Adaptive Parallel AND/OR Graph Importance Sampling™

By Gary Steelman and Mathew Gray

Introduction

The goal of this project was to develop and test an implementation of AND/OR importance sampling.
In this report we outline our implementation of Adaptive Parallel AND/OR Graph Importance Sampling™ (APAOGIS) and test its accuracy and speed versus traditional w-cutset importance sampling (WCIS). We include our results as well as some improvements which could increase performance and accuracy.
AND/OR (AO) sampling aims to dramatically reduce the space complexity when computing the partition function of a large probabilistic graphical model (PGM). We further improved on AO sampling by inducing a junction graph from the original network. Thus, each sample generated can be decomposed into multiple virtual samples based on the context of a cluster in the junction graph, increasing the worth of each sample.
This in theory leads to more accurate results than general importance sampling.

Hypothesis

Our hypothesis is that by utilizing parallelism we can drastically reduce computation time for the partition function approximation given similar numbers of samples.

Implementation

We implemented Adaptive Parallel AND/OR Graph Importance Sampling™ using Java 7 standard libraries and ran our tests with a Python script.

Pseudocode

```
1. Build a pseudo tree (PT) from the initial BN.
    a. Construct the interaction graph (IG) for BN by creating an undirected,
       moral version of BN.
    b. Perform a DFS traversal on the IG and record the variables into the PT
       as it is traversed.
2. Create a proposal distribution Q.
    a. Set Q to the uniform distribution.
3. Generate samples from Q.
    a. Create ordering O along the topological linearization of the PT.
    b. Sample from Q along O.
4. Build a junction graph (JG) based on the PT.
    a. Perform a DFS traversal on the PT.
        i. Add current PT node's variable to the context of the current
           node in the JG.
        ii. Associate each function from BN with exactly one cluster.
        iii. Associate each Q with exactly one cluster.
        iv. Multiply functions in clusters to generate initial sparse table for a
            cluster.
        v. Multiply sparse table weights by counts of samples for each
           cluster.
        vi. Divide sparse table weights by probability of sample from Q for
            the cluster.
5. Compute the partition function approximation.
    a. Starting at the leaf nodes in the JG, compute upward messages in
       parallel.
        i. Compute message to parent nodes by averaging out variables
           in a cluster which are not in its parent cluster's context.
        ii. Send message to parent cluster.
        iii. If the parent cluster has received all its messages, recurse and
             compute the message to its parent. If not, exit computation
             tree and return to the thread pool.
    b. Average out at root in the JG to obtain the partition function
       approximation.
6. Update the Q using the weights of each sample.
    a. Begin with a prior of the learning rate for each component probability
```

in Q.

b. Sum the weights of all samples.
c. For each sample, add to each component probability the value obtained in ii where the sample agrees with the component probability.
d. Divide each component probability by the total weight from ii.
e. Normalize each component probability.

Theorem 1

The total number of threads needed after constructing the JG will be no more than the number of leaf nodes.

The algorithm utilizes the final thread which sent its message to its parent to compute the outgoing message at the parent. If a thread reaches a node which has not received all its incoming messages, the thread returns to the thread pool and awaits a new task.

Features

- Parallel computation of the partition function approximation once the JG is constructed.
- Uses Java standard libraries for cross compatibility.
- Command line interface for easy configuration.
    - Can configure the number of samples to use.
    - Can configure the number of threads to use.
- Works on standard UAI file formats.

Results

Configuration and Testing Routine

The configuration for our program to produce the results in this section is:

- Intel Core i7 vPro.
- 4GB 1333 DDR3 RAM.
- Fixed w-cutset importance sampling size of 4.
- Fixed thread pool size of 16.
- Fixed learning rate of 1.

To test the accuracy and speed of our algorithm, we ran multiple test cases using various networks of various sizes with various numbers of samples. We tested our algorithm against w-cutset importance sampling (WCIS) because gives accurate estimations for small-ish networks and typically runs faster for large-ish networks.

Observations

The main observations from our results:

1. There is a linear increase in computation time proportional to the number of

samples generated.

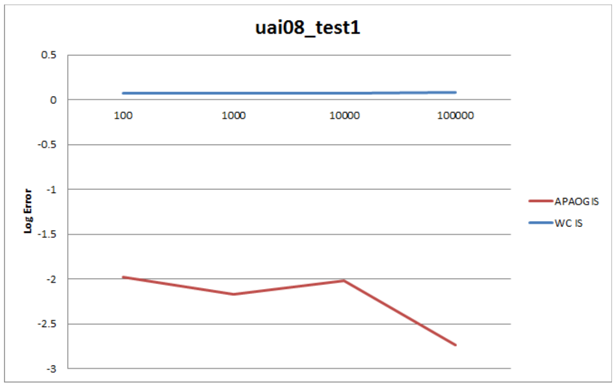
2. There is reduced error given more samples.
3. Without using a learning rate, the aggressiveness gives skewed results.
4. Higher learning rates (more aggressive) are recommended when the network's distributions are more disparate from the proposal distribution.
5. For large networks, APAOGIS outperforms WCIS. We also tested on small networks, which showed better results using WCIS.

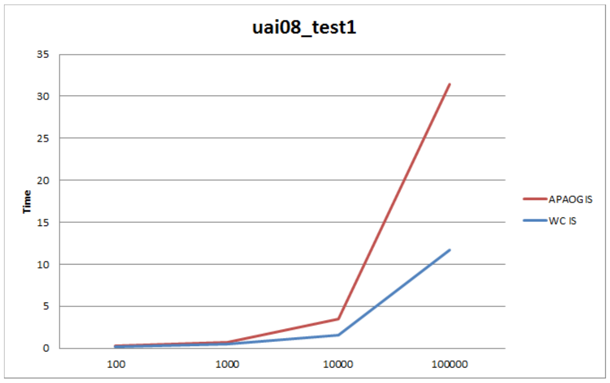
Theorem 2

APAOGIS returns a lower bound, α, on the partition function if:
$\alpha = min(z_i)$ *where z is the set of approximations from running APAOGIS*.

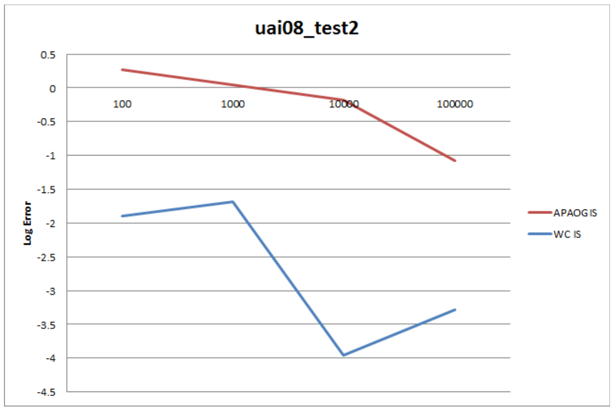
Graphs

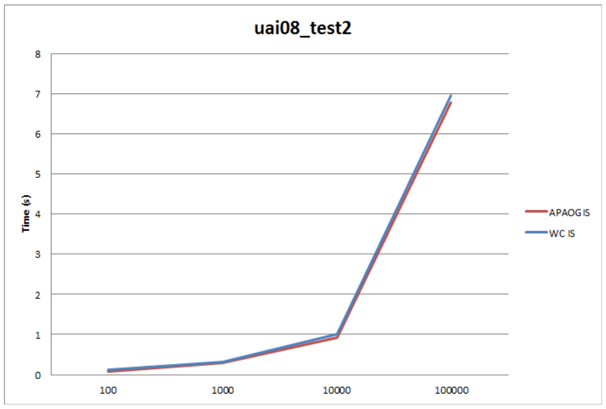
Below are graphs visualizing our results. Tables containing all test values can be found in the appendix.



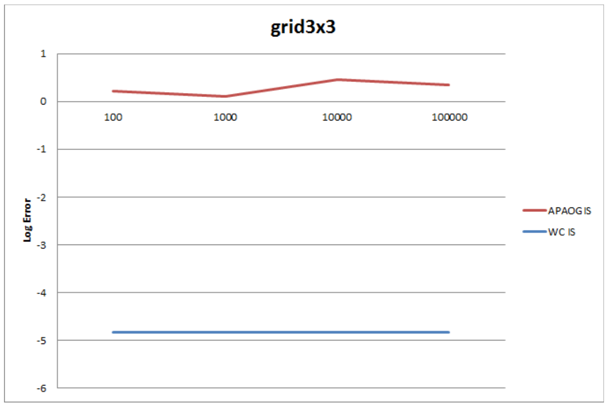


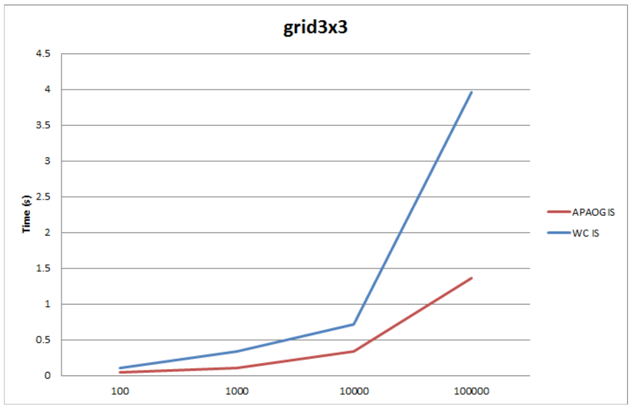
For the uai08_test1 Bayesian network case we see the log error for APAOGIS is orders of magnitude smaller than WCIS. However, computation time is much greater due to the high width of the network.





For the uai08_test2 Bayesian network case we see APAOGIS' error is larger than WCIS and the computation times are almost exactly the same. This is because the width of the network was small.





For the grid3x3 Markov network case we see APAOGIS' error is larger than WCIS but the computation time is much faster.

Improvements/Optimizations/Future Work

As with any project there are optimizations, features, and improvements which could be made. We identified the following as candidate tasks:

- Building the JG in parallel.
    - We attempted this, but due to time constraints we fell back to the original serial implementation.
- Building more efficient sparse tables using tries or using an open source implementation.
- Support networks which contain determinism (hard clauses).
- Use a Q which supports conditional independence, perhaps created from Iterative Join Graph Propagation.

Conclusion

We implemented AO importance sampling as APAOGIS. It improves performance by parallelizing decomposable computations in the JG. The algorithm presented is easily configurable and cross-platform and works on UAI standard files. APAOGIS is the superior algorithm for large networks due to its speed and improved accuracy. However, for small networks, WCIS generally completes faster. If the network width is smaller than the WCIS cutset bound, then WCIS performs admirably, but on large networks WCIS loses accuracy due to pruning a large number of nodes.

Appendix

The table of all result values can be found here: http://goo.gl/pqtu3

References

http://www.hlt.utdallas.edu/~vgogate/pgm/homeworks/homework3.5.pdf
http://www.hlt.utdallas.edu/~vgogate/papers/aij09-aos.pdf