estimates which are pre-computed by bounded inference (i.e. mini-bucket approximation)

i-bound (mini-bucket size) controls the accuracy of the heuristic

No constraint propagation
AOMB(i,j) – Mini-Bucket Heuristics

• Each node \( n \) has a static heuristic estimate \( h(n) \) of \( v(n) \)
  – \( h(n) \) is an upper bound on the value \( v(n) \)
  – \( h(h) \) is computed based on the augmented bucket structure generated by the mini-bucket approximation \( \text{MBE}(i) \)

• For every node \( n \) in the AND/OR search graph:
  – \( \text{lb}(n) \) – current best solution cost rooted at \( n \)
  – \( \text{ub}(n) \) – upper bound on the most probable explanation at \( n \)
  – Prune the search space below current node \( t \) if \( \text{ub}(m) < \text{lb}(m) \), where \( m \) is an ancestor of \( t \) along the current path from the root

• During search, merge nodes based on context (caching); maintain cache tables of size \( O(\exp(j)) \), where \( j \) is a bound on the size of the context.
AOMB(i,j) – Implementation

- C++ implementation
- B&B procedure is recursive
  - Could be a bit faster if we simulate the stack
- Cache tables implemented as hash tables
- No ASM code or other optimizations
- Static variable ordering determined by the min-fill ordering (minimizes the context size)
- Choosing (i,j) parameters:
  - i-bound: choose $i$ such that the augmented bucket structure generated by MBE($i$) fits 2GB of RAM ($i < 22$)
  - j-bound: $j = i + 0.75\times i$ ($j < 30$)
- No Constraint propagation
References


Conclusions
Conclusions and Discussion

- Most teams said they had fun and it was a learning experience – people also became somewhat competitive 😊
- Teams that used C++ (teams 4-5) arguably had faster times than those who used Java (teams 1-3).
- Use harder BNs and or harder queries next year
  - hard to find real-world BNs that are easily available but that are hard. If you have a BN that is hard, please make it available for next year.
  - Regardless of who runs it next year, please send candidate networks directly to me for now <bilmes@ee.washington.edu>
- Provide better resolution and standardized/unified timer (use IPM package).
- Have a dynamic category (needs to be more interest).
- Have an approximate inference category, look at time/space/accuracy tradeoffs