6.825A

6.825 Project 2: Inference in Bayesian Networks

Eric Mumpower
Dan RK Ports

November 8, 2005

plots - average?
how split work?

108/120

1 Overview

This project explores the use of a variety of Bayesian network inference algorithms. We implemented a couple of heuristic variants of an exact algorithm (variable elimination) and two different approximate inference algorithms (likelihood weighting and Gibbs sampling). These implementations were used to compare the accuracy and performance of these algorithms under a variety of conditions.

In the case of the variable elimination algorithm, the heuristics we implemented were randomized elimination ordering and greedy elimination ordering. A number of trials were run performing the suggested queries with each of these variants; the relative experimental performance of these heuristics are explored in Sections 2.2 and 2.3.

The approximate inference algorithms were run in a series of experiments using a range of different operational parameters, comparing their accuracy to results obtained using exact methods. This allowed us to determine what level of accuracy can be obtained from the algorithms given a particular number of samples. We compare the effectiveness of likelihood weighting and Gibbs sampling on four queries in two different networks, and examine how the structure of the network and query affects the appropriate choice of algorithm. We also compare the runtime of the sampling algorithms to that of finding the exact solution with variable elimination, to understand what level of quality can be obtained from sampling algorithms in less time than finding the exact solution.

Our experimental results and analysis are presented as follows:

Section 2 discusses our work with variable elimination, and the initial data we obtained (§2.2). There, we explore the subtle differences between some query results (§2.3.1), analyze the performance of random elimination-ordering (§2.3.2), and compare random elimination with a greedy elimination-ordering heuristic (§2.3.3).

In Section 3, we evaluate the likelihood weighting and Gibbs sampling solvers. We first experiment with the number of samples that should be discarded during the burn-in period of Gibbs sampling (§3.1), and then use this to compare the result quality and runtime of likelihood weighting and Gibbs sampling for different numbers of samples (§3.2).

Finally, Section 4 gives a summary of our work on this project.

2 Variable Elimination

Task 1 10/10

2.1 Experiment

We implemented the Variable Elimination algorithm as described by Koller and Friedman [2005, unpublished].

It may be noted that some VE queries can be significantly optimized by eliminating irrelevant nodes from the graph prior to performing Variable Elimination. In particular, any node which is d-separated from the query (given the evidence) may be removed from the network. As has been pointed out by Russell and Norvig, any node which is not an ancestor of the query or evidence nodes will be d-separated from the query; our implementation performs the optimization of removing all such nodes from the net before performing a query.

$P(Burglary \mid JohnCalls = true, MaryCalls = true)$	
true	0.2841718353643929
false	0.7158281646356071

$P(Earthquake \mid JohnCalls = true, Burglary = true)$	
true	0.0020199831098788364
false	0.9979800168901212

$P(PropCost \mid Age = Adolescent, Airbag = False, Mileage = TwentyThou)$	
Thousand	0.49761258318989815
TenThou	0.34597846229050366
HundredThou	0.13571003396327844
Million	0.020698920556319802

Table 1: Results of Variable Elimination Queries: Task 1

$P(N104 \mid N41 =$ "2", $N84 =$ "1", $N116 =$ "0")	
"0"	0.9429999988500375
"1"	0.05700000114996252

$P(N73 \mid N152 =$ "1", $N116 =$ "0", $N43 =$ "1")	
0	0.9979999999565901
1	0.002000000043409999

Table 2: Results of Variable Elimination Queries: Task 2(b)

The initial data demonstrating the operation of our VE implementation may be found in Section 2.2.

2.2 Data: Variable Elimination

Our VE implementation obtains the results given in tables 1 and 2 for the queries from tasks 1 and 2(b). (The results from task 2(a) will be discussed in section 2.3.1.) The *Burglary*-network results precisely match the results returned by the Enumeration solver with these queries; the *Insurance* query matches the reference distribution given in the project handout.

2.3 Analysis: Variable Elimination

20/20

2.3.1 Basic Queries

Results from the suggested *Insurance*-network queries can be found in table 3.

One might imagine that in an insurance claim, knowing the car was a luxury vehicle might shift the probable property costs uniformly higher (due to effects on the probable outcomes of CarValue, ThisCarDam, and Accident). However, the network reports that this is not the case — the property costs merely shift away from the low-to-mid-range (tens of thousands of dollars) into the lower and

P(PropCost | *Age = Adolescent, Airbag = False, Mileage = TwentyThou, MakeModel = Luxury)*	
Thousand	0.5220516748791952
TenThou	0.26169185473341977
HundredThou	0.1839241319496738
Million	0.03233233843771127
P(PropCost | *Age = Adolescent, Airbag = False, Mileage = TwentyThou, GoodStudent = True)*	
Thousand	0.5004883507298429
TenThou	0.33851209276039584
HundredThou	0.1394412996694684
Million	0.021558256840292778

Table 3: Results of Variable Elimination Queries: Task 2(a)

higher cost ranges. In fact, it makes the likelihood of millions of dollars in property damage grow by a factor of 1.5. Given these results, one might wonder if young drivers are less likely to be moderately-reckless when driving an especially expensive car: either they're fairly careful, or completely throw caution to the wind.

From an inspection of the *Insurance* graph, one might imagine that knowing the driver is a good student might not have a great deal of effect on the probable property costs. In particular, the GoodStudent node has no descendants and only two ancestors; the value of one of GoodStudent's ancestors (Age) is known, and is the sole parent of GoodStudent's other ancestor (Age → SocioEcon → GoodStudent)). The network supports this insight, as the query results change only very slightly with the addition of GoodStudent to the query evidence.

2.3.2 Random Elimination Orderings

20/20

We ran the Variable Elimination solver with a random-elimination-order mechanism against the four queries from task 2; histograms of the solver runtimes are plotted in Figures 1-4. (**Note**: due to the trivial and largely-uniform nature of the *Carpo* queries, they are plotted against a *linear* timescale, while the other queries are plotted against a *logarithmic* timescale.)

As can clearly be seen in Figures 3-4, the *Carpo* queries are rather trivial to solve: in over 450 trials per query, the solver never took more than 22 ms to answer either query. This triviality is curious — the other VE queries tend to be measured in *seconds*. However, an examination of the relevant nodes on the *Carpo* graph makes it clear what is happening — all the evidence variables are d-separated from their queries, one query variable is a prior, and the other query has only one ancestor. Thus, the non-ancestor pruning optimization mentioned in Section 2.1 causes both the suggested *Carpo* queries to reduce to a highly trimmed graph. In the $N104$ query, the largest possible CPT is of dimension 5, and in the $N73$ query, the largest CPT which can be generated is of dimension 1 (!). Thus, we see that d-separation (and network structure) can have a tremendous effect on query complexity.

good

The *Insurance* network queries are much more subtly structured.[1] To begin with, our 480 trials (per query) achieved only a rough sample of the possible elimination orderings: there were 16-17 nodes left for elimination after graph-pruning and evidence-assertion, yielding over $16! \approx 10^{12}$ possible elimination orderings. Also, the inclusion of the one extra node in the GoodStudent query increases the mean runtime by a factor of 4 (from around 15 seconds to around 60 seconds) relative to the simpler query. Finally, these histograms have a number of local clumpings of timings which hint at some patterns which may relate to the sizes of computed CPTs.

2.3.3 Greedy Elimination Ordering

10/10

When querying the *Insurance* network with the VE solver, using the greedy elimination-ordering mechanism was noticeably faster[2] than the fastest times achieved by the random elimination-ordering mechanism. Given the vast size of the optimal-elimination-ordering search space, it should come as little surprise that random orderings were unlikely to perform as well as an ordering which employed some kind of selection heuristic.

With the much smaller elimination-ordering space of the *Carpo* queries, however, the random orderer often matched the speed of the greedy heuristic. In the greatly reduced universe of these queries, there is little reward for the sort of planning represented by the greedy heuristic.

3 Sampling

We implemented both likelihood weighting and Gibbs sampling techniques. To evaluate their effectiveness, we first investigated the burn-in time for Gibbs sampling: the number of initial samples that must be discarded before reaching the stationary distribution. We then used this information when comparing the quality of likelihood weighting and Gibbs sampling to the exact solution, for various numbers of samples.

3.1 Gibbs Sampling Burn-In Time

10/10

The Gibbs sampling algorithm begins with a random sample that ignores the evidence, and so requires some time for the Markov chain to converge to its stationary distribution. Hence, the earliest samples generated are not likely to be accurate, and we can expect that discarding them will improve the overall accuracy of the algorithm.

good!

To test this hypothesis, we experimented with the fraction of samples that would be discarded. We ran experiments where 10,000 samples were taken, and an initial prefix was discarded. Note that this means the number of samples actually used decreases as the prefix length increases. We tested the Gibbs sampler with varying prefix lengths on queries in the Burglary (Figure 5), Insurance (Figure 6), and Carpo (Figure 7) networks, and plotted the Kullback-Leibler divergence relative

[1] Note, the rightmost (slowest) bin in each of Figures 1 and 2 represent trials which failed to terminate due to exhausting the heap memory available to Java.

[2] The MakeModel query took 150 *ms* (versus a best time of 284 *ms* with random ordering); the GoodStudent query took about 310 *ms* (versus a best time of 760 *ms* with random ordering).

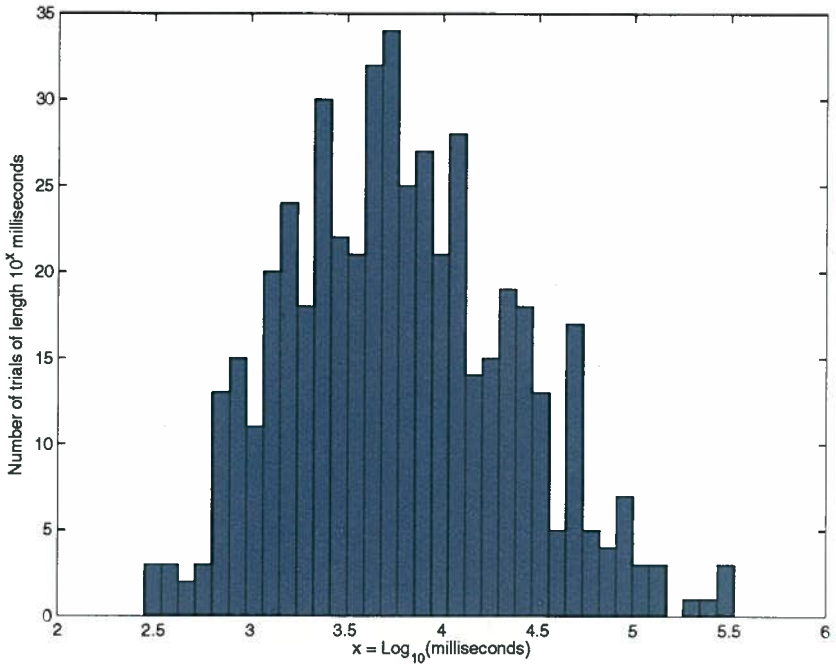

Figure 1: Logarithmic runtimes of VE solver with random elimination-ordering: *P(PropCost | Age = Adolescent, Airbag = False, Mileage = TwentyThou, MakeModel = Luxury)*

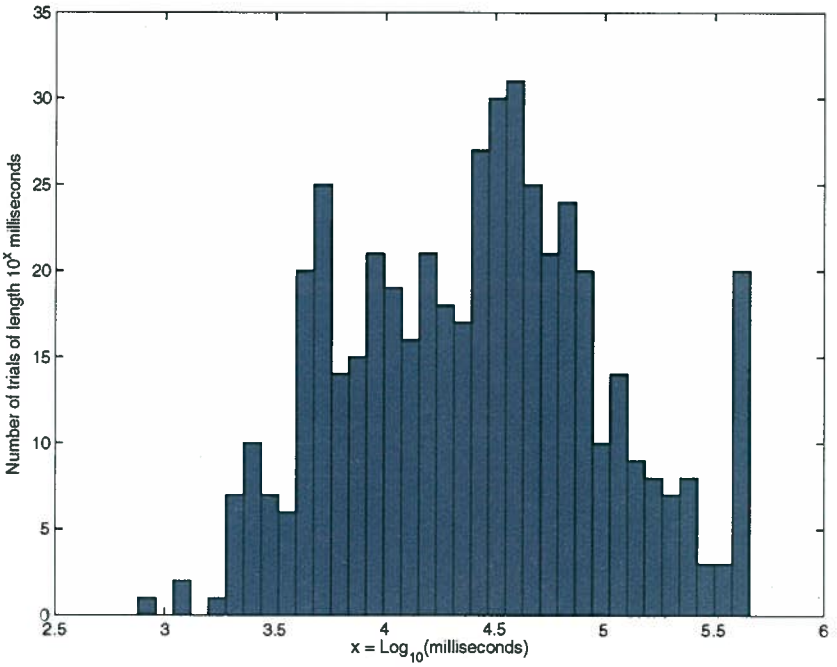

Figure 2: Logarithmic runtimes of VE solver with random elimination-ordering, query *P(PropCost | Age = Adolescent, Airbag = False, Mileage = TwentyThou, GoodStudent = True)*

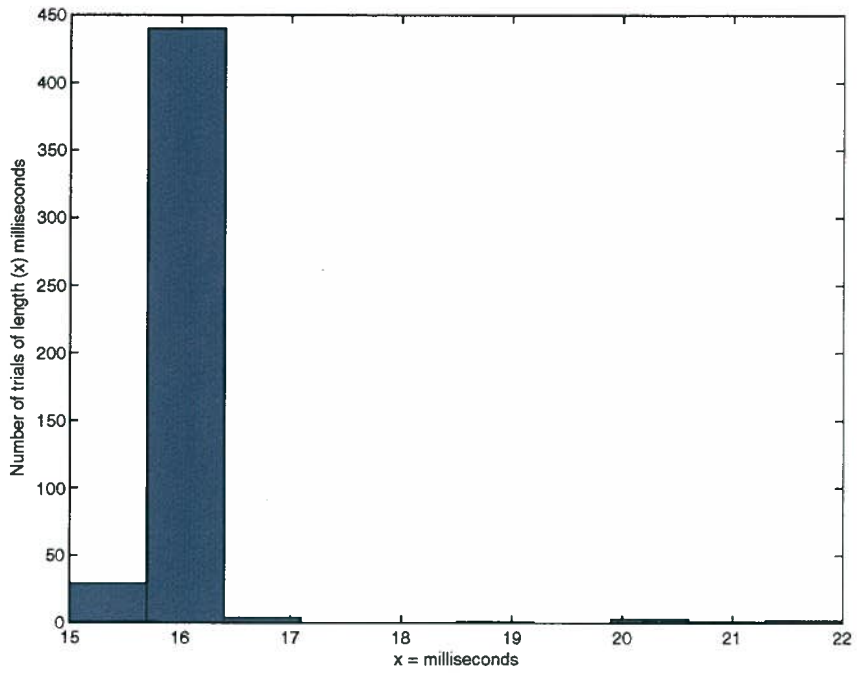

Figure 3: Runtimes of VE solver with random elimination-ordering, query $P(N104 \mid N41 = \text{“2”}, N84 = \text{“1”}, N116 = \text{“0”})$

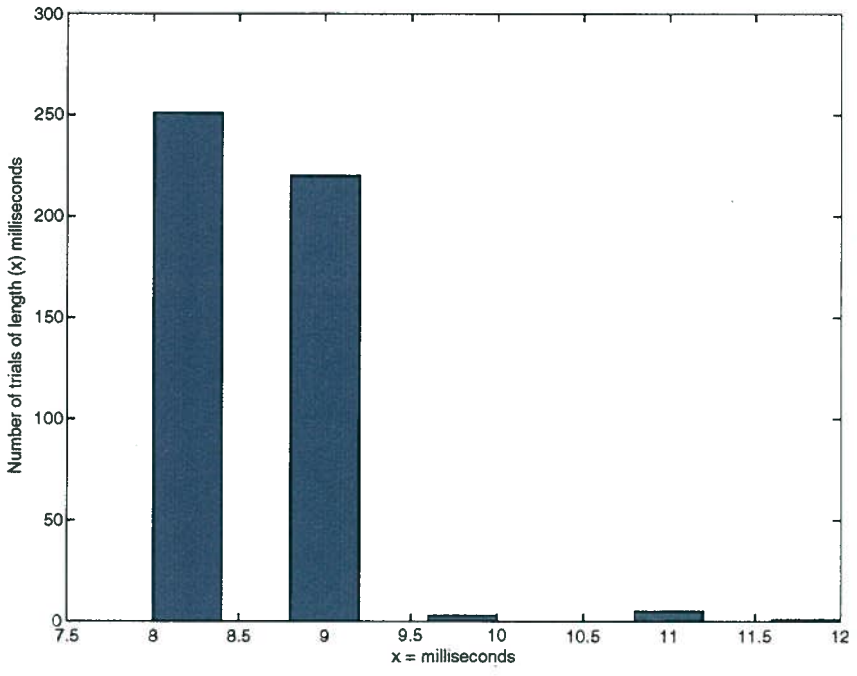

Figure 4: Runtimes of VE solver with random elimination-ordering, query $P(N73 \mid N152 = \text{“1”}, N116 = \text{“0”}, N43 = \text{“1”})$

to the exact distribution. Each experiment was repeated five times, and the mean and standard deviation are shown.

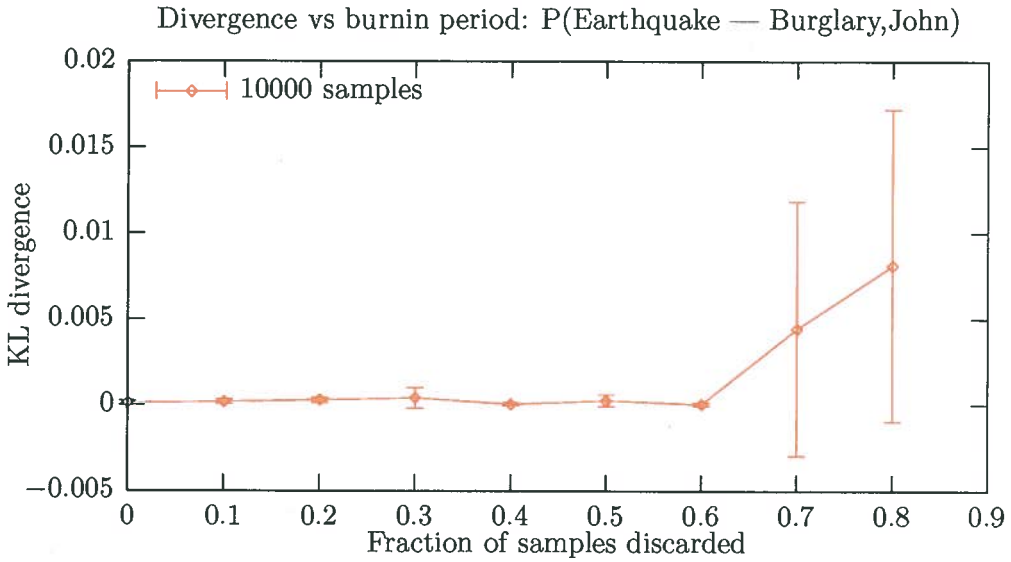

Figure 5: Result quality vs. fraction of discarded samples, Burglary network

The Burglary and Carpo networks do not give interesting results: the divergence remains very low with little variation until the burn-in time is increased to a large fraction (0.7 or greater). This suggests that the samples quickly converge to the stationary distribution; then there are so few samples from before the Markov chain has converged that they are easily outweighed by the rest of the sample, and so discarding them has little effect. It isn't surprising that this is the case for the Burglary and Carpo networks, since these are very simple queries. The Burglary network is quite small, so only a few passes will be required before all the nodes have been sampled from a distribution that takes the evidence into account. The queries in the Carpo network are independent of the evidence, which means both that much of the network can be ignored, and that sampling is from the prior distribution so there is effectively no mixing time. Finally, we observe that with a very large fraction of samples discarded, the divergence increases, as well as its variation between experiments — this is also no surprise, since the distribution is now being computed from a smaller number of samples, increasing the error.

The Insurance network query (Figure 6) displays the expected pattern: if few or no samples are discarded, the initial samples introduce error into the sampled distribution, and if too many samples are discarded, the error increases because the effective sample size is shrinking. The minimum error occurs somewhere in the middle. For this query, the graph shows that discarding the first 0.4 fraction of the samples gives the best results; the other Insurance queries (not shown) agree with this result. There is quite a bit of noise in the data, perhaps due to the influence that the random starting positions have on the Markov chain's trajectory.

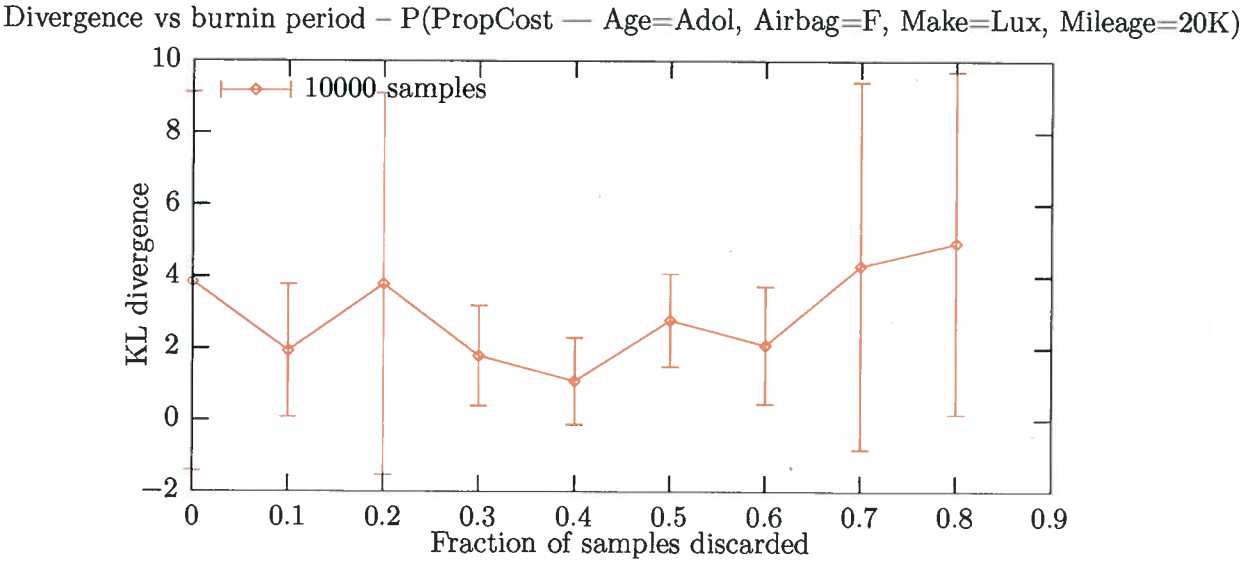

Figure 6: Result quality vs. fraction of discarded samples, Insurance network

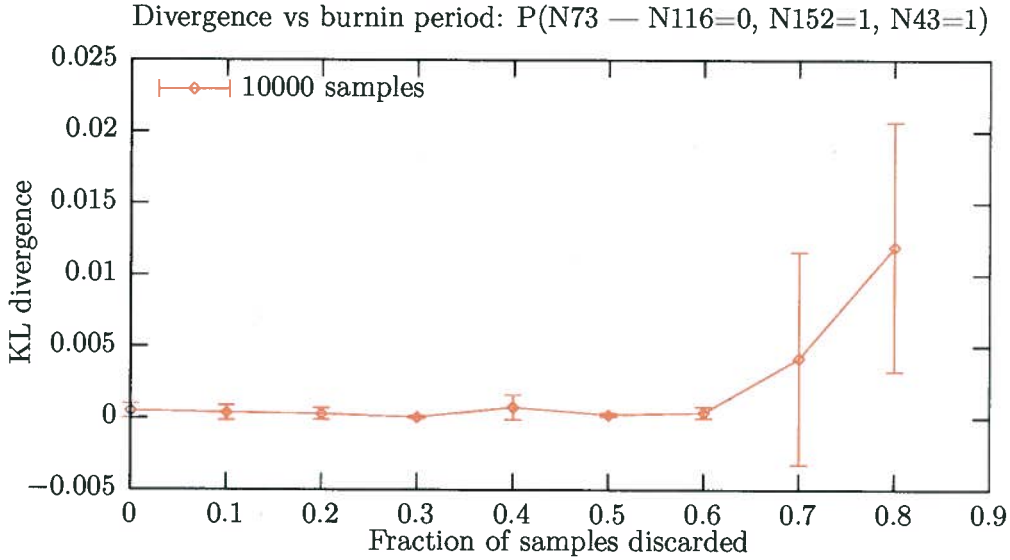

Figure 7: Result quality vs. fraction of discarded samples, Carpo network

3.2 Quality of Approximate Inference

To evaluate the quality of these inference algorithms, we tested them on four queries:

(1)	Insurance network:	$P(PropCost \mid Age = Adolescent, Airbag = False, Mileage = TwentyThou, MakeModel = Luxury)$
(2)	Insurance network:	$P(PropCost \mid Age = Adolescent, Airbag = False, Mileage = TwentyThou, GoodStudent = True)$
(3)	Carpo network:	$P(N104 \mid N41 = 2, N84 = 1, N116 = 0)$
(4)	Carpo network:	$P(N73 \mid N152 = 1, N116 = 0, N43 = 1)$

For each query, we ran both likelihood weighting and Gibbs sampling with varying numbers of samples, and plotted the Kullback-Leibler divergence of the sampled distribution relative to the exact distribution. For Gibbs sampling, an initial prefix of 0.4 times the number of samples was discarded for the burnin period, as per our results in Section 3.1. Likelihood weighting results are shown in Figures 8, 10, 12, and 14 (at the top of each page); Gibbs sampling results are shown in Figures 9, 11, 13, and 15 (at the bottom of each page). Each experiment was performed ten times; the individual experiment curves are shown on the left graph, and the mean and standard deviation are plotted on the right graph.

To put the number of samples in context, we compared the runtime of the variable elimination algorithm with the time taken for each sample. Table 4 shows the results. For VE, the greedy elimination order was used, as it gave the best performance.[3] From this, we find that on the order of 500 LW samples or 5000 Gibbs samples can be performed in the same time as VE for the Insurance network queries; for the Carpo network queries, these numbers are approximately 250 and 2500 respectively. The tenfold increase in sampling speed for the Gibbs sampler relative to LW comes from the fact that the LW sampler must evaluate and resample the entire network for each sample, while the Gibbs sample only needs to resample one node. The Insurance network contains 27 nodes, so an order of magnitude increase in runtime for the LW sampler is expected; we see only a ≈10-fold rather 27-fold increase because the Gibbs sampler must evaluate all the children of the selected node in addition to the node itself.

Query	VE runtime	LW time/sample	Gibbs time/sample
1	149	0.479	0.0365
2	271	0.515	0.0397
3	35	0.151	0.0131
4	14	0.058	0.0062

Table 4: Runtimes of sampling and VE algorithms (all times in ms)

Comparing Figures 8–11, we gain some insight about whether likelihood weighting or Gibbs sampling is preferable for different kinds of queries in the Insurance network. In query 1, Gibbs sampling

[3]Note that the runtimes for greedy VE in this experiment may differ from those presented in Section 2.3, as they were performed on a different test system.

is clearly preferable: with as low as 200 samples, the average KL divergence is below 5; likelihood weighting requires 500-1000 samples to reach this level (which takes many times longer). This means that likelihood weighting has little or no utility for this query, since in the time required to perform 500 LW samples, the exact solution could be computed with variable elimination. Inspecting the network, we can see why this query is a poor candidate for likelihood weighting: the evidence combination of $Age = Adolescent$ and $MakeModel = Luxury$ is rare, so most samples generated will have low weights, and there will be a large amount of error unless many samples are used. Gibbs sampling does not have this problem.

For query 2, likelihood weighting performs better than Gibbs sampling. Likelihood weighting gives a divergence slightly greater than 5 with only 20 samples, and below 1 when more than 40 samples are used. Gibbs sampling gives reasonable results when 1000 or more samples are used (KL divergence of about 5 with 1000 samples and less than 2 for more than 4000). However, these results are still inferior in quality than those from likelihood weighting, and they require more time. In addition, Gibbs sampling has a higher variance of divergence: even when repeating the experiment with the same number of samples, it is possible to have an unlucky run that gives a large error. Likelihood weighting shows much less variance. This query is well-suited for likelihood weighting, since most of the evidence variables ($Age = Adolescent$, $Mileage = TwentyThou$, $GoodStudent = True$) are located near the top of the graph, so forcing them to take their observed values does not greatly impact the particle weight.

The Carpo queries (3 and 4) are quite a different case because the query variables are d-separated from the evidence variables. Hence, we are effectively sampling the query values from the prior distribution. Both sampling algorithms give very good results for such queries. Likelihood weighting works well because the query variables are near the top of the network — the query variable in query 3 has only one ancestor, and the variable in query 4 has none at all! Indeed, for query 3 likelihood weighting will give a KL divergence of 0.5 with as few as 10 samples, and a divergence below 0.1 with 100 or more; Gibbs sampling gives a KL divergence below 0.5 with 200 samples, and below 0.1 with 1000 samples. For query 4, the results are even better: a divergence below 0.05 can be obtained with as few as 10 LW samples or 100 Gibbs samples. We can conclude that queries of this sort can be sampled very accurately without much effort — but this is not a particularly interesting result since the queries are so simple, with a graph in which most of the nodes can be ignored. Moreover, the ultimate value of sampling may be limited, since the exact solution to these queries can be computed in less than 50 ms by variable elimination.

A final observation is that the standard deviation of the divergences can be quite high when the number of sample is small, especially for Gibbs sampling. In particular, looking at the graphs of individual runs in Figures 9 and 11, we see that some of the runs give high-quality results, while there are some outliers with very large errors. Hence, running Gibbs sampling with few samples may a good result or a terrible one — unfortunately, one cannot tell which. This suggests that if few samples are performed, repeating the experiment multiple times and taking the average (or otherwise eliminating outliers) might be useful to give a better guarantee of a good result. (Of course, this increases runtime, so it would need to be compared against simply taking more samples in one experiment.)

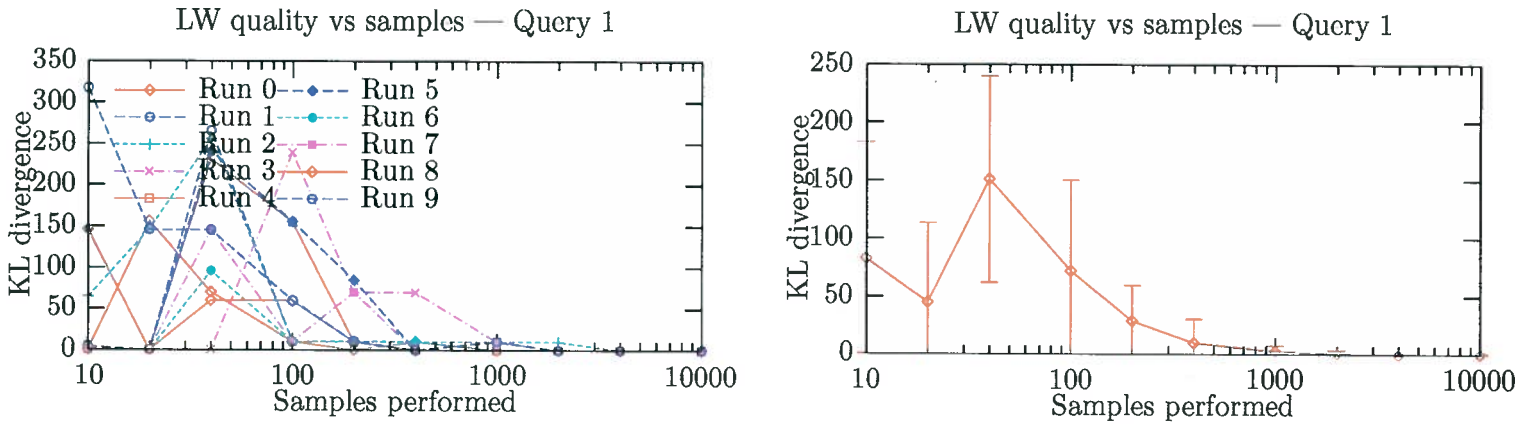

Figure 8: Divergence vs number of samples for LW sampling, query 1

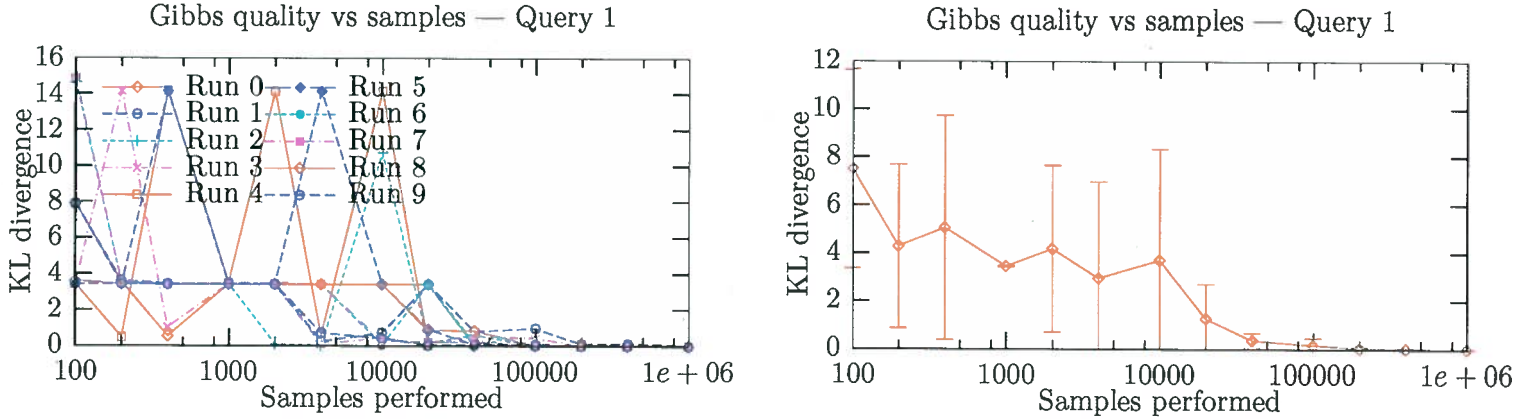

Figure 9: Divergence vs number of samples for Gibbs sampling, query 1

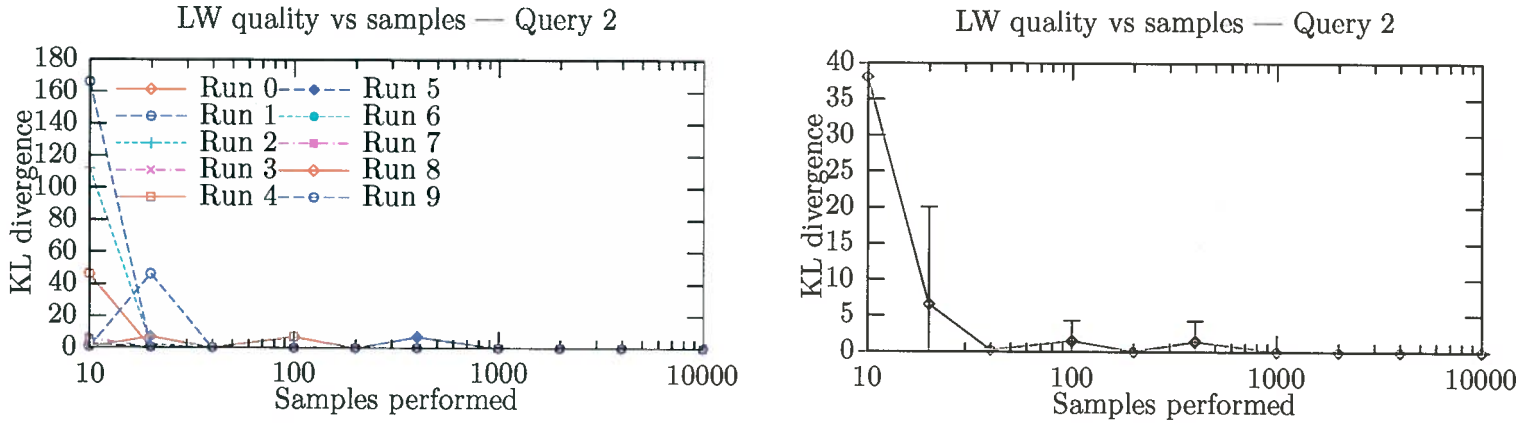

Figure 10: Divergence vs number of samples for LW sampling, query 2

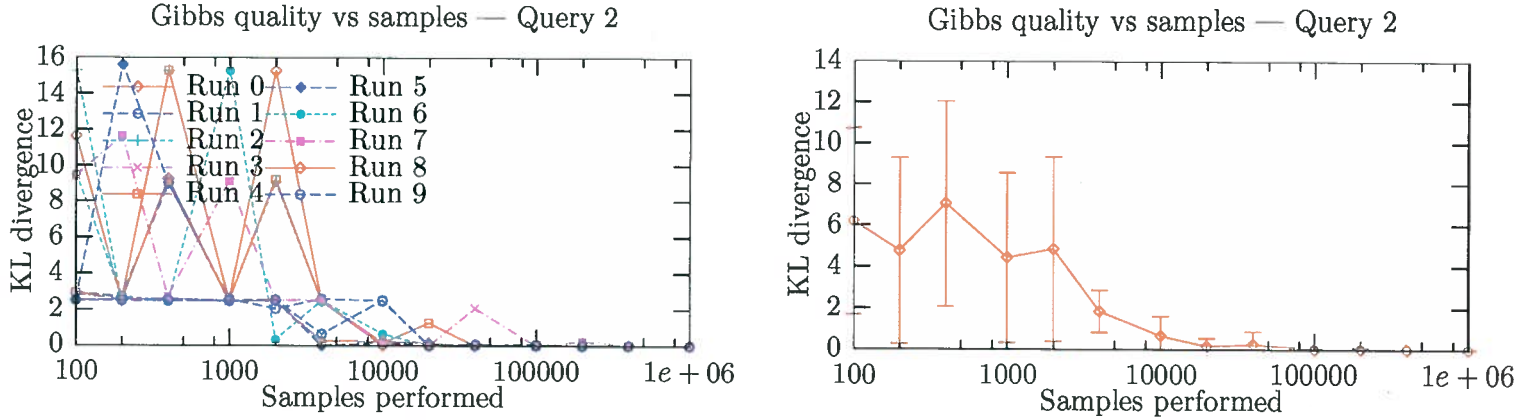

Figure 11: Divergence vs number of samples for Gibbs sampling, query 2

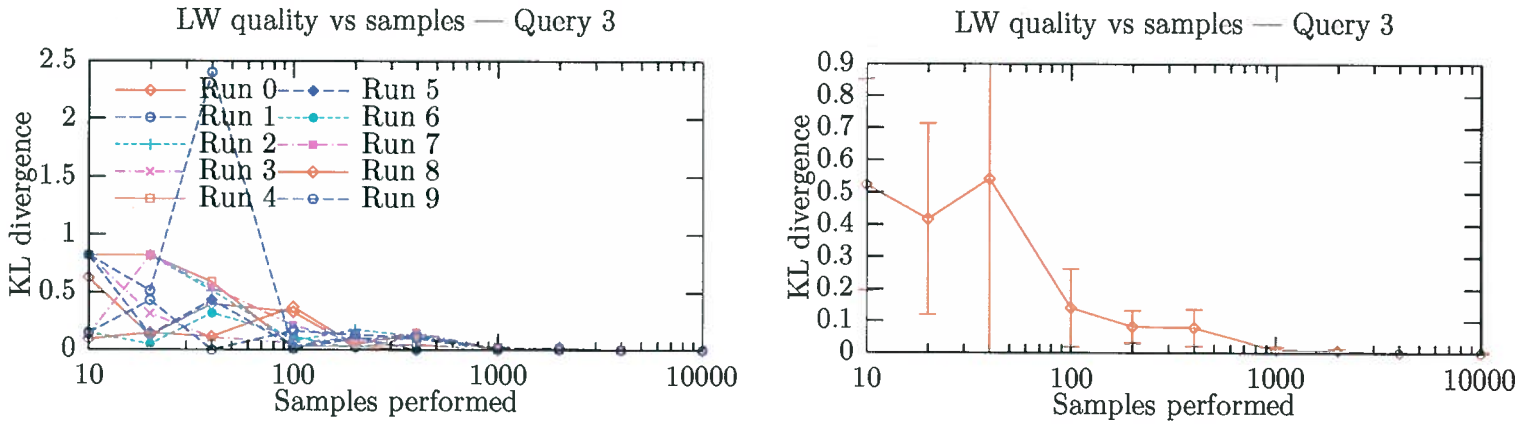

Figure 12: Divergence vs number of samples for LW sampling, query 3

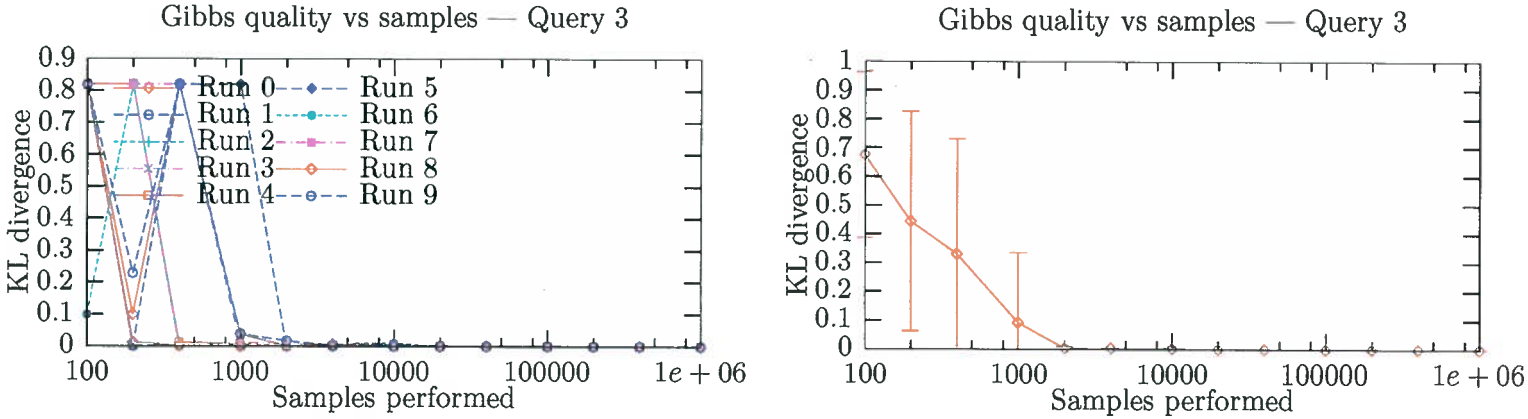

Figure 13: Divergence vs number of samples for Gibbs sampling, query 3

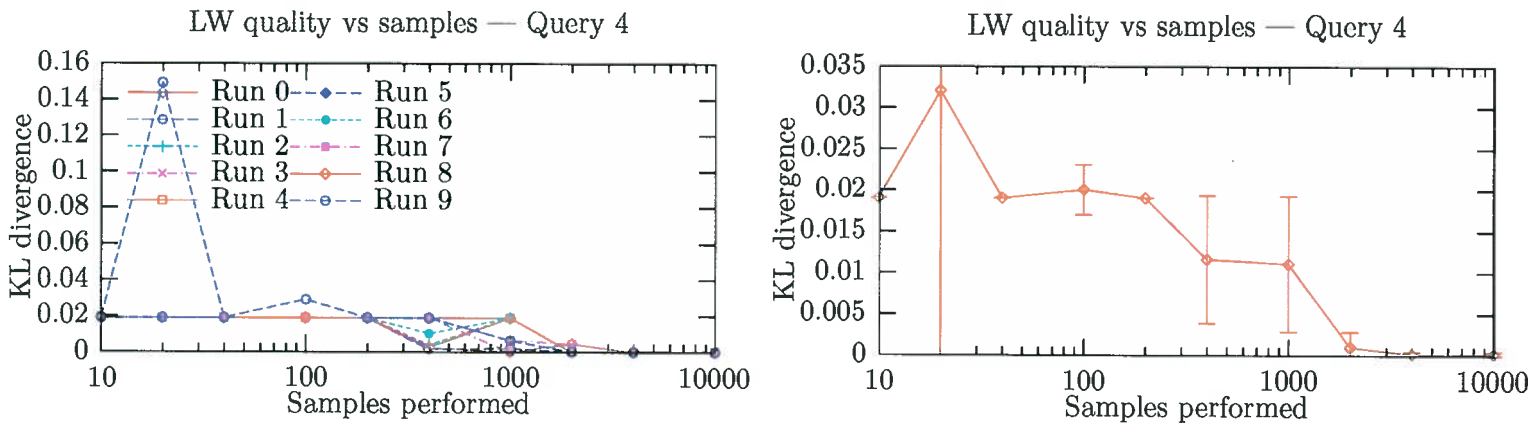

Figure 14: Divergence vs number of samples for LW sampling, query 4

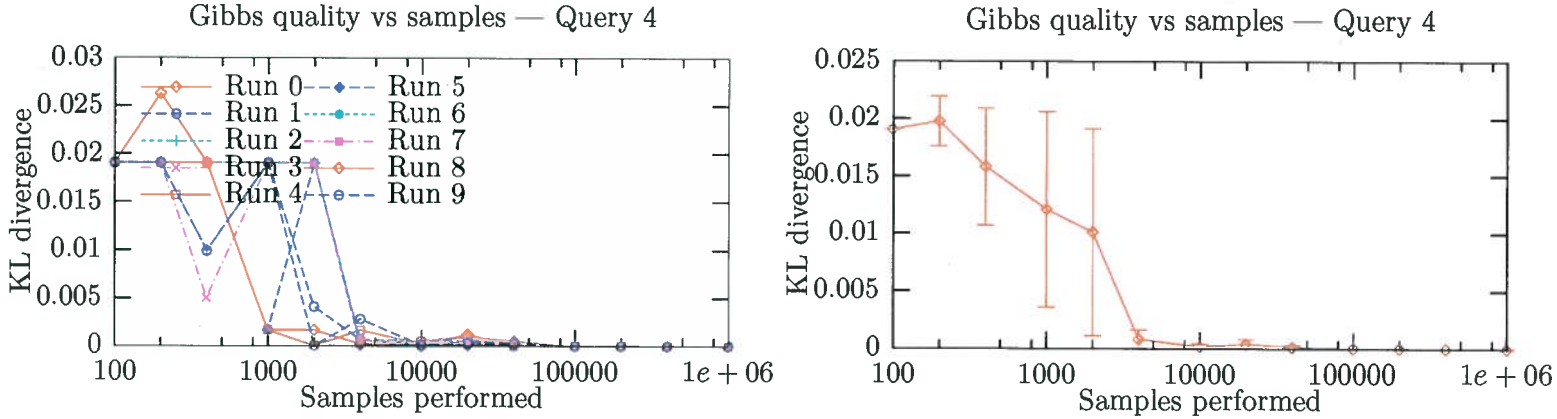

Figure 15: Divergence vs number of samples for Gibbs sampling, query 4

4 Conclusion

We implemented and studied a variety of Bayes'-network inference algorithms. Our experiments allowed the comparison of their relative performance, and assessment of the relative accuracy of inexact algorithms when run with various parameters. Additionally, analysis of our experimental results led to a deeper understanding of the issues affecting query performance when performing inference in Bayes' nets, and a better understanding of the appropriate choice of sampling algorithm and parameters for performing approximate inference in different Bayes' nets.

```
import edu.mit.six825.bn.functiontable.*;
import edu.mit.six825.bn.bayesnet.*;
import edu.mit.six825.bn.inputs.*;
import techniques.utils.Random;

import java.util.*;

/**
 * An implementation of the Variable Elimination bayes-nets solver,
 * implemented as per the pseudo-code provided in Koller & Friedman [2005]
 * pp.226-235
 *
 * @author nocturne
 */
public class VESolver extends Solver {
    protected static ElimOrderer _elimOrderFunc = new DumbOrder();

    public void setOrderer(ElimOrderer eo) {
        this._elimOrderFunc = eo;
    }

    public String toString() {
        return "VESolver";
    }

    public static void main(String[] args) {

        BayesNet bn = Nets.getBurglary();

        Solver solver = new VESolver();
        solver.setBayesNet(bn);

        try {
            Assignment evidence =
                ConstructEvidence(bn.nodes,
                                  new String[] {"JohnCalls", "true",
                                                "MaryCalls", "true"});
            solver.setEvidence(evidence);
        }
        catch (IllegalArgumentException e) {
            System.out.println("ERROR: " + e);
            return;
        }

        // Have to explicitly cast solver to a VESolver in order to
        // access setOrderer() method!

        ((VESolver)solver).setOrderer(new GreedyOrder());
        //((VESolver)solver).setOrderer(new RandomOrder());
        //((VESolver)solver).setOrderer(new DumbOrder());

        Function answer = solver.query(bn.nodes.getNode("Burglary"));
        System.out.println(answer);

        System.out.println("-----------------");

        bn = Nets.getInsurance();

        solver.setBayesNet(bn);
        try {
            Assignment evidence =
                ConstructEvidence(bn.nodes,
                                  new String[] {"Age", "Adolescent",
                                                "Airbag", "False",
                                                "Mileage", "TwentyThou"});
            solver.setEvidence(evidence);
        }
        catch (IllegalArgumentException e) {
            System.out.println("ERROR: " + e);
            return;
        }
        answer = solver.query(bn.nodes.getNode("PropCost"));
        System.out.println(answer);
    }

    public Function query (BayesNetNode variable) {
        FunctionVariable [] queryvar = new FunctionVariable[] {variable.var};

        Assignment e = _evidence;
        if (e == null) {
            // if evidence is null, whip up a null assignment and
            // splice that in, to keep the rest of our code happy.
            e = new Assignment(new FunctionVariableSet(queryvar[0]));
            e = e.subtract(e); // I.e. e = <the empty assignment>
            System.out.println("Warning: performing evidenceless query.");
        }

        BayesNetNodeSet bnset = GrabRelevantNodes(_bn.nodes, queryvar, e);
        System.out.println("BN shrank from " + _bn.nodes.size() + " to " +
                           bnset.size());

        return (CondProbVE(bnset, queryvar, e));
    }

    public VESolver(BayesNet _bn) {
        super(_bn);
    }
    public VESolver() {
        super();
    }


    /*************************************************************
     **                                                         **
     **                VE Algorithm Subroutines                 **
     **                                                         **
     *************************************************************/

    /**
     * @return normalized results of querying query variables, in
     * bayesnet bn, given evidence
     */
    public static Function CondProbVE (BayesNetNodeSet bnodes,
                                       FunctionVariable [] queryvars,
                                       Assignment evidence) {

        // Make sure we're only looking at the relevant nodes.
        bnodes = GrabRelevantNodes(bnodes, queryvars, evidence);

        Function [] cpts = new Function[bnodes.size()];

        for (int i=0; i < bnodes.size(); i++) {
            BayesNetNode n = bnodes.getNode(i);
            // System.out.println(n + " --> " + n.parents);
            if (n.cpt.variables.isSubsetOf(evidence.variables)) {
                // This node is rendered irrelevant by the evidence.
                // We will have to filter this null out in a moment...
                cpts[i] = null;
            } else if (n.cpt.variables.containsAnyOf(evidence.variables)) {
                cpts[i] = FilterForEvidence(n.cpt, evidence);
            } else {
```

```
            cpts[i] = n.cpt;
        }
    }
    // Strip out the null CPTs resulting from the cases where the
    // original CPT vars were a subset of the evidence vars
    cpts = RemoveNullElts(cpts);

    FunctionVariableSet toEliminate =
        VarsToElim(bnodes, queryvars, evidence);

    return Compute.normalize(SumProductVE(cpts,
                                          queryvars,
                                          toEliminate,
                                          _elimOrderFunc));
}

/**
 * @return product of factors remaining after elimination, with
 * variables elimvars eliminated in the order determined by the
 * ordering function ElimOrderer.
 */
public static Function SumProductVE(Function [] factors,
                                    FunctionVariable [] query,
                                    FunctionVariableSet toEliminate,
                                    ElimOrderer Orderer) {
    /* Implementation of algorithm on K&F p.228 */

    while (toEliminate.size() > 0) {
        FunctionVariable nextvar =
            Orderer.doNext(factors, toEliminate, query);

        toEliminate = toEliminate.subtract(nextvar);

        factors = SumProductEliminateVar(factors, nextvar);
    }

    return(FactorProduct(factors));
}

/**
 * @return factors, with variable elimvar eliminated
 */
public static Function[] SumProductEliminateVar(Function [] factors,
                                                FunctionVariable elimvar) {
    /* Implementation of algorithm on K&F p.228 */

    List AlongForTheRide = (List)new ArrayList();
    List Condensing      = (List)new ArrayList();

    for (int i=0; i < factors.length; i++) {
        if (factors[i].variables.contains(elimvar)) {
            Condensing.add(factors[i]);
        } else {
            AlongForTheRide.add(factors[i]);
        }
    }

    if (Condensing.isEmpty()) {
        String factstr = "";
        for (int i=0; i < factors.length; i++, factstr += ", ") {
            factstr += factors[i].variables.toString();
        }
        throw new IllegalArgumentException(
                  "Variable \"" + elimvar + "\" is not present in "
                  + "any of the given factors: " + factstr);
    }

    Function product = FactorProduct(FuncCollToFArray(Condensing));

    Function tau = FactorMarginalize(product, elimvar);

    // It's possible we had some evidence variable that was off in
    // la la land, somewhere completely disconnected from the
    // query (and hence irrelevant to the query). My code doesn't
    // attempt to detect this in advance, so when the last
    // remaining variable from the disconnected/irrelevant island
    // is marginalized, you get an empty factor. That is caught
    // here: when this happens, tau will be empty
    if (tau.variables.size() > 0) {
        // Strictly speaking, AFtR is no longer just along for the ride
        AlongForTheRide.add(tau);
    } else {
        System.out.println("Some evidence variable was irrelevant "
                           + "to the query.");
    }
    return (FuncCollToFArray(AlongForTheRide));
}

/**                                                                **
 **                                                                **
 **                                                                **
 ********************************************************************/

/********************************************************************
 **                                                                **
 **                    CPT/Factor Utilities                        **
 **                                                                **/

/**
 * @return factor product of the factors in array /funcs/
 */
public static Function FactorProduct(Function [] funcs) {
    if (funcs.length < 1) {
        throw new IllegalArgumentException("Cannot take null product");
    }
    // Start with first factor
    Function product = funcs[0];
    // Iterate over any remaining factors
    for (int i=1; i < funcs.length; i++) {
        product = FactorProduct(product, funcs[i]);
    }
    return (product);
}

/**
 * @return factor product of the CPTs Fa and Fb
 */
public static Function FactorProduct(Function Fa, Function Fb) {
    FunctionVariableSet FaVars = Fa.variables;
    FunctionVariableSet FbVars = Fb.variables;

    FunctionVariableSet ResultVars = FunctionVariableSet.union(FaVars,
                                                               FbVars);

    if (ResultVars.size() == (FaVars.size() + FbVars.size())) {
        throw new IllegalArgumentException(
                  "Can/should not compute factor product of functions "
                  + "with no common variables: " + FaVars.toString()
                  + " x " + FbVars.toString());
    }
```

```
    double [] entries = new double[ResultVars.cartesianProductSize()];

    System.out.println("Creating CPT of dimension " + ResultVars.size()
                       + " (" + entries.length + " entries)");
    /*
     * The structure of this Function-population mechanism was
     * largely stolen from the actual implementation of the
     * Function(FunctionVariableSet, double[]) constructor. The
     * expected form of that constructor's arguments is specified
     * only by the code (i.e. there's no documentation which
     * explains what's going on, so UTSL).
     */
    for (Iterator i = ResultVars.assignmentIterator(); i.hasNext(); ) {
        Assignment a = (Assignment) i.next();

        /* We could do index bookkeeping more efficiently here,
           but I have better things to worry about. */
        int index = a.computePosition();
        entries[index] = Fa.evaluate(a) * Fb.evaluate(a);
    }

    return new Function(ResultVars, entries);
}

/**
 * @return factor marginalization of /var/ in /cpt/ (sum out var from cpt)
 */
public static Function FactorMarginalize(Function cpt,
                                         FunctionVariable var) {
    if (!cpt.variables.contains(var)) {
        throw new IllegalArgumentException(
                    "Cannot marginalize variable " + var.toString()
                    + " in a CPT which does not contain it: "
                    + cpt.variables.toString());
    }

    FunctionVariableSet ResultVars = cpt.variables.subtract(var);

    double [] entries = new double[ResultVars.cartesianProductSize()];

    /*
     * See comments in FactorProduct regarding use of the Function
     * constructor.
     */
    for (Iterator i = ResultVars.assignmentIterator(); i.hasNext(); ) {
        Assignment a = (Assignment) i.next();

        // sum over the domain of var
        double sum = 0;
        for (final Iterator j = var.domain.iterator(); j.hasNext(); ) {
            final DomainIndex dindex = (DomainIndex)j.next();
            final Comparable value = dindex.getValue();

            final Assignment subass = new Assignment(a, var, value);

            sum += cpt.evaluate(subass);
        }

        /* We could do index bookkeeping more efficiently here,
           but I have better things to worry about. */
        int index = a.computePosition();
        entries[index] = sum;
    }

    return new Function(ResultVars, entries);
}

/**                                                                    **
 **                                                                    **
 **                                                                    **
 ************************************************************************/

/************************************************************************
 **                                                                    **
 **                      Elimination Orderers                          **
 **                                                                    **/

// An abstract interface which effectively lets us have function
// pointers to elimination-ordering functions.
//
// Arguments: bnset -- must contain all bayes-net nodes relevant
//                     to given query+evidence
//            query -- array of query variables
//            evidence -- assignment of evidence variables
//
// Returns: array of FunctionVariables in order of elimination
//           (index 0 is first to be eliminated)
//
public static abstract interface ElimOrderer {
    public FunctionVariable doNext(Function [] factors,
                                   FunctionVariableSet eliminating,
                                   FunctionVariable [] query);
}

// An elimination-ordering instantiation which orders the
// elimination variables randomly. (Nondeterministically: order
// will be different on from invocation to invocation.)
public static class RandomOrder implements ElimOrderer {
    public FunctionVariable doNext(Function [] factors,
                                   FunctionVariableSet eliminating,
                                   FunctionVariable [] query) {
        FunctionVariable fv =
            eliminating.getVariable(Random.random(eliminating.size()));
        return(fv);
    }
}

// A dumb orderer implementation which orders the variables in the
// order they're given in the FVSet /eliminating/. (Which is
// deterministically ordered for a given query method, so this
// will generally have the same results given a particular query
// formulation.)
public static class DumbOrder implements ElimOrderer {
    public FunctionVariable doNext(Function [] factors,
                                   FunctionVariableSet eliminating,
                                   FunctionVariable [] query) {
        return(eliminating.getVariable(0));
    }
}

// A greedy elimination-ordering instantiation which picks the
// variable whose elimination from the current set of factors
// yields the smallest resulting factor.
public static class GreedyOrder implements ElimOrderer {
    public FunctionVariable doNext(Function [] factors,
                                   FunctionVariableSet eliminating,
                                   FunctionVariable [] query) {
        // seed with bogus values that will be immediately overwritten
        int smallestsize = java.lang.Integer.MAX_VALUE;
```

```
        FunctionVariable bestvar = eliminating.getVariable(0);

        for (int i=0; i < eliminating.size(); i++) {
            int size = WouldMakeFactorSize(eliminating.getVariable(i),
                                           factors);
            if (size < smallestsize) {
                smallestsize = size;
                bestvar = eliminating.getVariable(i);
            }
        }
        return(bestvar);
    }
}

/**                                                                **
 **                                                                **
 **                                                                **
 ********************************************************************/

/********************************************************************
 **                                                                **
 **                  Random Utility Functions                      **
 **                                                                **/

/**
 * @return the size (# of entries) of the CPT which would be
 * created by the elimination of var from the given factors.
 */
public static int WouldMakeFactorSize(FunctionVariable var,
                                      Function [] factors) {
    // Really, we want an empty FuncVarSet here, but FVS.union
    // assumes that no FVSet is empty (and will blow out if that's
    // not the case), and so we can't do that. So, instead,
    // initialize vars to contain just the query var, which we're
    // going to remove at the end anyway; this yields the desired
    // end result.
    FunctionVariableSet vars =
        new FunctionVariableSet(new FunctionVariable[] {var});

    for (int i=0; i < factors.length; i++) {
        if (factors[i].variables.contains(var)) {
            vars = FunctionVariableSet.union(vars,
                                             factors[i].variables);
        }
    }
    return(vars.subtract(var).cartesianProductSize());
}

/**
 * @return set of variables which will need to be eliminated from
 * bnodes in order to answer a query on queryvars (given evidence)
 */
public static FunctionVariableSet VarsToElim(BayesNetNodeSet bnodes,
                                             FunctionVariable[] queryvars,
                                             Assignment evidence) {
    FunctionVariableSet bnVars = bnodes.getFunctionVariableSet();
    FunctionVariableSet  qVars = new FunctionVariableSet(queryvars);
    FunctionVariableSet  eVars = evidence.variables;

    return(bnVars.subtract(qVars).subtract(eVars));
}


/**
 * @return nodeset of our query/evidence nodes plus those nodes
 * from bayesnet bn which are relevant to (i.e. an ancestor of)
 * the query/evidence nodes.
 */
public static BayesNetNodeSet GrabRelevantNodes(BayesNetNodeSet bnodes,
                                                FunctionVariable [] queryvars,
                                                Assignment evidence) {

    // Set of nodes we care about
    Set childNodes = (Set) new HashSet();

    // Feed query variables into child set
    for (int i=0; i < queryvars.length; i++) {
        childNodes.add(FindVariableNode(bnodes, queryvars[i]));
    }

    // And all the evidence variables
    FunctionVariableSet evars = evidence.variables;
    for (int i=0; i < evars.size(); i++) {
        childNodes.add(FindVariableNode(bnodes, evars.getVariable(i)));
    }

    BayesNetNodeSet childSet = NodeCollToBNNSet(childNodes);

    // And return the child nodes plus all their ancestors
    return (NodesAndAncestors(childSet));
}

public static BayesNetNodeSet NodesAndAncestors(BayesNetNodeSet nodes) {
    // Fringe of relevant nodes to be expanded in the BN graph
    List nodeFringe   = (List) new ArrayList();
    // Set of the previously-expanded nodes and/or ancestors
    Set familyNodes = (Set) new HashSet();

    // Initialize fringe to be all our input nodes
    for (int i=0; i < nodes.size(); i++) {
        nodeFringe.add(nodes.getNode(i));
    }

    // Process and expand the fringe until we're done
    while (nodeFringe.size() > 0) {
        // Grab some node from the fringe
        BayesNetNode n = (BayesNetNode) nodeFringe.remove(0);

        // And, if we haven't already looked at it...
        if (!familyNodes.contains(n)) {
            // ...add it to the relevant node set
            familyNodes.add(n);
            // ...and add its parents to the fringe
            for (int i=0; i< n.parents.size(); i++) {
                nodeFringe.add(n.parents.getNode(i));
            }
        }
    }

    return(NodeCollToBNNSet(familyNodes));
}

// Syntactic sugar for converting a Func collection to an array
public static Function[] FuncCollToFArray(Collection fcoll) {
    return((Function[]) fcoll.toArray(new Function[0]));
}

// Syntactic sugar for converting a BNN collection to a BNNSet
public static BayesNetNodeSet NodeCollToBNNSet(Collection ncoll) {
    BayesNetNode [] narray =
```

```
        (BayesNetNode[]) ncoll.toArray(new BayesNetNode[0]);
    return(new BayesNetNodeSet(narray));
}

/**
 * @return BayesNet node for var (from network /bnodes/)
 */
public static BayesNetNode FindVariableNode(BayesNetNodeSet bnodes,
                                            FunctionVariable var) {
    for (final Iterator i = bnodes.iterator(); i.hasNext(); ) {
        BayesNetNode n = (BayesNetNode) i.next();
        if (n.var.equals(var)) {
            return (n);
        }
    }
    throw new IllegalArgumentException(
                    "Could not find node for variable " + var);
}

/**
 * @return cpt|_(E=e)_ --- i.e. construct and return a filtered
 * CPT (based on the input CPT) which is consistent with the
 * evidence e. The evidence variables will not be in the scope of
 * the resulting CPT; the resulting scope will be the scope of the
 * input CPT minus the set of variables assigned by the evidence.
 *
 * This is used by Cond-Prob-VE line 2 (K&F p. 235).
 */
public static Function FilterForEvidence(Function cpt, Assignment e) {
    FunctionVariableSet EVars      = e.variables;
    FunctionVariableSet ResultVars = cpt.variables.subtract(EVars);

    if (ResultVars.size() == 0) {
        throw new IllegalArgumentException(
           "FilterForEvidence should never be given cpt+evidence "
           + "where cpt vars are a subset of evidence vars");
    }

    // For efficiency's sake, remove from EVars the variables not
    // present in this particular CPT:
    FunctionVariableSet IgnorableVars = EVars.subtract(cpt.variables);
    EVars = EVars.subtract(IgnorableVars);

    double [] entries = new double[ResultVars.cartesianProductSize()];
    /*
     * See comments in FactorProduct regarding use of the Function
     * constructor.
    */
    for (Iterator i = ResultVars.assignmentIterator(); i.hasNext(); ) {
        /* We're iterating over all possible assignments of
           the *result* variables */
        Assignment ass = (Assignment) i.next();

        // fold the assignments from our evidence into this assignment
        for (int j=0; j < EVars.size(); j++) {
            FunctionVariable v = EVars.getVariable(j);
            ass = new Assignment(ass, v, e.getAssignedValue(v));
        }

        double wanted = cpt.evaluate(ass);

        /* compute the position only relative to the ResultVars
           (rather than all the vars in the assignment): only a subset
           of the assignment variables are actually indices in the
           function/CPT we're constructing. */
        /* (Note, Assignment.computePosition(FunctionVariableSet) is
           not really documented, but its usage elsewhere indicates
           the vars in the assignment must be a superset of the vars
           in the FunctionVariableSet.) */
        int index = ass.computePosition(ResultVars);
        entries[index] = wanted;
    }

    return new Function(ResultVars, entries);
}

/**
 * @return Comparable object corresponding to value with
 * stringification "val" in domain of FunctionVariable /var/.
 * (This lets us look up Comparable objects on the basis of their
 * string representation, which is kind of handy.)
 */
public static Comparable FindVariableValueObj(FunctionVariable var,
                                              String val) {
    Domain d = var.domain;
    for (int i=0; i < d.size(); i++) {
        if (d.getValue(i).toString().equals(val)) {
            return(d.getValue(i));
        }
    }
    throw new IllegalArgumentException(d + " does not include value \""
                                       + val + "\"!");
}

/**
 * @return Assignment corresponding to the evidence specified by
 * estrings, according to nodes in bn. Estrings must contain an
 * even number of strings: for any even /x/ estrings[x] is the
 * (stringified) name of a variable, and estrings[x+1] is the
 * (stringified) value to which that variable should be fixed.
 */
public static Assignment ConstructEvidence(BayesNetNodeSet bn,
                                           String[] estrings) {
    if ((estrings.length % 2) != 0) {
        throw new IllegalArgumentException("ConstructEvidence given "
                    + "odd number of strings: " + estrings);
    }

    FunctionVariable [] eVars = new FunctionVariable[estrings.length/2];
    Comparable       [] eVals = new Comparable[estrings.length/2];

    for (int i=0; i < estrings.length; i += 2) {
        String nameString = estrings[i];
        String valString  = estrings[i+1];
        BayesNetNode n = bn.getNode(nameString);
        if (n == null) {
            throw new IllegalArgumentException("No such node \""
                                               + nameString + "\"");
        }
        eVars[i/2] = n.var;
        eVals[i/2] = FindVariableValueObj(n.var, valString);
    }

    return(new Assignment(new FunctionVariableSet(eVars), eVals));
}

/**
 * @return array containing the non-null elements of arr.
 */
public static Function[] RemoveNullElts(Function[] arr) {
```

```
        int nulls = 0;
        for (int i=0; i<arr.length; i++) {
            if (arr[i] == null) nulls++;
        }
        if (nulls == 0) {
            return arr;
        } else {
            Function[] answer = new Function[arr.length - nulls];
            int j=0;
            for (int i=0; i<arr.length; i++) {
                if (arr[i] != null) {
                    answer[j++] = arr[i];
                }
            }
            return(answer);
        }
    }

}
```

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import edu.mit.six825.bn.functiontable.*;
import edu.mit.six825.bn.bayesnet.*;
import edu.mit.six825.bn.inputs.*;
import techniques.utils.*;
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;

/**
 * An implementation of the Gibbs sampling algorithm for Bayes nets.
 *
 * @author dkp
 */
public class GibbsSolver extends Solver {
    private static final boolean DEBUG = false;

    private int discardPrefixLen = 5000;
    private int requiredSamples = 10000;
    private boolean randomFlipChoice = false;

    private Map/*<BayesNetNode, List<BayesNetNode>>*/ childMap;

    public GibbsSolver(BayesNet _bn) {
        super(_bn);
    }

    public GibbsSolver() {
        super();
    }

    /**
     * Set the number of samples that are discarded while the Markov chain
     * converges. Note that the total number of samples performed is still
     * that specified by setRequiredSamples, i.e. the total number that are
     * actually used is requiredSamples - discardPrefixLen.
     */
    public void setDiscardPrefixLen(int newLen) {
        discardPrefixLen = newLen;
    }

    /**
     * Set the number of samples performed
     */
    public void setRequiredSamples(int newSamples) {
        requiredSamples = newSamples;
    }

    /**
     * If set, the variable to be flipped in each sample will be chosen
     * uniformly at random rather than in systematic passes.
     */
    public void randomFlipChoice(boolean newVal) {
        randomFlipChoice = newVal;
    }


    /**
     * Generate an initial sample with all non-evidence variables selected
     * randomly.
     */
    private Assignment initialSample(BayesNetNodeSet bnset,
                                     Assignment evidence) {
        Assignment r = evidence;

        for (Iterator i =
                 bnset.getNodesWithTopologicalOrdering().iterator();
             i.hasNext();) {
            BayesNetNode node = (BayesNetNode) i.next();
            if (!evidence.contains(node.var)) {
                Comparable val = SampleUtils.sampleNode(node, r, true, null);
                r = new Assignment(r, node.var, val);
            }
        }

        return r;
    }

    /**
     * Resample the given node in the network, choosing a new node at random
     * with the probability distribution given by the current assignment of
     * the rest of the network.
     */
    private Assignment flipNode(Assignment last, BayesNetNode node) {
        Comparable val = SampleUtils.sampleNode(node, last, false, childMap);
        if (DEBUG) {
            System.out.println("Flipping node " + node.var + " to " + val);
        }

        return new Assignment(last, node.var, val);
    }

    public Function query (BayesNetNode node) {
        System.out.println("Query for " + node);
        System.out.println("Samples = " + requiredSamples);
        System.out.println("Discard = " + discardPrefixLen);


        FunctionVariable [] queryvar = new FunctionVariable[] {node.var};
        BayesNetNodeSet bnset = VESolver.GrabRelevantNodes(_bn.nodes,
                                                           queryvar,
                                                           _evidence);

        childMap = SampleUtils.buildChildMap(bnset);

        double [] prob = new double[node.var.domain.size()];

        Assignment x = initialSample(bnset, _evidence);

        int sampleCount = 0;


        loop:
        while (true) {
            for (int i = 0; i < bnset.size(); i++) {
                if (sampleCount >= requiredSamples) {
                    break loop;
                }

                BayesNetNode nodeToBeFlipped;
                // Node to be flipped is either sequential (given by the for
                // loop) or random
                if (randomFlipChoice) {
                    nodeToBeFlipped =
                        bnset.getNode(Random.random(bnset.size()));
                } else {
                    nodeToBeFlipped = bnset.getNode(i);
```

```
                }

                // ...but it better not be an evidence variable.
                if (_evidence.contains(nodeToBeFlipped.var)) {
                    continue;
                }

                x = flipNode(x, nodeToBeFlipped);

                sampleCount += 1;
                if (sampleCount > discardPrefixLen) {
                    Comparable val = x.getAssignedValue(node.var);
                    int ind = node.var.domain.getIndex(val).i;
                    prob[ind] += 1;
                }
            }
        }

        return Compute.normalize(new Function(node.var, prob));
    }

    public String toString() {
        return "GibbsSolver";
    }

    public static void main(String[] args) {
        System.out.println("Prob(Burglary|JohnCalls=true, MaryCalls=true)");
        System.out.println("Burglary=TRUE AIMA: " + 0.284);

        final BayesNet bn = edu.mit.six825.bn.inputs.Nets.getBurglary();
        final Solver solver = new GibbsSolver();
        solver.setBayesNet(bn);
        //Solver solver = new EnumerationSolver(bn);
        // ...GibbsSamplerSolver(bn);
        // ...LikelihoodWeightingSolver(bn);
        // ...VariableEliminationSolver(bn);

        final FunctionVariable[] vars = new FunctionVariable[2];
        vars[0] = new FunctionVariable("JohnCalls");
        vars[1] = new FunctionVariable("MaryCalls");
        final Comparable[] vals = new Comparable[2];
        vals[0] = ComparableBoolean.TRUE;
        vals[1] = ComparableBoolean.TRUE;
        final Assignment evidence = new Assignment(new FunctionVariableSet(vars), vals
);
        solver.setEvidence(evidence);

        final BayesNetNode burgVar = bn.nodes.getNode("Burglary");
        final Function burgProb = solver.query(burgVar);
        System.out.println("Burglary=TRUE Calc.: " + burgProb);
    }

}
```

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import edu.mit.six825.bn.functiontable.*;
import edu.mit.six825.bn.bayesnet.*;
import edu.mit.six825.bn.inputs.*;
import techniques.utils.*;
import java.util.HashMap;

/**
 * An implementation of the likelihood weighting sampling algorithm
 * for Bayes nets.
 *
 * @author drkp
 */
public class LWSolver extends Solver {

    private static final double DEFAULT_REQUIRED_WEIGHT = 100.0;
    private double requiredWeight = DEFAULT_REQUIRED_WEIGHT;

    public LWSolver(BayesNet _bn) {
        super(_bn);
    }

    public LWSolver() {
        super();
    }

    public void setRequiredWeight(double newWeight) {
        requiredWeight = newWeight;
    }

    /**
     * A weighted sample -- just an assignment with a weight
     */
    public class WeightedParticle {
        public double weight;
        public Assignment assignment;

        public WeightedParticle(double _weight, Assignment _assignment) {
            assignment = _assignment;
            weight = _weight;
        }
    }

    /**
     * Generate a random sample satisfying the evidence, along with the
     * appropriate weight. This is implemented per procedure "LW-sample"
     * (Koller & Friedman p.330)
     */
    private WeightedParticle generateParticle(BayesNetNodeSet bnset) {
        double weight = 1.0;
        Assignment x = _evidence;       // sampled assignment so far
        List ordering = bnset.getNodesWithTopologicalOrdering();

//      System.out.println("Generating particle\n");
        for (Iterator i = ordering.iterator(); i.hasNext();) {
            BayesNetNode node = (BayesNetNode) i.next();

            if (_evidence.contains(node.var)) {
                double[] dist = SampleUtils.evaluateNodeGivenParents(node, x);
                Comparable val = _evidence.getAssignedValue(node.var);
                x = new Assignment(x, node.var, val);
                int ind = node.var.domain.getIndex(val).i;
                weight *= dist[ind];
//              System.out.println("Forcing " + node.var + " to " +
//                                 x.getAssignedValue(node.var));
//              System.out.println("Weight -> " + weight);
            } else {
                x = new Assignment(x, node.var,
                                   SampleUtils.sampleNode(node, x,
                                                          true, null));
//              System.out.println("Sampling " + node.var + " as " +
//                                 x.getAssignedValue(node.var));
            }
        }

        return new WeightedParticle(weight, x);
    }

    public Function query (BayesNetNode node) {
        System.out.println("Query for " + node);
        System.out.println("Required weight: " + requiredWeight);

        FunctionVariable [] queryvar = new FunctionVariable[] {node.var};
        BayesNetNodeSet bnset = VESolver.GrabRelevantNodes(_bn.nodes,
                                                           queryvar,
                                                           _evidence);

        double totalWeight = 0.0;
        int numSamples = 0;
        double [] prob = new double[node.var.domain.size()];
        while (numSamples < requiredWeight) {
            WeightedParticle particle = generateParticle(bnset);
            totalWeight += particle.weight;

            Comparable val = particle.assignment.getAssignedValue(node.var);
            int i = node.var.domain.getIndex(val).i;
            prob[i] += particle.weight;
            numSamples++;
        }

        // Normalize.
        for (int i = 0; i < node.var.domain.size(); i++) {
            prob[i] /= totalWeight;
        }

        return new Function(node.var, prob);
    }

    public String toString() {
        return "LWSolver";
    }

    public static void main(String[] args) {
        System.out.println("Prob(Burglary|JohnCalls=true, MaryCalls=true)");
        System.out.println("Burglary=TRUE AIMA: " + 0.284);

        final BayesNet bn = edu.mit.six825.bn.inputs.Nets.getBurglary();
        final Solver solver = new LWSolver();
        solver.setBayesNet(bn);
        //Solver solver = new EnumerationSolver(bn);
        // ...GibbsSamplerSolver(bn);
        // ...LikelihoodWeightingSolver(bn);
        // ...VariableEliminationSolver(bn);

        final FunctionVariable[] vars = new FunctionVariable[2];
        vars[0] = new FunctionVariable("JohnCalls");
        vars[1] = new FunctionVariable("MaryCalls");
```

```
        final Comparable[] vals = new Comparable[2];
        vals[0] = ComparableBoolean.TRUE;
        vals[1] = ComparableBoolean.TRUE;
        final Assignment evidence = new Assignment(new FunctionVariableSet(vars), vals
);
        solver.setEvidence(evidence);

        final BayesNetNode burgVar = bn.nodes.getNode("Burglary");
        final Function burgProb = solver.query(burgVar);
        System.out.println("Burglary=TRUE Calc.: " + burgProb);
    }

}
```

```
import java.util.Map;
import java.util.HashMap;
import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;

import edu.mit.six825.bn.functiontable.*;
import edu.mit.six825.bn.bayesnet.*;
import edu.mit.six825.bn.inputs.*;
import techniques.utils.*;



/**
 * Some utilities for implementing sampling-based solvers
 *
 * @author drkp
 */
public class SampleUtils {
    private static final boolean DEBUG = false;

    private SampleUtils() {

    }

    public static Map buildChildMap(BayesNetNodeSet bnset) {
        Map childMap = new HashMap();
        for (Iterator it = bnset.iterator(); it.hasNext();) {
            BayesNetNode node = (BayesNetNode) it.next();
            childMap.put(node, new ArrayList());
        }

        for (Iterator it = bnset.iterator(); it.hasNext();) {
            BayesNetNode parent = (BayesNetNode) it.next();
            for (Iterator it2 = bnset.iterator(); it2.hasNext();) {
                BayesNetNode child = (BayesNetNode) it2.next();
                if (child.parents.contains(parent)) {
                    List lst = (List) childMap.get(parent);
                    lst.add(child);
                }

            }
        }

        return childMap;
    }


    /**
     * Return the probability of each value in the node's variable's domain,
     * given an assignment that includes its parents values.
     */
    public static double [] evaluateNodeGivenParents(BayesNetNode node,
                                                     Assignment evidence) {
        double [] entries = new double[node.var.domain.size()];

        for (int i = 0; i < node.var.domain.size(); i++) {
            Comparable val = node.var.domain.getValue(i);
            Assignment fullAss = new Assignment(evidence, node.var, val);
            entries[i] = node.cpt.evaluate(fullAss);
        }

        return entries;
    }
```

```
    /**
     * Return the probability of each value in the node's variable's domain,
     * consistent with the current assignments of all of its children. This
     * requires that the assignment provided includes the full Markov blanket
     * (parents, children, and children's parents) of the given node.
     *
     * Implemented per formula (14.11) of AIMA.
     */
    public static double [] evaluateNodeGivenMarkovBlanket(BayesNetNode node,
                                                           Assignment as,
                                                           Map childMap) {

        Function cpt = node.cpt;
        double sum = 0;

        if (DEBUG) {
            System.out.println("Evaluating node " + node.var);
            System.out.println("Assignment is " + as);
        }

        double [] entries = new double[node.var.domain.size()];

        for (int i = 0; i < node.var.domain.size(); i++) {
            Comparable val = node.var.domain.getValue(i);
            Assignment fullAss = new Assignment(as, node.var, val);
            if (DEBUG) {
                System.out.println(" Considering assignment " + val);
            }

            entries[i] = cpt.evaluate(fullAss);
            if (DEBUG) {
                System.out.println("  CPT evaluates to " + entries[i]);
            }

            // Multiply by the probabilities of each child
            for (Iterator it2 = ((List)childMap.get(node)).iterator();
                 it2.hasNext();) {
                BayesNetNode child = (BayesNetNode) it2.next();

                double childProb = child.cpt.evaluate(fullAss);
                entries[i] *= childProb;
                if (DEBUG) {
                    System.out.println("  Child " + child.var +
                                       " gives prob " +
                                       childProb);
                }
            }
            sum += entries[i];
        }

        for (int i = 0; i < node.var.domain.size(); i++) {
            entries[i] /= sum;
            if (DEBUG) {
                System.out.println("Final: " + entries[i]);
            }
        }

        return entries;
    }

    /**
     * Select a value of a node's variable according to the probability
     * distribution specified by the given assignment.
     *
     * If useOnlyParents is specified, then the sampling is performed based
```

```
     * solely on the distribution given by the parents' values; otherwise it
     * is performed based on the full Markov blanket. So setting this false is
     * useful when generating an initial sample, and true useful when sampling
     * given a full assignment.
     */
    public static Comparable sampleNode(BayesNetNode node,
                                        Assignment evidence,
                                        boolean useOnlyParents,
                                        Map childMap) {
        double[] dist;
        if (useOnlyParents) {
            dist = evaluateNodeGivenParents(node, evidence);
        } else {
            dist = evaluateNodeGivenMarkovBlanket(node, evidence, childMap);
        }

        int r = Random.sampleCompleteProbDist(dist);
        return node.var.domain.getValue(r);
    }
}
```

```
import edu.mit.six825.bn.functiontable.*;
import edu.mit.six825.bn.bayesnet.*;
import edu.mit.six825.bn.inputs.*;

import java.util.*;

/**
 * A wrapper that lets us run any given solver in any given
 * configuration against any of the specified queries. See below for
 * invocation syntax.
 *
 * @author nocturne
 */
public class RunTest {

    public static Solver PickSolver (String ss) {
        if (ss.equals("enum")) {
            return(new EnumerationSolver());
        } else if (ss.equals("ve")) {
            return(new VESolver());
        } else if (ss.equals("lw")) {
            return(new LWSolver());
        } else if (ss.equals("gibbs")) {
            return(new GibbsSolver());
        } else {
            throw new IllegalArgumentException("Unknown solver type " + ss);
        }
    }

    public static VESolver.ElimOrderer PickElim (String ss) {
        if (ss.equals("dumb")) {
            System.out.println("Using dumb orderer.");
            return((new VESolver.DumbOrder()));
        } else if (ss.equals("random")) {
            System.out.println("Using random orderer.");
            return((new VESolver.RandomOrder()));
        } else if (ss.equals("greedy")) {
            System.out.println("Using greedy orderer.");
            return((new VESolver.GreedyOrder()));
        } else {
            throw new IllegalArgumentException("Unknown elim type " + ss);
        }
    }

    /**
     * Intended argument syntax:
     *     --solver ( enum | ve | lw )
     *     --elim   ( dumb | random | greedy )
     *     --weight (num)
     *     --samples (num)
     *     --discard (prefixlen)
     *     --randorder
     *     ( 1.(123) | 2.(1234) )
     *
     * specifying an elimination orderer is only relevant if you're
     * using the VE solver.
     * specifying a required weight is only relevant for LW solving.
     * number of samples, discard prefix len, and random ordering are
     * only relevant to Gibbs sampling.
     *
     * Sample args: "--solver ve --elim dumb 2.2"
     *              "--solver lw 1.3 --weight 100"
     *              "--solver gibbs samples 1000 --discard 500 --randorder"
     *
     */
```

```
public static void main(String[] args) {

    Solver      solver  = new EnumerationSolver();
    VESolver.ElimOrderer elimord = new VESolver.DumbOrder();
    String   whichquery = "1.1";
    double   requiredWeight = 100.0;
    int      requiredSamples = 1000;
    int      discardPrefixLen = 500;
    boolean  randOrder = false;

    for (int i=0; i < args.length; i++) {
        String arg = args[i];
        if (arg.equals("--solver")) {
            i++;
            if (i < args.length) {
                solver = PickSolver(args[i]);
            } else {
                throw new IllegalArgumentException("--solver requires"
                                                   + " an argument");
            }
        } else if (arg.equals("--elim")) {
            i++;
            if (i < args.length) {
                elimord = PickElim(args[i]);
            } else {
                throw new IllegalArgumentException("--elim requires"
                                                   + " an argument");
            }
        } else if (arg.equals("--weight")) {
            i++;
            if (i < args.length) {
                requiredWeight = Double.parseDouble(args[i]);
            } else {
                throw new IllegalArgumentException("--weight requires"
                                                   + " an argument");
            }
        } else if (arg.equals("--samples")) {
            i++;
            if (i < args.length) {
                requiredSamples = Integer.parseInt(args[i]);
            } else {
                throw new IllegalArgumentException("--weight requires"
                                                   + " an argument");
            }
        } else if (arg.equals("--discard")) {
            i++;
            if (i < args.length) {
                discardPrefixLen = Integer.parseInt(args[i]);
            } else {
                throw new IllegalArgumentException("--discard requires"
                                                   + " an argument");
            }
        } else if (arg.equals("--randorder")) {
            randOrder = true;
        } else {
            whichquery = arg;
        }
    }

    if (solver.getClass() == VESolver.class) {
        ((VESolver)solver).setOrderer(elimord);
    } else if (solver.getClass() == LWSolver.class) {
        ((LWSolver)solver).setRequiredWeight(requiredWeight);
    } else if (solver.getClass() == GibbsSolver.class) {
        ((GibbsSolver)solver).setRequiredSamples(requiredSamples);
```

```
        ((GibbsSolver)solver).setDiscardPrefixLen(discardPrefixLen);
        ((GibbsSolver)solver).randomFlipChoice(randOrder);
    }

    BayesNet   bnet;
    String []  evidencearr;
    String     query;

    if (whichquery.equals("1.1")) {
        bnet = Nets.getBurglary();
        evidencearr = new String[] {"JohnCalls", "true",
                                    "MaryCalls", "true"};
        query = "Burglary";
    } else if (whichquery.equals("1.2")) {
        bnet = Nets.getBurglary();
        evidencearr = new String[] {"Burglary",  "true",
                                    "JohnCalls", "true"};
        query = "Earthquake";
    } else if (whichquery.equals("1.3")) {
        bnet = Nets.getInsurance();
        evidencearr = new String[] {"Age", "Adolescent",
                                    "Airbag", "False",
                                    "Mileage", "TwentyThou"};
        query = "PropCost";
    } else if (whichquery.equals("2.1")) {
        bnet = Nets.getInsurance();
        evidencearr = new String[] {"Age", "Adolescent",
                                    "Airbag", "False",
                                    "MakeModel", "Luxury",
                                    "Mileage", "TwentyThou"};
        query = "PropCost";
    } else if (whichquery.equals("2.2")) {
        bnet = Nets.getInsurance();
        evidencearr = new String[] {"Age", "Adolescent",
                                    "Airbag", "False",
                                    "GoodStudent", "True",
                                    "Mileage", "TwentyThou"};
        query = "PropCost";
    } else if (whichquery.equals("2.3")) {
        bnet = Nets.getCarpo();
        evidencearr = new String[] {"N116", "0",
                                    "N41", "2",
                                    "N84", "1"};
        query = "N104";
    } else if (whichquery.equals("2.4")) {
        bnet = Nets.getCarpo();
        evidencearr = new String[] {"N116", "0",
                                    "N152", "1",
                                    "N43", "1"};
        query = "N73";
    } else {
        throw new IllegalArgumentException("Unknown query "
                                           + whichquery);
    }

    solver.setBayesNet(bnet);

    Assignment evidence;
    try {
        evidence =
            VESolver.ConstructEvidence(bnet.nodes, evidencearr);

        solver.setEvidence(evidence);
    }
    catch (IllegalArgumentException e) {
        System.out.println("ERROR: " + e);
        return;
    }

    BayesNetNode querynode;
    try {
        querynode = bnet.nodes.getNode(query);
    }
    catch (IllegalArgumentException e) {
        System.out.println("ERROR: " + e);
        return;
    }

    long starttime = System.currentTimeMillis();
    Function answer = solver.query(querynode);
    long endtime = System.currentTimeMillis();

    System.out.println("Query:    " + query);
    System.out.println("Evidence: " + evidence);
    System.out.println(answer);
    System.out.println("Solver time elapsed: " + (endtime-starttime) +
                       " milliseconds.");
  }
}
```

```
#!/usr/bin/perl -w

open (IN, "<timelog.ve.all");

%summary = ();

while (<IN>) {
  chomp;

  if (/^([^:]+\S)\s*:\s*([^\s:]+)\s*::\s*(\S.*)$/) {
    my $solver  = $1;
    my $query   = $2;
    my $datastr = $3;

    my @data = split(", ", $datastr);

    push @{$summary{$solver}{$query}}, @data;
  } else {
    chomp;
    die "Unknown time-log entry \"$_\"!\n";
  }
}

foreach my $solver (sort keys %summary) {
  foreach my $query (sort keys %{$summary{$solver}}) {
    my $varname  = "$solver.$query";
    $varname =~ s/ /_/g;
    $varname =~ s/-//g;
    $varname =~ s/\./_/g;

    my $filename = "./matlab/$varname.m";

    print "Data for solver $solver vs $query in $filename\n";
    open (DAT, ">$filename");
    print DAT "% Runtimes for solver $solver\n";
    print DAT "% using query $query\n";
    print DAT "times_$varname = [ ";
    print DAT join(",  ", &tidydata(@{$summary{$solver}{$query}}));
    print DAT " ];\n";

    if (($solver eq "ve --elim random") &&
        (($query eq "2.3") || ($query eq "2.4"))) {
      print DAT "hist(times_$varname, 10);\n";
      print DAT "xlabel('x = milliseconds');\n";
      print DAT "ylabel('Number of trials of length (x) milliseconds');\n";
    } else {
      print DAT "hist(log10(times_$varname), 35);\n";
      print DAT "xlabel('x = Log_{10}(milliseconds)');\n";
      print DAT "ylabel('Number of trials of length 10^x milliseconds ');\n";
    }
    print DAT "print -deps2 $varname";
    close (DAT);
  }
}

sub tidydata {
  my (@data) = @_;

  my $magic_time = 999999999999999;
  my $max_normal_time = 0;

  for my $n (@data) {
    if (($n != $magic_time) && ($n > $max_normal_time)) {
      $max_normal_time = $n;
    }
  }

  $new_magic_time = int(1.2*$max_normal_time);

  @data = map { if ($_ == $magic_time) {
                  $new_magic_time;
                } else {
                  $_;
                }
              } @data;

  return(sort { $a <=> $b } @data);
}
```

```
#!/usr/bin/perl -w

##
## Usage: ./GatherData.pl [--solver <solver>] [--runs N] [--loop]
##  --solver SOLVERSTRING : use solver SOLVERSTRING, e.g. "lw" or "ve" or
##       "ve --elim random"
##  --runs N : run solver N times on each query
##  --loop : run forever, logging output in batches of N
##
## e.g. ./GatherData.pl --runs 1 --solver "ve --elim greedy"
##

$JAVA = 'java -ea -Xmx512M -classpath .:./BN.jar';

my $Datestamp;

sub runtest {
  my ($solver, $query) = @_;

  # If java blows out, it won't output an elapsed-time, and this value
  # will be returned.
  my $time = 999999999999999;

  open (S, "$JAVA RunTest --solver $solver $query 2>&1 |");

  my $output = "";
  while (<S>) {
    $output .= $_;
    if (/Solver time elapsed: (\d+) milliseconds/) {
      $time = $1;
    }
  }
  close(S);

  return($time, $output);
}


sub dostuff {
  my ($solverref, $queriesref, $numberOfRuns) = @_;

  my @solversToTry = @{$solverref};
  my @queriesToRun = @{$queriesref};

  my %summary = ();

  my $logfile = "./runlog.$Datestamp";

  print "Logging solver output in $logfile\n";

  open (RUNLOG, ">>$logfile");
  foreach my $solver (@solversToTry) {
    foreach my $query (@queriesToRun) {
      my @data = ();
      for (my $i=0; $i < $numberOfRuns; $i++) {
        my ($time, $output) = &runtest($solver, $query);
        push @data, $time;
        print RUNLOG $output;

        print STDERR ".*; # to reassure the user that progress occurs
      }

      $summary{$solver}{$query} = \@data;
    }
  }
  print STDERR "\n";
  return %summary;
}

sub tellabout {
  my(%summary) = @_;

  my $filename = "./timelog.$Datestamp";

  print "Archiving these *timing* results in file $filename\n\n";

  open(LOG, ">>$filename");
  foreach my $solver (sort keys %summary) {
    foreach my $query (sort keys %{$summary{$solver}}) {
      print "$solver : $query :: ",
        join(", ", @{$summary{$solver}{$query}}), "\n";
      print LOG "$solver : $query :: ",
        join(", ", @{$summary{$solver}{$query}}), "\n";
    }
  }
  close(LOG);
}

######################################################################

my $runforever = 0;
#### Can in theory run on more than one solver per batch, but for now
#### not doing it.
my @solversToTry = ("ve --elim random");
my @queriesToRun = ("2.1", "2.2", "2.3", "2.4");
my $numberOfRuns = 10;

while (@ARGV) {
  my $arg = shift @ARGV;
  if ($arg =~ /--loop/i) {
    $runforever = 1;
  } elsif ($arg =~ /--solver/i) {
    if ($#ARGV != -1) {
      @solversToTry = shift @ARGV;
    } else {
      die "--solver takes an argument!\n";
    }
  } elsif ($arg =~ /--runs/i) {
    if ($#ARGV != -1) {
      $numberOfRuns = shift @ARGV;
    } else {
      die "--runs takes an argument!\n";
    }
  } else {
    die "Unknown argument $arg\n";
  }
}

my $haverunonce = 0;
while ($runforever || !$haverunonce) {
  $Datestamp=`date '+%Y%m%d.%H%M'`;
  chomp $Datestamp;

  my %summ = &dostuff(\@solversToTry, \@queriesToRun, $numberOfRuns);
  &tellabout(%summ);
  $haverunonce = 1;
}
```

```
import math, time, os
from jarray import zeros, array
from Gnuplot import Gnuplot, Data

from java.lang import Comparable, String

from edu.mit.six825.bn.functiontable import *
from edu.mit.six825.bn.bayesnet import *
from edu.mit.six825.bn.inputs import *
from techniques.utils import *
import VESolver
import LWSolver
import GibbsSolver

# Some usful constants

TRUE = ComparableBoolean.TRUE
FALSE = ComparableBoolean.FALSE
burglary = Nets.getBurglary()
insurance = Nets.getInsurance()
carpo = Nets.getCarpo()

# Global gnuplot instance
global g
g = Gnuplot(debug = 1)

# Utility functions
def enumerate(lst):
    return [(i,lst[i]) for i in range(len(lst))]


# mean/stddev, from
# http://www.atnf.csiro.au/people/Enno.Middelberg/python/avg.p
def stat(numbers):
    sum=0
    # Calculate avg, sx and sigmax
    if len(numbers)>1:
        for i in numbers:
            sum=sum+i
            # Store total sum for later
            total=sum
        avg=sum/len(numbers)
        sum=0
        for i in numbers:
            sum=sum+(i-avg)**2
        sx=math.sqrt(sum/(len(numbers)-1))
        sigmax=math.sqrt(sum/len(numbers))
    else:
        avg=numbers[0]
        sx=0
        sigmax=0
        sum=numbers[0]
    # Calculate mean
    numbers.sort()
    if (len(numbers) % 2)==1:
        mean=numbers[(len(numbers)-1)/2]
    else:
        mean=(numbers[(len(numbers)/2)-1]+numbers[(len(numbers)/2)])/2
    # Return results
    return avg, sx, sigmax, total, mean

def avg(nums):
    return stat(nums)[0]
def stddev(nums):
    return stat(nums)[2]


# Gnuplot exporters

def latex(f):
    g('set terminal push')
    #g('set terminal latex 10')
    g('set terminal epslatex color "default" 10')
    g('set format xy "$%g$"')
    g.set_string('output', f.replace(".tex",".eps"))
    #g('set size 1,1')
    #g('set size 3/5, 3/5')
    g.refresh()
    g('set terminal pop')
    g.set_string('output')
    # Damnit, fix the path in the output file
    time.sleep(3)
    os.system("echo 's,{%(b)s},{%(b)s.eps},\nw' | ed %(e)s" %
              {'b': f.replace(".tex",""), 'e': f})

def png(f):
    g('set terminal push')
    g('set terminal png')
    g.set_string('output', f)
    g.refresh()
    g('set terminal pop')
    g.set_string('output')

# Convert strings to Java strings
def stringify(x):
    if type(x) == type("foo"):
        return String(x)
    else:
        return x


# Represents all the parameters to a query

class Query:
    def __init__(self, bn, queryName, evidenceList):
        self.bn = bn
        self.queryVar = bn.nodes.getNode(queryName)
        functionVars = [bn.nodes.getNode(x[0]).var for x in evidenceList]

        evidenceVals = [stringify(x[1]) for x in evidenceList]
        self.evidence = Assignment(
            FunctionVariableSet(array(functionVars, FunctionVariable)),
            array(evidenceVals, Comparable))

def queryResToDict(res):
    d = {}
    assert(res.variables.size() == 1)
    var = res.variables.getVariable(0)
    domain = var.domain
    for i in range(domain.size()):
        x = domain.getValue(i)
        as = Assignment(
            FunctionVariableSet(array([var], FunctionVariable)),
            array([stringify(x)], Comparable))
        d[str(x)]= res.evaluate(as)

    return d

def runTest(solverType, query,
            orderer=VESolver.GreedyOrder(),
```

```
            weight=10.0,
            samples=10000,
            discard=5000,
            randomOrder=0):
    solver = solverType(query.bn)
    solver.setEvidence(query.evidence)
    if solverType == VESolver:
        solver.setOrderer(orderer)
    elif solverType == LWSolver:
        solver.setRequiredWeight(weight)
    elif solverType == GibbsSolver:
        solver.setRequiredSamples(samples)
        solver.setDiscardPrefixLen(discard)
        solver.randomFlipChoice(randomOrder)

    return queryResToDict(solver.query(query.queryVar))

def kullbackLeibler(exactDist, approxDist):
    totalDiv = 0.0
    for x in exactDist.keys():
        if approxDist[x] == 0:          # Avoid division by zero
            approx = 0.000001
        else:
            approx = approxDist[x]
        totalDiv += (exactDist[x] *
                     math.log(exactDist[x] / approx) / math.log(2))
    return totalDiv


def plotGibbsBurnin(query, filename, titleinfo):
    burninFractions = [0.1 * x for x in range(0,9)]
    sampleCounts = [10000]
    runs = 5

    exactDist = runTest(VESolver, query)

    g('set data style errorlines')
    g('set key left Left reverse')
    g('set lmargin 5')
    g('set ylabel "KL divergence" 1.5, 0')
    g('set xlabel "Fraction of samples discarded" 0, .5')
    g('set title "Divergence vs burnin period --- \%s"' % titleinfo)
    gpData = []
    for sampleCount in sampleCounts:
        series = []
        for burninFraction in burninFractions:
            divergences = [kullbackLeibler(exactDist, runTest(
                GibbsSolver, query,
                samples = int(sampleCount),#*(1+burninFraction)),
                discard = int(sampleCount*
                              burninFraction)))
                           for run in range(runs)]
            divMean = avg(divergences)
            divSD = stddev(divergences)
            series.append([burninFraction, divMean, divSD])
        gpData.append(Data(series, title=str(sampleCount) + " samples"))
    g.plot(*gpData)
    png(filename + ".png")
    latex(filename + ".tex")

def plotLWQuality(query, filename, titleinfo):
    #weights = [0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000]
    weights = [10, 20, 40, 100, 200, 400, 1000, 2000, 4000, 10000]
    runs = 10

    exactDist = runTest(VESolver, query)

    g('set size 0.75,0.75')
    g('set logscale x 10')
    g('set data style linespoints')
    g('set key left Left reverse')
    g('set lmargin 5')
    g('set ylabel "KL divergence" 1.5, 0')
    g('set xlabel "Samples performed" 0, .5')
    g('set title "LW quality vs samples --- %s"' % titleinfo)
    g('set yrange [0:*]')
    g('set xrange [10:10000]')

    gpData = []
    aggData = [[] for x in weights]

    for run in range(runs):
        series = []
        for n,weight in enumerate(weights):
            div = kullbackLeibler(exactDist,
                                  runTest(LWSolver, query, weight = weight))
            series.append([weight, div])
            aggData[n].append(div)
        gpData.append(Data(series, title="Run " + str(run)))
    g.plot(*gpData)
    png(filename + ".png")
    latex(filename + ".tex")

    g('set data style errorlines')
    series = []
    for x,weight in zip(aggData,weights):
        series.append([weight, avg(x), stddev(x)])
    g.plot(Data(series))
    png(filename + "-agg.png")
    latex(filename + "-agg.tex")

def plotGibbsQuality(query, filename, titleinfo):
    #weights = [0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000]
    weights = [100, 200, 400, 1000, 2000, 4000, 10000, 20000, 40000, 100000, 200000, 4
00000, 1000000]
    runs = 10
    dRate = 0.4

    exactDist = runTest(VESolver, query)

    g('set size 0.75,0.75')
    g('set logscale x 10')
    g('set data style linespoints')
    g('set key left Left reverse')
    g('set lmargin 5')
    g('set ylabel "KL divergence" 1.5, 0')
    g('set xlabel "Samples performed" 0, .5')
    g('set title "Gibbs quality vs samples --- %s"' % titleinfo)
    g('set yrange [0:*]')
    g('set xrange [100:1000000]')
    gpData = []
    aggData = [[] for x in weights]

    for run in range(runs):
        series = []
        for n,weight in enumerate(weights):
            div = kullbackLeibler(exactDist,
                                  runTest(GibbsSolver, query,
                                          samples = weight,
                                          discard = int(dRate * weight)))
```

```
            series.append([weight, div])
            aggData[n].append(div)
        gpData.append(Data(series, title="Run " + str(run)))
    g.plot(*gpData)
    png(filename + ".png")
    latex(filename + ".tex")

    g('set data style errorlines')
    series = []
    for x,weight in zip(aggData,weights):
        series.append([weight, avg(x), stddev(x)])
    g.plot(Data(series))
    png(filename + "-agg.png")
    latex(filename + "-agg.tex")


query11 = Query(burglary, "Burglary",
                [("JohnCalls", TRUE), ("MaryCalls", TRUE)])
query12 = Query(burglary, "Earthquake",
                [("Burglary", TRUE), ("JohnCalls", TRUE)])
query13 = Query(insurance, "PropCost",
                [("Age", "Adolescent"), ("Airbag", "False"),
                 ("Mileage", "TwentyThou")])
query21 = Query(insurance, "PropCost",
                [("Age", "Adolescent"), ("Airbag", "False"),
                 ("MakeModel", "Luxury"), ("Mileage", "TwentyThou")])
query22 = Query(insurance, "PropCost",
                [("Age", "Adolescent"), ("Airbag", "False"),
                 ("GoodStudent", "True"), ("Mileage", "TwentyThou")])
query23 = Query(carpo, "N104",
                [("N116", "0"), ("N41", "2"), ("N84", "1")])
query24 = Query(carpo, "N73",
                [("N116", "0"), ("N152","1"), ("N43", "1")])


# exact = runTest(VESolver, query11, orderer=VESolver.GreedyOrder())
# lw = runTest(LWSolver, query11, weight=5)

# print "Exact:", exact
# print "LW:", lw
# print "KL divergence: ", kullbackLeibler(exact, lw)
#print runTest(GibbsSolver, query13)

# plotGibbsBurnin(query11, "figures/gibbs-burnin-query11",
#                 "P(Burglary | John,Mary)")
# plotGibbsBurnin(query12, "figures/gibbs-burnin-query12",
#                 "P(Earthquake | Burglary,John)")
# plotGibbsBurnin(query13, "figures/gibbs-burnin-query13",
#                 "P(PropCost | Age=Adolescent, Airbag=False, Mileage=TwentyThou)")
# plotGibbsBurnin(query21, "figures/gibbs-burnin-query21",
#                 "P(PropCost | Age=Adolescent, Airbag=False, MakeModel=Luxury, Mileag
e=TwentyThou)")
# plotGibbsBurnin(query22, "figures/gibbs-burnin-query22",
#                 "P(PropCost | Age=Adolescent, Airbag=False, GoodStudent=True, Mileag
e=TwentyThou)")
# plotGibbsBurnin(query23, "figures/gibbs-burnin-query23",
#                 "P(N104 | N116=0, N41=2, N84=1)")
# plotGibbsBurnin(query24, "figures/gibbs-burnin-query24",
#                 "P(N73 | N116=0, N152=1, N43=1)")


plotLWQuality(query21, "figures/lw-quality-query21",
              "Query 1")
plotLWQuality(query22, "figures/lw-quality-query22",
              "Query 2")
plotLWQuality(query23, "figures/lw-quality-query23",
              "Query 3")
plotLWQuality(query24, "figures/lw-quality-query24",
              "Query 4")

plotGibbsQuality(query21, "figures/gibbs-quality-query21",
              "Query 1")
plotGibbsQuality(query22, "figures/gibbs-quality-query22",
              "Query 2")
plotGibbsQuality(query23, "figures/gibbs-quality-query23",
              "Query 3")
plotGibbsQuality(query24, "figures/gibbs-quality-query24",
              "Query 4")
```