

Cinthia Valdéz Guillen A01203141

Cristian Alexsis González Jaime A01204478

Luis Pablo Morales Cruz A01205724

Report Lab04 Implementing Bayesian Networks

Delivery date:

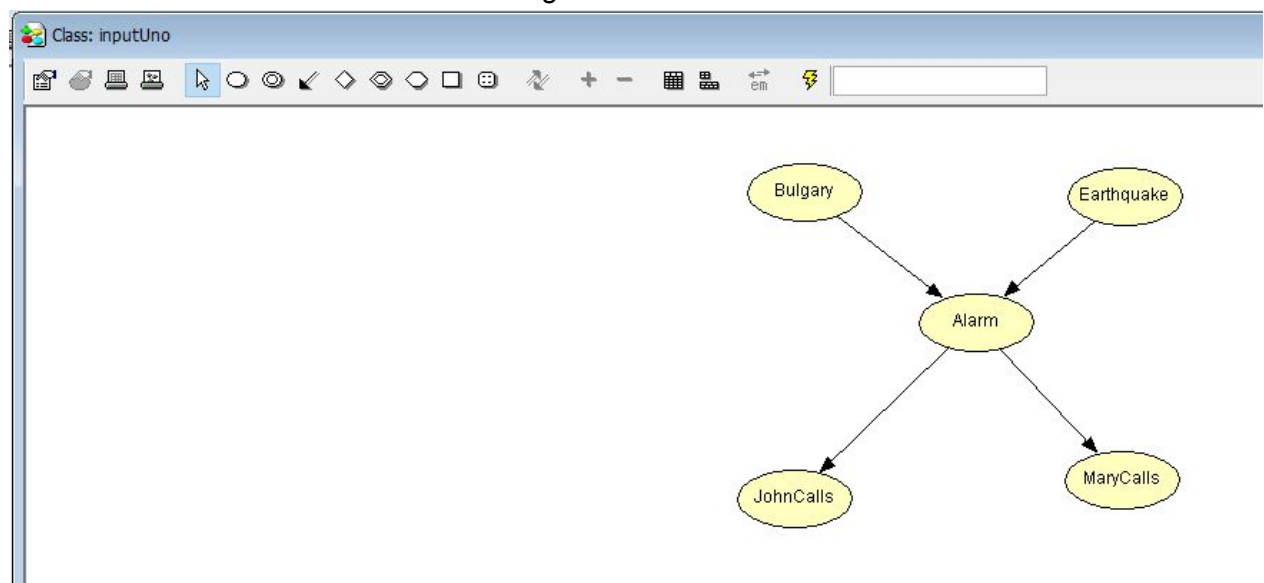
For this time the lab was about to create a bayesian network from the scratch using the programming language that we want, for this case we choose c++ language, to test our program the teacher gave us two test cases, this two cases have to be tested on hugin lite and in our program.

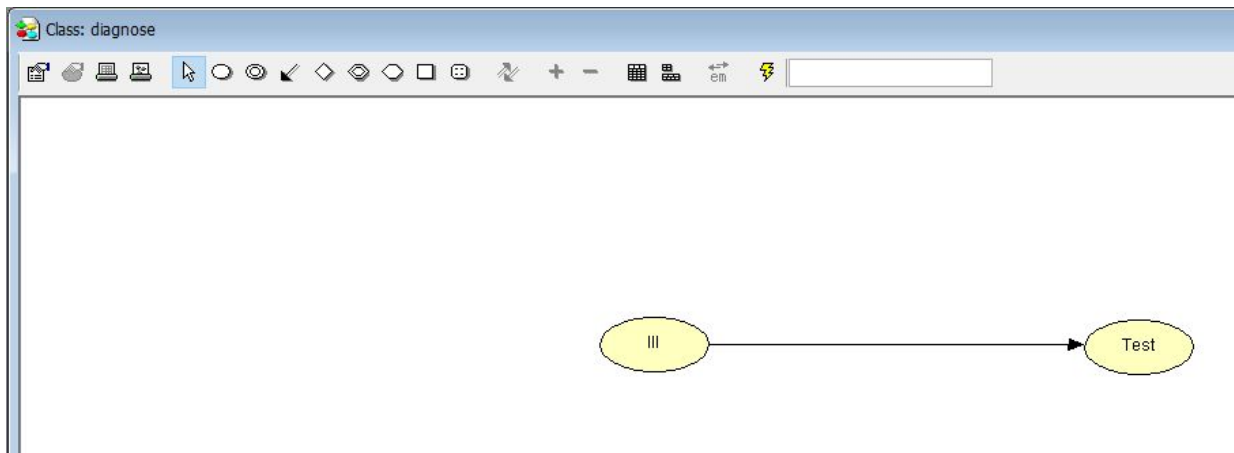
First we are going to talk about Hugin lite, Hugin is a software based on bayesian networks and influence diagram technology, an advanced artificial intelligence technique widely used for supporting decision-making under uncertainty.

On this program you design your bayes network, on a graphical way, so for each variable associated with a cause, probability is used to specify the extent that one variable affects the other. Bayesian networks can operate with multiple sources of information: historical data and the more subjective observations of experts and other data that is missing, uncertain or complicated but has valuable impact on the decision support model.

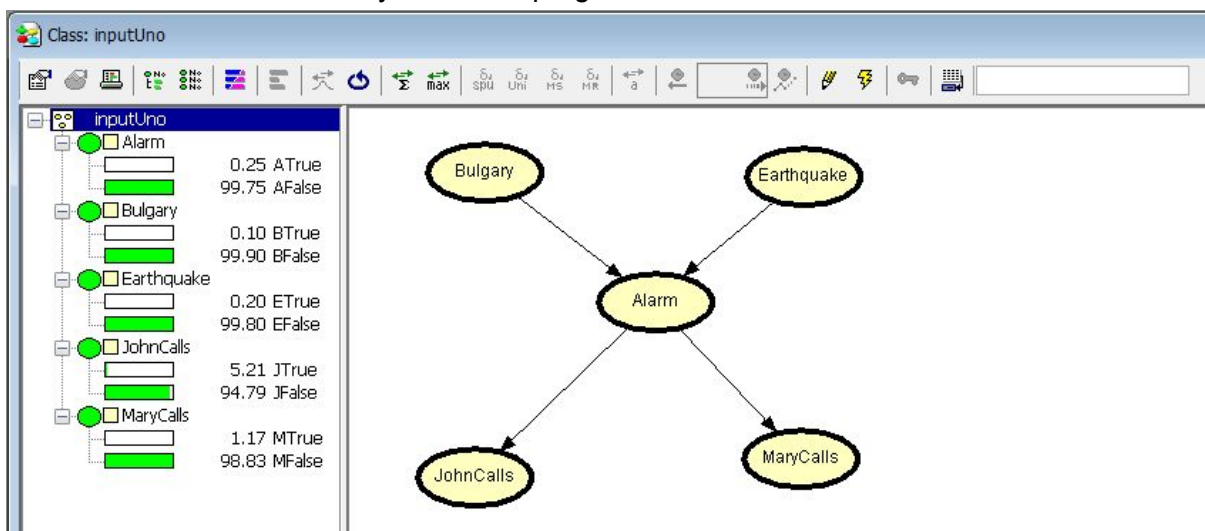
So the network has to be filled with all the data that every node needs, this is a weak point for the program, because if we have a big amount of data, we will spend many time to put all the data. So for industrial terms, this tool is not a good choice, at least the free version is ok for academical terms, and maybe the pay version of hugin have other characteristics.

This are our two networks modeled on hugin:

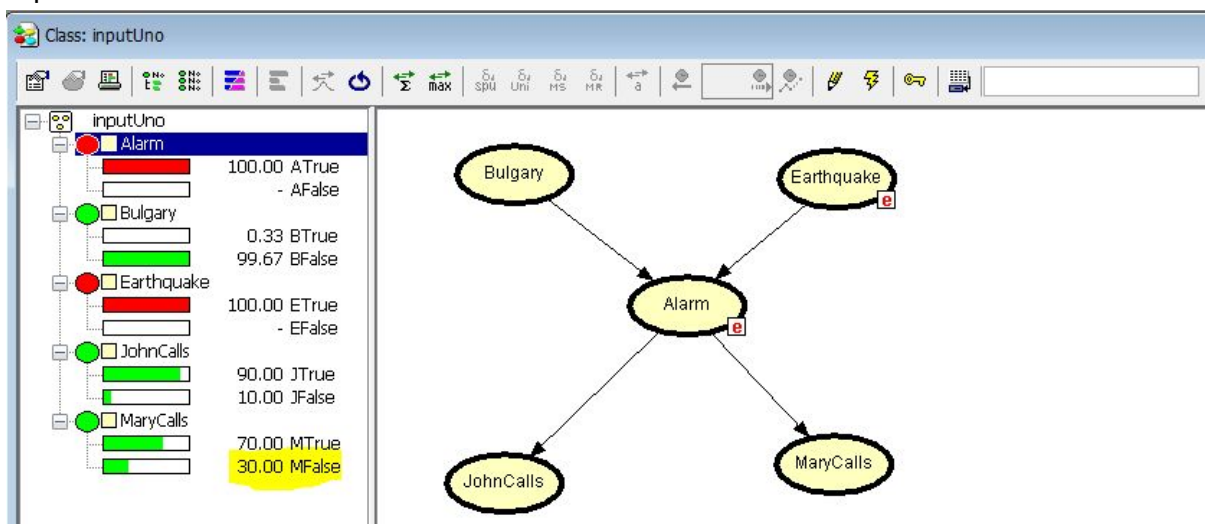




In this case we are going to use the first network, because is the most complex, so this is how the network looks when you run the program:



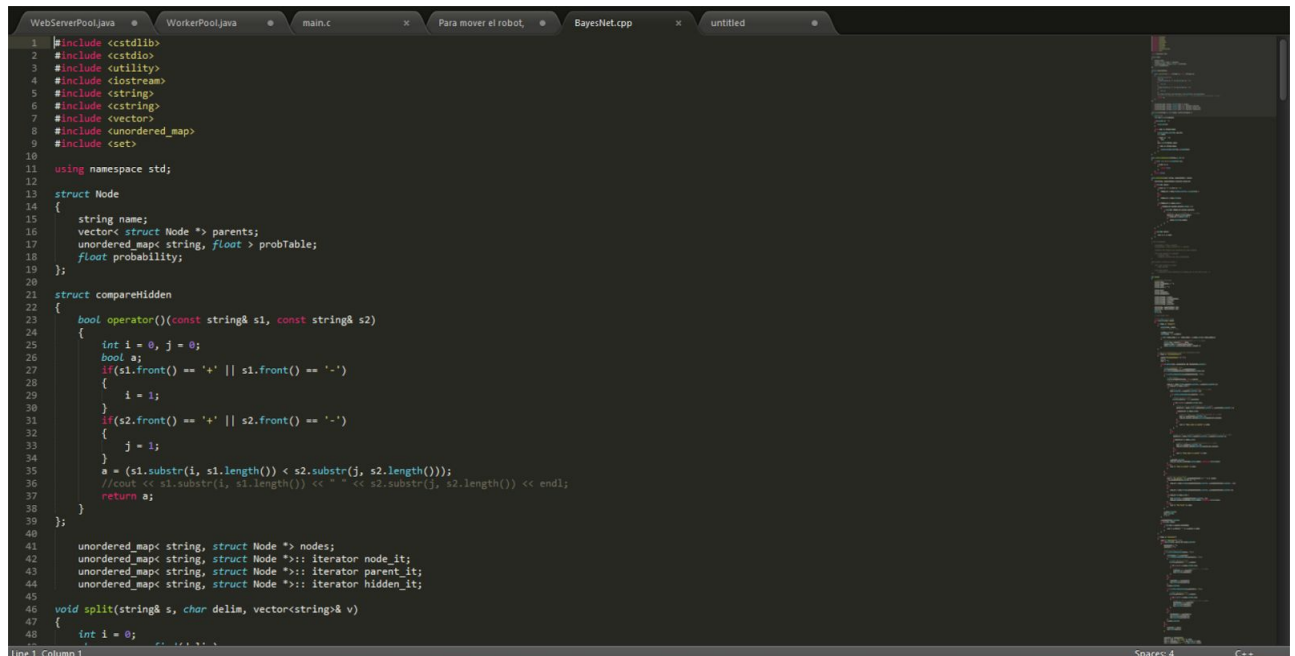
As you can see the program shows you all the probabilities, so let's run one of the examples, we are going to run -MaryCalls|+Earthquake, +Alarm, according to the given output the expected result must be 0.3



And indeed the result in hugin is 30.00% or 0.3, because hugin give the results in percentage.

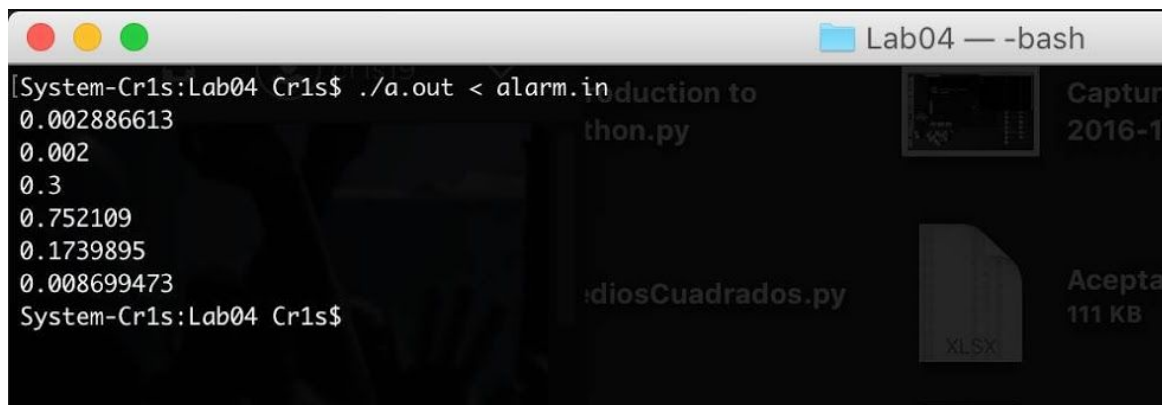
Now in the case of the program, we implement a bayesian network from the scratch, we use a hash table to store the data, because in that way we can acces to the required data more easy, so first of all the challenges of this laboratory was, first of all the lecture of the input, then the structure to store the data, because we know what is a hash table, but we never code one, and for finish the recursivity function to know which nodes are parents and which are parents from parents, this was the part of the lab that took more time because is not only about to find how to do it, it's about to implement it on a program language that .

This is how part of the code looks like:



```
1 #include <cstdlib>
2 #include <cstdio>
3 #include <utility>
4 #include <iostream>
5 #include <string>
6 #include <cstring>
7 #include <vector>
8 #include <unordered_map>
9 #include <set>
10
11 using namespace std;
12
13 struct Node
14 {
15     string name;
16     vector< struct Node *> parents;
17     unordered_map< string, float > probTable;
18     float probability;
19 };
20
21 struct compareHidden
22 {
23     bool operator()(const string& s1, const string& s2)
24     {
25         int i = 0, j = 0;
26         bool a;
27         if(s1.front() == '+' || s1.front() == '-')
28         {
29             i = 1;
30         }
31         if(s2.front() == '+' || s2.front() == '-')
32         {
33             j = 1;
34         }
35         a = (s1.substr(i, s1.length()) < s2.substr(j, s2.length()));
36         //cout << s1.substr(i, s1.length()) << " " << s2.substr(j, s2.length()) << endl;
37         return a;
38     }
39 };
40
41 unordered_map< string, struct Node *> nodes;
42 unordered_map< string, struct Node *>::iterator node_it;
43 unordered_map< string, struct Node *>::iterator parent_it;
44 unordered_map< string, struct Node *>::iterator hidden_it;
45
46 void split(string& s, char delim, vector<string& v)
47 {
48     int i = 0;
```

So for the program we will test the same example to see if there is any difference on the obtained values:



```
Lab04 — -bash
[System-Cr1s:Lab04 Cr1s$ ./a.out < alarm.in
0.002886613
0.002
0.3
0.752109
0.1739895
0.008699473
System-Cr1s:Lab04 Cr1s$
```

As you can see there is no difference on the results between the program and hugin.

So, as we can see both solutions are available to solve bayes problems, depending of the problem, you have to choose the correct tool, because they share the same bayesian rules and probabilistic rules, so the main thing is to analyze the problem properly in order to choose the correct solution, because you are not implementing a program for a small amount of data, and you are not going to use hugin to analyze millions of data, both works but the way you insert the data and how you build it is different.

Bibliography:

Hugin Experts. (-). Hugin GUI Help. 20/10/2016, de Hugin Experts Sitio web:
<http://download.hugin.com/webdocs/manuals/Htmlhelp/>