

# UAI06 Inference Evaluation Results

Jeff Bilmes

University of Washington, Seattle  
Department of Electrical Engineering

\$Id: results.tex,v 1.4 2006/08/09 08:22:57 bilmes Exp bilmes

August 14, 2006

## Abstract

This brief report describes the results of the UAI 2006 inference evaluation. The web page with more information and the complete Bayesian network listing is at <http://ssli.ee.washington.edu/~bilmes/uai06InferenceEvaluation/results>.

## 1 Introduction and Overview

Over the years, many different algorithms for making queries of probability distributions represented by graphical models have been developed. There are now a number of algorithms for exact inference (e.g., junction-tree based schemes using graph triangulations, search or conditioning based approaches based on SAT/CSP techniques), and approximation methods (including variational, generalized belief propagation, and sampling schemes). Also, aspects of software implementation, such as relevant data structures, memory management, compiler use, and other subsidiary algorithms can have a significant effect on actual wall-clock runtime.

The purpose of this evaluation was to compare the performance of a variety of different software systems on a single set of Bayesian network (BN) problems. A number of different Bayesian networks were made available to each participant, and each will be evaluated according to the following criteria:

- Exact inference only. While approximate inference is crucial in many problem domains, recent results have shown that exact inference can still be performed on a variety of interesting (and thought previously to be intractable) problems. Results from the SAT/CSP community have shown that it is possible to push the envelope of exact algorithms far beyond what was previously believed to be possible, even considering the enormous increase in available compute power over the past 20 years. Therefore, this first UAI competition involved systems that perform exact inference only (meaning everyone's probability score should be numerically identical).
- Discrete state space. All observed and hidden variables were discrete, and had varying cardinalities.
- Evaluation involved both wall-clock run time, and amount of memory used. Both fastest time with a memory ceiling, and smallest memory with a time limit will be evaluated and reported.
- Several forms of query were evaluated, including Probability(evidence) (i.e., summing out hidden variables), MPE. MAP was also considered, but there were not enough interested participants this year for this. The resulting scores/assignments for all queries needed to be correct and accurate (see below).
- A set of complete Bayesian network models were made available to participants. Some of the BNs were strictly positive (meaning there were not zeros in the CPTs). Other BNs were mixtures of dense, sparse, and deterministic CPTs.
- We evaluated only single threaded performance (i.e., no multi-threading or parallelism in this first evaluation). We used the same underlying (Intel-class) microprocessor running the Linux operating system for all systems. Each process had unlimited memory available to them other than the fact that the machine had its own limits (see below).

- One type of BNs was evaluated, namely static BNs. It was hoped that dynamic models (such as dynamic Bayesian networks) were also of interest, but there were not enough interested participants. The BNs were placed in a standard easy-to-read format.
- Each participant (or any interested party) has the option to submit BNs of their own to be considered for evaluation and most of the ones that were submitted were selected. All the resulting BNs used in the competition, and all results, are listed on the UAI inference evaluation web site (see <http://ssli.ee.washington.edu/~bilmes/uai06InferenceEvaluation/results>).

More details regarding how the competition was ran are included below. Detailed results and numbers are contained at <http://ssli.ee.washington.edu/~bilmes/uai06InferenceEvaluation/results>, including the actual timing numbers for each team, the BNs themselves, various scripts, the powerpoint presentation given at UAI06, etc.

## 2 Teams

We had 5 teams participating this year. The main contacts for teams included:

- Team 1: UCLA: Mark Chavira <[chavira@cs.ucla.edu](mailto:chavira@cs.ucla.edu)> Arthur Choi <[aychoi@cs.ucla.edu](mailto:aychoi@cs.ucla.edu)> David Allen <[david.allen.us@gmail.com](mailto:david.allen.us@gmail.com)> Adnan Darwiche <[darwiche@cs.ucla.edu](mailto:darwiche@cs.ucla.edu)>
- Team 2: IET: Masami Takikawa <[takikawa@iet.com](mailto:takikawa@iet.com)> Chad K. Bisk <[ckbisk@iet.com](mailto:ckbisk@iet.com)> Francis Fung <[fung@iet.com](mailto:fung@iet.com)>
- Team 3: UBC: Jacek Kisynski, Michael Chiang, David Poole contact: “Jacek Kisynski” <[kisynski@cs.ubc.ca](mailto:kisynski@cs.ubc.ca)>
- Team 4: University of Pittsburgh, Decision Systems Laboratory: Marek J. Druzdzal <[marek@sis.pitt.edu](mailto:marek@sis.pitt.edu)> Tomasz Sowinski <[shooltz@shooltz.com](mailto:shooltz@shooltz.com)>
- Team 5: UCI: Robert Mateescu <[mateescu@ics.uci.edu](mailto:mateescu@ics.uci.edu)> Radu Marinescu <[radum@ics.uci.edu](mailto:radum@ics.uci.edu)> Rina Dechter <[dechter@ics.uci.edu](mailto:dechter@ics.uci.edu)>

Each team also listed a few other participants and/or collaborators/contributors. See the powerpoint presentation on the web page for their names.

Significant acknowledgment should be given to the graduate student assistance this time around, and that included Chris Bartels and Karim Filali at the University of Washington, Radu Marinescu at University of California Irvine, and some last minute help from Mukund Narasimhan University of Washington. The organizers consisted of Jeff Bilmes (University of Washington) and Rina Dechter (UCI). While both Radu Marinescu and Rina Dechter both were organizers and participants, as the main organizer I (J. Bilmes) made sure that there was no conflict of interest (e.g., inherent bias towards networks that any particular team had a particularly large amount of experience with over the other teams).

## 3 Computing Environment

A set of three dedicated computers were reserved for each of the teams to use. This included a login and test machine, where no timing was done but the computing environment, file system, available tools/compilers, etc. were identical to the timing machines. Anyone at any time was free to log in and test their programs and algorithms anytime they wanted.

Two dedicated timing machines were made available for a period of about 6 weeks total. These machines were used to produce the timing numbers. The machines are each dual-CPU 3.8GHz Pentium Xeons with 8Gb of RAM each, with hyper-threading turned on. The machines were running the Linux operating system. Each team was instructed not to log into these machines except during designated time slots so as to not distort the timing numbers for the other teams. Other users were instructed not to log in to these machines during the evaluation.

Each team had exclusive use of one of the timing machines for a period of 4 days total. We had assumed that 4 days should have been plenty of time to get numbers on all the BNs on both types of queries we evaluated (and we were correct).

The evaluation took place before UAI 2006, during the period of June 19th through July 8th, 2006. Some teams requested a non-contiguous set of 4 days to use, which was granted.

## 4 Queries

We ended up deciding to evaluate using only two forms of query, namely what we called PE and MPE. There was not sufficient interest in MAP queries this time to warrant inclusion in the evaluation (but hopefully next year). Moreover, as mentioned above, dynamic graphical models (such as DBNs or CRFs) were not evaluated due to insufficient numbers of teams interested in these models.

We review the computation of both PE and MPE scores here.

### 4.1 PE

We have a universe of variables with index set  $U$ , so the random variables are  $X_U$  and can take values  $x_U$ . For each PE query, we have a bi-partition of  $U$  into  $U = E \cup H$  where  $E$  is the index set of variables that have been observed to hold values  $\bar{x}_E$ , and the remaining variables  $X_H$  are those variables that must be marginalized over. Thus, we have that the PE score is:

$$\text{PE-score} = p(\bar{x}_E) = \sum_{x_H} p(\bar{x}_E, x_H)$$

Thus, a PE query is just intelligently summing out the hidden variables, something that can be done with the elimination algorithm, search, hybrid schemes, junction trees, etc.

### 4.2 MPE

We have a universe of variables with index set  $U$ , so the random variables are  $X_U$  and can take values  $x_U$ . For each MPE query, we have a bi-partition of  $U$  into  $U = E \cup H$  where  $E$  is the index set of variables that have been observed to hold values  $\bar{x}_E$ , and the remaining variables  $X_H$  are those variables that must be maximized over. Thus, we have that the MPE score is:

$$\text{MPE-score} = p(\bar{x}_E, x_H^*) = \max_{x_H} p(\bar{x}_E, x_H)$$

where

$$x_H^* = \text{argmax}_{x_H} p(\bar{x}_E, x_H)$$

### 4.3 A discussion of the queries

A wide variety of queries were originally considered to be evaluated this year. This included PE, MPE, MAP, and others included full collect-distribute evidence junction tree posteriors  $p(x_h|\bar{x}_E)$  for each hidden variable  $x_h$ , clique-based posteriors  $p(x_c|\bar{x}_E)$  for clique  $c$ , various modifications to MAP, the  $K$  most probable assignments, and so on. Many forms of approximate inference queries were also considered, where we would have evaluated both time, memory, and accuracy of the result of the query (using some loss function). It was decided, however, to keep this first evaluation relatively simple while still using queries for which there was some interest.

Regarding rationale for these queries, there are a number of instances where PE is regularly used. First, in computing  $p(C|E)$ , one needs  $P(C, E)/P(E)$ , where  $C$  ranges over a set of variables that do not factorize. This is used whenever one wants to perform parameter learning with hidden variables, say using EM or gradient methods, and where the values  $x_E$  can change from query to query. Some applications use this form of query many hundreds of thousands of times for each application run (so getting it fast is crucial). Also, just to compute the numerator  $P(C, E)$ , we can think of this as  $E' = C \cup E$  and compute  $P(E')$  which is a form of  $P(E)$  query for a set of values.

Also,  $P(E)$  is almost identical to computing the log conditional partition function. In a conditional random field (CRF), where we have a tri-partition of  $U = F \cup H \cup E$ , where  $F$  are the variables we want the posterior on,  $H$  are hidden, and  $E$  is evidence, we are constantly computing queries of the form:

$$p(F|E) = \frac{1}{Z(E)} \sum_H \prod_C \psi_C(C)$$

where  $C$  ranges over cliques, and where

$$Z(E) = \sum_{F,H} \prod_C \psi_C(C)$$

In other words, the conditional partition function  $Z(E)$  is essentially computationally  $P(E)$  (it can be computed using the same algorithms, except that in the CRF case, the factors are not locally normalized). CRFs are now widely used in image processing, speech, language, and general pattern recognition, and whenever they are trained, we are constantly needing to compute exact or approximate values for the conditional partition function.

Also, MPE queries are used ubiquitously in coding networks, and in pattern recognition, and classification using BNs. For example, Viterbi decoding is used in speech, language, handwriting recognition, and image processing, where it is often the case that computing MAP queries is intractable, but computing MPE is not. While MPE suffers from the problems that Pearl mentions many years ago, and which is also a widely known problem in speech recognition with Viterbi decoding, (essentially that if  $F$  are the variables we are interested in, and we compute the joint max assignment to  $X_F$  and  $X_H$ , the resulting  $X_F$  may be very different than the maximum of  $p(X_F|\bar{X}_E)$ ), and while this doesn't exist in MAP based queries, MPE is still widely used. Also, there are ways to quickly convert a BN in to a Markov random field (essentially by exponentiating certain probabilities) which can significantly mitigate these problems. Having fast MPE queries implemented makes it such that we can compute the answer to desired pattern recognition questions.

## 5 The Networks

The networks we chose to evaluate were taken from a variety of sources, and were mostly a real-world BNs that had been changed (in particular the query), and then "anonymized" to reduce the chance that a given team would recognize the network (but of course the anonymization was easy to break, and we did not control for this in any way). The networks were numbered from  $BN_i$  where  $i$  is an integer. Some of the networks were used for PE and others were used for MPE. Some of the networks for MPE and PE might have come from the same source, but were given different numbers for the PE and MPE queries (so the networks for PE and MPE did not overlap). All networks were specified in XBIF format. Here is a brief summary of the networks we used. More details on the networks, the networks themselves, XBIF format, and the partition into PE and MPE networks are given on the web site <http://ssli.ee.washington.edu/~bilmes/uai06InferenceEvaluation/results>.

- BNs 0-15 were random modifications of the Alarm graph
- BNs 16-19 were modified forms of the Diagnosis graph
- BNs 20-27 were DBNs from speech recognition that were unrolled a fixed amount (thus the v
- BNs 28-29 were the Water DBN
- BNs 30-41 were grids
- BNs 42-46 were from iscas85
- BNs 47-68 were from iscas89
- BNs 69-77 were genetic linkage graphs
- BNs 78-93 were from cpcs
- BNs 94-103 were randomly generated graphs
- BNs 104-113 were known tree-width random k-trees, with determinism (k=24)
- BNs 114-125 were known tree-width random positive k-trees, (k=24)
- BNs 126-134 were coding graphs.

## 6 Instructions Given to Teams

For each of the PE and MPE queries needing to be evaluated, we produced a list of BNs all in `xbif` format. The files each have a designated set of variables that are observed (i.e., the index set  $E$ ) and the files also have a given set values assigned to those observed variables (i.e., the values  $\bar{x}_E$ ). Each team's task was to:

1. read in those files,
2. pre-process them to form whatever data-structures so desired
3. compute the appropriate score on that BN.

Each team needed to produce two ASCII files, one for PE (78 lines long, since there were 78 PE networks) and one for MPE (57 lines long). Each file should be a list of results for each file line, where each line of the file contains:

1. the BN name
2. the appropriate score (in log base e)
3. two running time scores
  - (a) the wall-clock time (in seconds) it takes to compute P(E). This can be obtained using the unix 'time' command.
  - (b) all but the read-in time. I.e., this should include the time to pre-process the BNs, and perform any actual inference/search for P(E). In seconds.

All times were reported in seconds, and all scores reported log base e (natural logarithm). For example, one of the team's files should have looked like:

```
+-----+
| BN_1  -3.4343  34.341  20.343
| BN_2  -33.4343 19.343  10.205
| ...
```

Each team was free to produce results for multiple systems. I.e., say a team has 2 entirely different algorithms for MPE and three algorithms for PE, then the team would produce 2 separate MPE results files, and three separate results files for PE. If a team's systems are entirely different, then indeed this route is probably more appropriate.

It was not appropriate for a team to use multiple systems and report the best time out of all of them. If a team did have multiple algorithms, then a meta-algorithm itself needed to figure out what the best BN algorithm was, but the seconds reported at the end should have included the time needed to figure out which algorithm to use.

If a system failed on a given graph (due to running out of memory, numerical underflow, or taking an exorbitant amount of time), the team was instructed to include the following line for that BN:

```
| BN_k  FAIL  FAIL  FAIL
```

In addition to the results file for each system submitted, each team was instructed to submit a script that runs their results on a given graph. I.e., given 2 MPE systems (A and B) and 1 PE system (A), then the script should be invocable using the following form:

```
team_3_mpe_A BN_3
```

meaning, this will run team 3's MPE A system on BN<sub>3</sub> and produce an answer. These scripts were used to verify that the evaluators get the same timing numbers and PE/MPE scores.

Lastly, in addition to the results file and the script above, each system submitted should also include a README file that describes some of the technical details of the system – i.e., what form of computation does a team's system perform (elimination, search, hybrid, what heuristics were used, etc.). Each team was instructed so that there should be enough detail so that it is clear why each system runs particularly quickly (or slowly) on a particular graph. These descriptions (and a few extra bits of email) are included on the web page.

As mentioned above, there were a total of 57 MPE BNs, and 78 P(E) BNs, for a total of 135 BNs.

There are a variety of scripts that we have collected for converting between some common BN file formats, and which might be useful to the community and they are located on the web page under the `scripts` subdirectory.

## 7 Timing and Scoring Strategy

All of the probability scores checked out using a relative difference formula  $100 * |a - b|/|a| < \tau$ , where  $a$  is our score, and  $b$  is a team's score on a particular BN. Everyone met the threshold ( $\tau = 0.1$ ) for all graphs on PE and MPE except for one graph by team 2 (whose relative difference was 1.214 on BN<sub>70</sub>) but I thought that was not different enough to warrant failure on that BN. Most of the other relative differences were more on the order of 1e-2 to 1e-5 or so. Note that we got the  $a$  scores from a variety of systems for which we were certain of the scores.

The timings were tallied up in a number of different ways, including:

1. How many times and percentage of time each team "FAILED" (i.e., either reported FAILED, or reported a score if -infinity). I consider an underflow a failure, as some teams took big speed hits to make sure that they didn't underflow.
2. For the BNs that no-one failed on, I computed their average speedup of the best time over a teams time for a particular BN. I.e., suppose that  $m_j$  is the shortest (best) time reported for BN <sub>$j$</sub>  out of all teams, and that  $s_j$  is the time for a particular team (say team A), then the average speedup over team A was:

$$\text{avg-speedup-A} = (1/N) \sum_{i=1}^N s_j/m_j$$

Said again, an average speedup of the best time over team A's time (please read this last sentence again). Speedup is a notion that is widely used in, for example, the high-performance literature (see for example Hennessey and Patterson's text). I also computed and report the min, max, and standard deviation of the speedup as well.

3. Next, some teams failed on some BNs that others did not. In this case, let  $m_j$  be the min time that any team got on BN <sub>$j$</sub>  (note that there was no BN that everyone failed on, as team 1 didn't fail on any of them). I computed average speedup for each team on the BNs that it did not fail on, using the same formula above (again computing average, standard deviation, min, and max).
4. Ranks. Basically, how many times was each team rank 1 (had the fastest system on a BN), rank 2 (2nd fastest), rank 3 (third fastest), etc. Note that there could be ties (i.e., two teams could both have achieved the fastest time, or two teams could both have failed in which case they tied for rank 5). Due to ties, the ranks may not add up to the total number of BNs.
5. Linear program. Given a set of timings for each BN, the winner really depends on the workload  $\lambda$  which is a vector that gives the probability that a particular BN will be run (or how important a BN is). Using a simple linear program, it was possible to show that each team had a workload where they were the clear winner, or in other words, no team was everywhere dominated by the other teams. The timing matrix (matrix with each teams times) and the matlab linear programming script is on the web page.

All of the above were computed for both the "wall clock time" (as reported by the unix time command) and the "inference time" (which programs should be reporting internally). The timing for each BN was run 10 times, and the final reported time for a BN was taken minimum time over those 10. This was done to reduce timing variance.

Note that the unix 'time' command has a resolution of only 1/100th of a second and a few of the timings were being reported as 0. This lead to divide by zero problems in the above formula, so I rounded those numbers up to 0.01 (the shortest measurable time for unix time) for those numbers. This happened very seldomly though.

## 8 Evaluation of Results: Summary

Detailed results are on the web, as here is only a summary. We do say here that teams 1, 2, and 3 wrote code in JAVA, and teams 4 and 5 coded in C++. Team 4 was used the Intel optimizing C++ compiler, team 5 used GNU gcc.

For the PE results, all teams participated. Team 1 was the only team who was able to compute scores for **all** of the BNs. Some of the BNs were such that the PE/MPE scores fell below the IEEE 64-bit dynamic range, and it was necessary to do either scaling or log-based arithmetic to get scores for those. These networks were mostly DBNs.

In terms of having the overall fastest times on the most number of BNs, team 4 did quite well, and this was both based on their reported inference time and on “wall clock” time reported by the Unix ‘time’ command. As seen in the results files (see web page), when team 4 did not fail, they were quite a bit faster on many of the graphs than any of the other teams. The other teams did well also. Team 5 had a very nice tradeoff of fast timings and failure rates. Teams 2 and 3 also had very nice trade-offs. In all, I think all teams were winners.

For the MPE results, 3 teams participated. Team 1 again had a 0 failure rate (they succeeded on scoring all graphs). Team 5 had a nice tradeoff of a low failure rate, but fast timing numbers, and team 2 was in the middle.

On the web page are 4 files, for each of mpe,pe x inf, wall where mpe,pe is obvious, and inf/wall is the scoring for reported inference time or reported wall clock time. All the results are also contained in the results directory on the web. All results are in:

```
team_i_{pe,mpe}_results.txt
```

The baseline probability scores are in {pe,mpe}\_results.txt. There is a script that checks that the scores are correct, to run it, say with team 1, do:

```
check_rel_diff.pl pe_results.txt team_1_pe_results.txt
```

There is another script to tally the scores (which generated the attached files). To run it, (in tsh) do:

```
./compute_times.pl -inf team*_pe*.txt
```

for reported inference time, or

```
./compute_times.pl -wall team*_pe*.txt
```

for reported ‘wall clock’ time.

To get scores on mpe, do:

```
./compute_times.pl -inf team*_mpe*.txt
```

Note that there are a few commented out lines in the above script that will need to be changed before running it for PE or MPE, in the script search for the string PEOPTION to find the comments.

The outputs of the script are also this directory as {pe,mpe}\_scores\_{inf,wall}.txt.

## 9 Recommendations to future UAI evaluations

- Make sure the person running the evaluation is independent and unbiased, and has no favored team.
- Make sure the person running the evaluation is someone completely immune to persuasion by any of the team participants.
- This year, the BNs and/or the queries that were evaluated with the BNs were too simple (i.e., the running times were too short). It is useful to evaluate such queries, as there are applications that depend on BN query evaluations occurring many times during an applications run. In this case, the timing should be on running each such BN many times, not just a few times. This will improve the timing accuracy.

Also, there is interest in solving very hard problems, ones that current computing systems are not able to solve exactly. Therefore, more difficult BNs should be evaluated. While we did attempt to do this, we found it difficult under the queries we chose to find BNs that were both extremely difficult to evaluate and that we ourselves were capable of evaluating in a reasonable amount of time. It is recommended that next years evaluators look at the SAT/CSP community to see what is done in this case.

- Use a high-quality and controlled wall-clock timer, and give the timer to each participant beforehand, and make sure each team uses exactly the same timer. A good one is “IPM: Interval Performance Monitoring” by K. Asanovic <http://www.cag.lcs.mit.edu/~krste/ipm/IPM.html>. Something also needs to be done when teams are using different languages (i.e., IPM works with C and C++ but a java interface needs to be done).
- More diverse forms of graphical model and query type. Evaluate Dynamic models (DBNs), and other forms of query mentioned above. Also, evaluate approximate inference and investigate time/space/accuracy tradeoffs.