COL776 - Assignment 1 (Part A)

Akshay Kumar Gupta
2013CS50275

Implementation Language: C++

Q1. Input from the user are the numbers n (number of nodes) and k. To generate a random Bayesian network, nodes $1 \dots n$ are considered in topological order. For each node i, a number u between 1 and k is chosen uniformly at random. Now the list $(i+1 \dots n)$ is taken and shuffled uniformly. The first u elements of the shuffled list are chosen as children of node i. This guarantees that the u children are picked uniformly. Shuffling a list takes time proportional to the length of the list so time taken for network generation is $\mathcal{O}(n^2)$.

Q2. Summary of the paper "Bayes-Ball: The Rational Pastime" by Ross D. Schachter :

Problem Statement

Given a Bayesian network $\mathcal{G}$, a set of observed nodes K and a set of query nodes J, find the set of nodes which are d-separated from J (irrelevant nodes), the set of nodes whose conditional probability distributions are required to compute $\Pr\{X_J|X_K\}$ (requisite probability nodes), and the subset of observed nodes whose states are required to compute $\Pr\{X_J|X_K\}$ (requisite observation nodes).

Algorithm

The algorithm is known as the Bayes' Ball algorithm. The basic idea of the algorithm is to send a bouncing ball to visit nodes in the network, starting from the nodes in J. The ball can either pass through a node, bounce back from a node, or get blocked. From basic knowledge of d-separation, it can be observed that:

- An unobserved node will pass the ball through and also bounce the ball back from its children.
- An observed node will bounce the ball back from parents and block the ball from children.

One further insight is that an edge need only be visited in the same direction once. This is accomplished by marking a node on either top or bottom when visited. Another advantage of this is that whether a node is marked on top or bottom (or both) will give us additional information about whether a node is a requisite probability node or a requisite observation node. The full algorithm is as follows:

Algorithm 1 Bayes' Ball Algorithm

```
1:  procedure BayesBall(G, K, J)
2:      ∀ i ∈ J, add i to frontier F as if i was visited from a child
3:      while F is not empty do
4:          Pop a node j from F
5:          Mark j as visited
6:          if j ∉ K and visit to j is from a child then
7:              if top of j is not marked then
8:                  Mark j on top
9:                  Add parents of j to F
10:             end if
11:             if bottom of j is not marked then
12:                 Mark j on bottom
```

```
            Add children of j to F
        end if
    end if
    if visit to j is from a parent then
        if j ∈ K and top of j is not marked then
            Mark j on top
            Add parents of j to F
        end if
        if j ∉ K and bottom of j is not marked then
            Mark j on bottom
            Add children of j to F
        end if
    end if
  end while
end procedure
```

Results

1. The set of irrelevant nodes, $N_i(J|K)$, are those nodes not marked on bottom. This includes all the observed nodes as well as the nodes that are not visited.
2. The set of requisite probability nodes, $N_p(J|K)$, are those nodes marked on top.
3. The set of requisite observation nodes, $N_e(J|K)$, are those nodes in K marked as visited.

Running Time

Because each edge is visited at most twice, the running time of the algorithm is linear in the size of visited part of the graph. In the worst case, this is $\mathcal{O}(m+n)$.

Q3. and Q4. The Bayes' ball algorithm is implemented to compute the requisite probability nodes, the requisite observation nodes, and the nodes d-separated from the query nodes . A frontier of nodes to be visited is maintained and vectors are maintained which record which nodes are visited, marked on bottom and marked on top. At each step a node is popped from the frontier, processed and its parents/children are added to the frontier in accordance with the algorithm. Algorithm terminates when the frontier becomes empty. Running time of the algorithm is linear in the size of the visited part of the graph, which is $\mathcal{O}(m+n)$ in the worst case, where m is the number of directed edges.

Q5. Task is to find all sets of d-separated nodes, given a set of observed nodes.

Implemented Algorithm:
We maintain an $n \times n$ boolean matrix A in which $a_{ij} = true$ iff the i^{th} and j^{th} nodes are d-connected. We consider the nodes $1 \dots n$ in topological order. We run the Bayes' Ball algorithm with the first unobserved node in this order as the query node. We obtain a set of nodes S that are reachable from this node. All the nodes in S are d-connected to each other (the broad reason being this node only has outgoing edges). Hence we set $a_{i,j} = true \ \forall \ i, j \in S$. We also mark all the nodes in S. We again run Bayes' Ball on the next node in order which is not marked, and so on. In the end the entries in A which are marked $false$ are all the pairs of nodes which are d-separated. The running time benefit we get from this algorithm is that the number of calls to the Bayes' ball algorithm is at most equal to the number of nodes that don't have any unobserved parents. The time complexity of the algorithm is (not sure).

Unimplemented algorithm:
The above algorithm suffers from the fact that in different calls to the Bayes' Ball algorithm, the same paths can be explored multiple times. The following algorithm addresses this problem.

1. When running the Bayes' Ball algorithm, mark the connections in a unidirectional manner, so nodes occurring earlier in the path are connected to each node occurring later in the path and not the other way round.

2. In subsequent calls to Bayes' Ball, if a node is visited which was visited in an earlier call to Bayes' Ball, this node is not added to the frontier. However, connections are added between all other nodes which are encountered in this run of Bayes' Ball and all the nodes that are connected to this node.

This algorithm will only visit edges at most twice, and pairs of nodes in A will only get marked as *true* once. Hence the running time of this algorithm is $\mathcal{O}(m + n^2)$.

Discussed assignment with : Barun Patra, Haroun Habeeb, Kabir Chhabra