AI - CSL 302
LAB - 4 Report

Eeshaan Sharma
2015CSB1011
Subhashish Sharma
2014CSB1036

April 2018

1 Planning - Shakey The Robot

1.1 State Formulation

There are 3 propositions in Shakey's world and to exhaustively list out the propositions that describe the current state, a boolean array has been constructed of size - N*B + 2*N + B + 1 (explained later). A value of 0 indicates that the proposition described by that location in the boolean array is false and a value of 1 indicates that the proposition is true in the current state. Note that here N refers to Number of rooms and B refers to Number of boxes. The propositions are encapsulated together in the boolean array in the following manner

1. **at(object, location)** - This tells the room in which an object is present. Object can be a box or Shakey, the robot.
    - **Object is box** - First N*B entries in the boolean array (0 to (N*B) - 1) for propositions at(b,r). If State[(N*b + r - 1)] = true, means that box b is present in room r.
    - **Object is Shakey** - Next N entries (N*B to (N*B + N) - 1) for propositions at(shakey, r). If State[(N*B + r - 1)] = true, means that Shakey is present in room r.
2. **on(shakey, b)** - Tells whether Shakey is on b. Here b could be floor or box.
    - **b is box** - Next B entries in the boolean array (N*B + N to (N*B + N + B) - 1) for propositions on(shakey, b). If State[(N*B + N + b - 1)] = true, means that Shakey is standing on box b.

- **b is floor** - Next single entry (N*B + N + B) for proposition on(shakey, floor). If State[(N*B + N + B)] = true, means that Shakey is standing on the floor.

3. **switchon(light)** - This tells if light is switched on or not. Next N entries (N*B + N + B + 1 to (N*B + N + B + 1 + N) - 1) for propositions switchon(l). If State[(N*B + N + B + 1 + l - 1)] = true, means that the light in room l is switched on in the current state.

Using the above encapsulation in a boolean array of size (N*B + N + B + 1 + N), the propositions of the world have been exhaustively listed to describe the current state.

1.2 Forward (Progression) Planner using Breadth First Search

1.2.1 Introduction

Forward planner is a deterministic planner which solves a problem by producing a plan or a sequence of actions to achieve a goal state from a given starting state. Forward planning is one of the simplest planning strategies to treat the planning problem as a path planning problem in the state-space graph. The input to the planner is an initial world description, a specification of the actions available to the agent and a goal description. The specification of the actions includes their preconditions and their effects.

In the state space graph which is actually a tree, nodes are states, and arcs correspond to actions from one state to another. The arcs coming out of a state s correspond to all of the legal actions that can be carried out in that state i.e. actions whose preconditions holds in state s. Once we have a state with a valid action, the new state can be generated by adding the positive effects(predicates) of the action and removing the negative effects(predicates) of the action from that state.

A forward planner searches the state-space graph from the initial state looking for a state that satisfies the goal description. The search technique used for carrying out forward planning in this part of the lab is Breadth First Search (BFS). In BFS we search through the tree in such a way that we explore all nodes at a particular depth before moving onto the next level. As a result the Goal state which we get from this search is optimal(considering path length 1 for each edge). But this strategy also explores many unwanted states due to which the time complexity increases exponentially with depth.

1.2.2 Observations

For 1.txt or 2.txt

1. No. of nodes expanded = 15
2. Time taken = 0.00083
3. Solution length = 5

For 3.txt or 4.txt

1. No. of nodes expanded = 95
2. Time taken = 0.0126
3. Solution length = 5

For 5.txt or 6.txt

1. No. of nodes expanded = 96
2. Time taken = 5
3. Solution length = 0.0074

For 7.txt or 8.txt

1. No. of nodes expanded = 142
2. Time taken = 0.0067
3. Solution length = 8

For 9.txt or 10.txt

1. No. of nodes expanded = 555
2. Time taken = 0.034
3. Solution length = 9

1.3 Goal Stack Planner

1.3.1 Introduction

Goal stack planning combines the advantages of forward search and backward search. The small branching factor advantage of backward search and the soundness of forward search. An action is added to the plan only if all its preconditions are satisfied. If any precondition is not satisfied, then we add a relevant action to satisfy that precondition and push its preconditions in turn on the stack and repeat the same process.

Since there are more than one relevant actions for some predicates, we have to choose the relevant action wisely. The time taken by goal stack planning is less as compared to progression search using BFS because Goal Stack planning tries to choose only the relevant action starting from the goal state. But it does not always guarantee the optimal path. It only gives the optimal solution when the predicates are pushed in the stack in a serializable order, which might not always be possible (Sussman's Anomaly). Since the order is not known beforehand, it doesn't always give the optimal solution.

1.3.2 Observations

For 1.txt or 2.txt

1. Time taken = 5
2. Solution length = 0.00068

For 3.txt or 4.txt

1. Time taken = 0.00054
2. Solution length = 5

For 5.txt or 6.txt

1. Time taken = 0.0008
2. Solution length = 5

For 7.txt or 8.txt

1. Time taken = 0.0006
2. Solution length = 11

For 9.txt or 10.txt

1. Time taken = 0.0008
2. Solution length = 9

1.4 Conclusion

On comparing the search time of BFS and Goal Stack Planning the following graph is obtained -

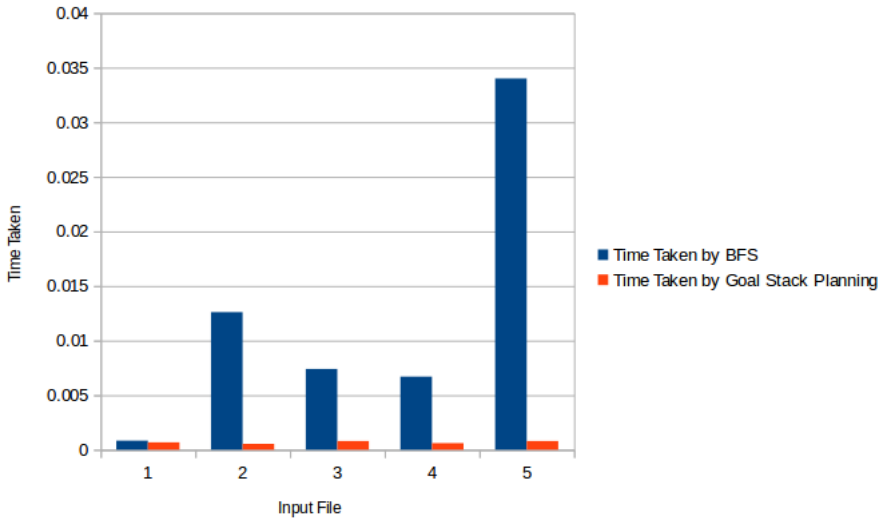

Figure 1: Time Taken by BFS vs Goal Stack Planning

The above graph shows that although forward planning is complete and optimal but it takes a large amount of time as it expands a large number of irrelevant nodes.

Goal Stack Planning, on the other hand, is not complete as it might not always find a solution even if one exists (although in our case it solves all the .txt files) and also Goal Stack Planning is not optimal (as explained above-Sussman's Anomaly) and can be seen in the solution lengths of the files 7.txt or 8.txt. Progression search using BFS finds a solution involving 8 actions (optimal) where as solution found by Goal Stack Planning takes 11 actions (suboptimal). But since it tries to always choose the relevant action starting from the goal state, whenever it finds the solution, it takes very less time compared to other search strategies.

2 Bayesian Networks - Gibbs Sampler

2.1 Introduction

Sampling from a Bayesian network is an approximate method of inferencing from the true distribution.

Gibbs Sampling a one of the major types of sampling methods which ensures that the sample conditions on the upstream as well as the downstream evidences.

The procedure to obtain a Gibbs sample from a given Bayesian net is as follows

- Fix the evidence variable
- Randomly initialize all other variables
- Repeat:
  - Choose a non evidence variable X
  - Resample X from all other variables from P (X / Parents(X))

The main property that holds true for Gibbs sampling is that given an infinite number of sampling iterations, the sample obtained is from the true distribution.

2.2 Formalizing The Problem

The given dataset is Adult Income Dataset consisting of 14 different variables i.e. Age, Workclass, education, education-num, marital status, occupation, relationships, race, sex, capital-gain, capital-loss, hours-per-week, native-country and salary.

The given implementation of generating Bayesian network in chowliu.py, generates all the edges of the Bayesian network, marginal distribution for all the values in a variable and the joint distribution of all pairs of variables. We had to tweak the code to also generate the joint distribution of all triplets of variables. This was needed for the calculation of conditional probability while sampling since the maximum number of parents in the Bayesian network formed was 2.

Different Gibbs sample set were generated by changing the number of sampling iterations and number of samples generated.

The test set has 7561 samples out of which 7299 samples are unique. The probability of occurrence of a test sample in the test set is taken as the ground truth.

Error is defined as the sum of the absolute difference of ground truth and Gibbs probability for all the unique test instances.

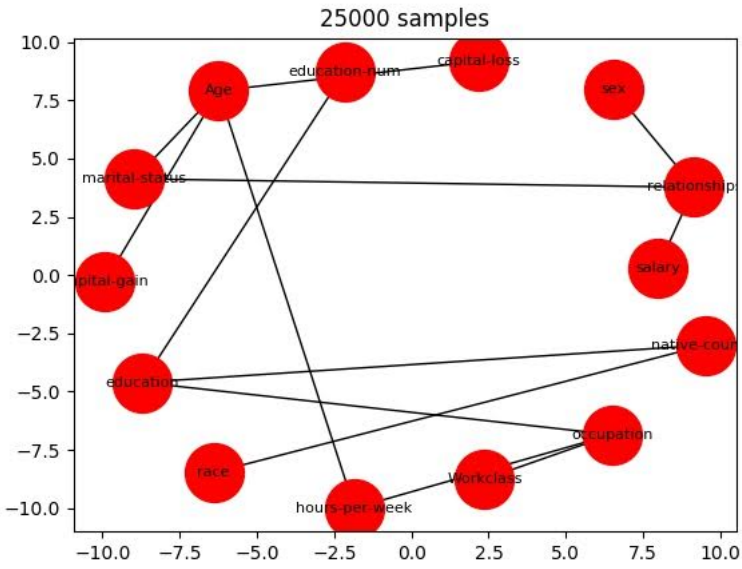

Figure 2: Bayesian Network as generated by Chowliu.py

2.3 Observations and Inferences

The following errors were observed after varying the Gibbs sample size and the sample iterations.

N_samples\Iter	200	400	600	800	1000
5000	0.9948	0.9951	0.9954	0.9958	0.9972
10000	0.9891	0.9909	0.9907	0.9892	0.9893
15000	0.9883	0.9877	0.9916	0.9878	0.9884
20000	0.9885	0.9897	0.9896	0.9893	0.9906

Figure 3: Errors for different Sample size and Sample iterations

2.3.1 Varying Sample Size

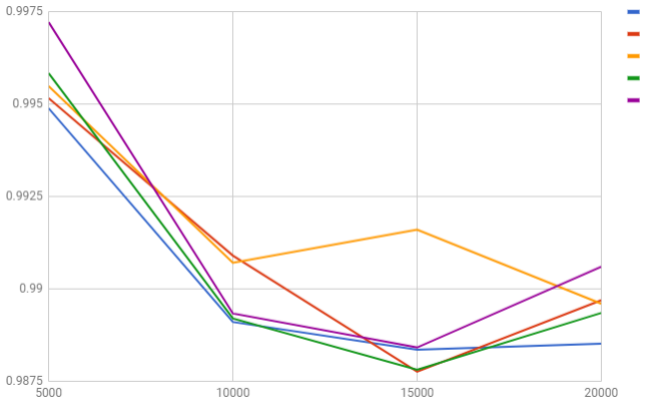

Figure 4: Change in error by changing the sample size for a constant number of iterations

From the above graph we observe that the net error decreases on increasing the number of Gibbs samples. Increasing the size of the Gibbs sample set increases the probability of the the test instance getting sampled from the true distribution therefore a net decrease in error is observed.

2.3.2 Varying Number of Iterations

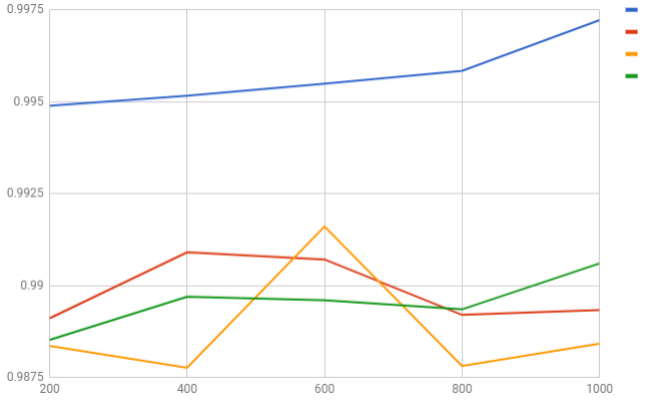

Figure 5: Change in error by changing the number of iterations for a fixed sample size

Theoretically Gibbs Sampling produces sample from the query distribution P(Q/e) in the limit of resampling often. In the limit of increasing the number of iterations infinitely many times the resulting sample comes from the correct distribution. Therefore on increasing the number of iterations for generating a Gibbs sample theoretically should result in a better estimate and reduce the net error.

From the graph the trend that is observed is that the net error remains constant. The most likely reason for this is the skewness in the data set for some particular variable values. Although increasing the iteration to a very high number might result in a better fit but the trade off would be a huge computing time.