USING FUNCTIONAL DEPENDENCY THRESHOLDING TO DISCOVER FUNCTIONAL DEPENDENCIES FOR DATA CLEANING

Ruben Smink
University of Twente
PO Box 217, 7500 AE Enschede
the Netherlands
r.b.smink@student.utwente.nl

ABSTRACT
Cleaning data is important before it can be processed. Erroneous data needs to be filtered out or repaired in order to achieve good results. One interesting method is to use functional dependencies to clean data. This is possible to do by hand on smaller data sets. However, when the data sets become larger and contain more attributes, this becomes labor intensive. In this paper, we describe a method of discovering functional dependencies useful for data cleaning. Using a method of data cleaning that uses FDs, we can test and evaluate how well a functional dependency performs. After this we can score them and use bayesian optimization to threshold the minimum score for a functional dependency to have a positive impact on the data cleaning process.

Keywords
data cleaning, learning model, functional dependency, bayesian optimization, thresholding

1. INTRODUCTION
With more and more industries relying on data and the automatic processing of it, it is important to make sure this data is accurate. Inaccuracies in data could lead to erroneous conclusions which in turn leads to the wrong decisions being made. To increase the accuracy of information we can use data cleaning. This is the process of detecting inaccurate or corrupt data and modifying or deleting it in order to increase its accuracy [3]. One way to clean data is by using functional dependencies: "If Y is a function of X, we can say that X functionally determines Y, written $X \rightarrow Y$. This constraint is a functional dependency (FD)." (p.140 [5]) In data sets these FDs are between attributes. An example of an FD in a data set is: Postal code → City. This FD means that whenever you know the postal code, you also know the city. Thus, city is a function of postal code. These FDs are not universal between data sets. For the example FD, knowing the postal code does not always allow you to know the city. This depends on where the data is recorded.

Recent work has shown that integrity constraints such as FDs can be used to train machine learning models for data cleaning in a weakly supervised manner. [5] The only issue with this approach is that the FDs still need to be entered manually. It is not possible to simply check whether a functional dependency is met in a database due to errors in the data. It would be possible to have domain experts analyze the data and define FDs by hand. However, this becomes quite difficult as the amount of attributes grows and is it very labor intensive to do. Thus, it is attractive to automate the process of discovering FDs.

Therefore, multiple algorithms and methods have been developed to automate the discovery of FDs in an unclean database.

1.1.1 Data mining
The data mining community has attempted to view FDs as statistical dependencies. With this point of view they are able to determine FDs using an information-theoretic approach. [8] This method is mainly focused on data profiling, but can also be used for data cleaning.

1.1.2 Database
The database community attempted to find approximate FDs that are not often violated in a database. [9] This approach does not work well for data cleaning as the noisy data can lead to incorrect FDs.

1.1.3 Machine learning
The machine learning community attempted to find FDs by viewing noisy data as a graphical model over binary random variables. They were then able to use structure learning to learn this model and thus determine FDs from it. [10] Similarly to the data mining approach, this method can be used to successfully clean data. However, data cleaning is not its main purpose.

1.1.4 Proposal
The already existing methods can all be used for data cleaning, however it is likely possible to improve upon when focusing solely on data cleaning.

In this paper we will attempt to discover FDs for the purpose of data cleaning. To be able to do this we will be using an open source version of the system that trains a machine learning model to clean data using manually entered FDs called holoclean[1].

The issue with attempting every single combination of FDs is that it would take a very long time, as holoclean takes some time for every evaluation. This problem only grows worse when you consider that the amount of possible FDs increases exponentially when increasing the amount of attribute sets in a dataset.

[1]https://github.com/HoloClean/HoloClean

This is why we use machine learning to solve the problem. We can score FDs in multiple different ways. To prove our concept we will use the weight that holoclean gives to an FD and the mutual information between the 2 attributes that the FD consists of. Different methods of scoring could be used and this will be discussed in section 11. When sorting FDs on these scores, we can use thresholding to determine the minimum amount of weight or mutual information is necessary in order to have a positive effect on the data cleaning process.

To perform the thresholding we use bayesian optimization. "Bayesian optimization (Mockus et al., 1978) is a method for performing global optimization of unknown "black box" objectives that is particularly appropriate when objective function evaluations are expensive (in any sense, such as time or money)." [4]

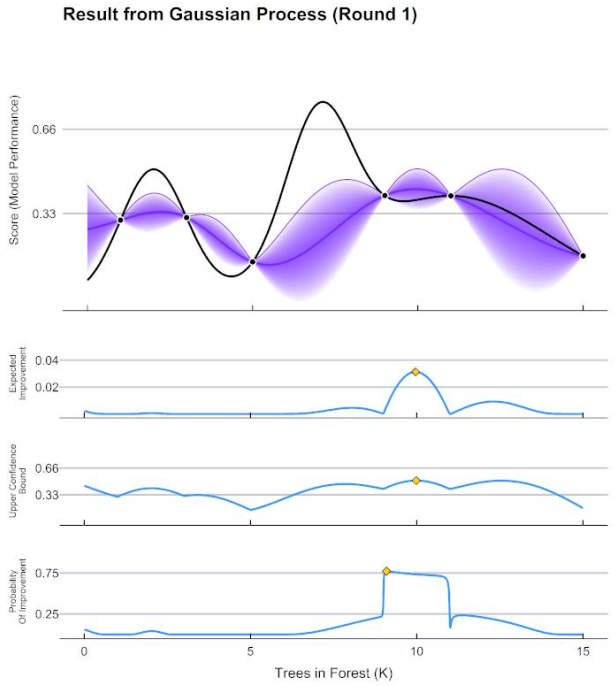

Figure 1: The results of one round of bayesian optimization[2].

Bayesian optimization works by first evaluating a few random points. Using this information, a surrogate model is created. This surrogate model is called a prior. The prior is an estimation of the model we are evaluating. However, it is much cheaper to evaluate. This allows us to minimize the amount of evaluations on the model we are evaluating. The prior contains a posterior probability distribution, The model to evaluate and the prior with its posterior probability distribution as a purple area can be seen in figure 1. After the prior is created, the optimal hyperparameter will be estimated by using one of the possible acquisition functions which maps beliefs on how promising each hyperparameter is when evaluated next.

[2]https://github.com/AnotherSamWilson/ParBayesianOptimization

The most popular ones are: Expected Improvement (EI), Upper Confidence Bound (UCB) and Probability Of Improvement (POI). These can be seen in figure 1. The optimal hyperparameter is selected using the maximum of one of these acquisition functions. When the optimal hyperparameter is evaluated, it is added to the prior, after which the process is repeated.

2. RESEARCH QUESTION

In this paper we will attempt to answer the following question.

1. Is thresholding using bayesian optimization a viable method of discovering FDs for the purpose of data cleaning.
   (a) How well does this method perform when compared to other existing methods.
   (b) How does the time requirement of this method scale with the amount of attributes in a data set.

3. PROBLEM STATEMENT

We consider a relational schema R. For this schema there exists a probability distribution P_R which is able to generate data according to R. Using this probability distribution a data set D is generated. However, we assume that there are some errors due to missing or incorrect values, this is called noise. As such we will denote the clean data set with D and the noisy data set with D'. We consider a value in a cell (c) in D' an error when D'(c) ≠ D(c). This procedure is the same as the often used method described in database literature [1].

In this case both D and D' are known. Our goal is to find FDs in D' that have a positive effect on the data cleaning process. We denote the attributes as $A = \{A_1, A_2, ..., A_N\}$ and the set of attribute pairs as $T = \{(i, k) \text{ where } i,k \in A \text{ and } i \neq k\}$. Every attribute pair in this set represents an FD.

4. SOLUTION OVERVIEW

In order to use Function Dependency Thresholding (FDT) to discover FDs for data cleaning. We go through a few steps:

4.1.1 Scoring

First, in order to make thresholding possible we need to score the FDs. This will allow us to sort them. The way we do it is by scoring the weight assigned to an FD by holoclean and the mutual information between the attribute pair in the FD. This is further discussed in section 5.

4.1.2 Optimization

After the scoring and sorting we are able to threshold the minimum amount of weight or mutual information necessary for the FD to provide good data cleaning results. This is done with bayesian optimization. Not only will the FDs be optimized, but also the threshold of certainty necessary for an error to be repaired. This is further discussed in section 6.

4.1.3 Evaluation

After every round of optimization we evaluate the results using holoclean. This program calculates a precision, recall and F1-score for the data cleaning done using our set of FDs. We can then use this result to fine tune the thresholds in another round of optimization.

5. SCORING

5.1 Weight

This score is extracted from the output of holoclean. The specific weight we use is this OccurAttrFeaturizer. Extracting these weights requires holoclean to be run once. This allows us to extract the weight.

5.2 Mutual Information

The second score is the mutual information between the 2 attributes in an FDs (p.7 [2]). This is calculated using the formula.

$$I(X;\ Y) = H(X) - H(X|Y) = \sum_{x,y} p(x,\ y)\ log\ \frac{p(x,y)}{p(x)p(y)}.$$

Where H(X) is the entropy of attribute X and H(X|Y) is the conditional entropy. When the mutual information between 2 attributes is high, it indicates a possible FD.

6. OPTIMIZATION

The issue with attempting every single combination of FDs is that it would take a very long time. Evaluating a set of FDs using holoclean is very costly. Thus, the amount of evaluations needs to be minimized.

For a task like this machine learning comes to mind. Bayesian optimization is well suited for a task like this because of its low evaluation count and its ability to work on a black-box function.

The python library we use for the optimization[3] uses the gp_hedge acquisition function. This is a method that combines 3 acquisition functions we have mentioned before: Expected Improvement (EI), Upper Confidence Bound (UCB) and Probability Of Improvement (POI).

Using this method, each acquisition function is optimized independently, after which one of the methods is probabilistically chosen to evaluate. The process of choosing the acquisition involves how strong the belief of the acquisition function is and the average gain of it's previous evaluations.

Other than the set of FDs, we also optimize the threshold for a cell to be seen as erroneous by. During the repair holoclean decides for each cell a certainty of being erroneous. When the threshold is set to 0, it means every cell that is even remotely seen as erroneous will get repaired. When the threshold is 1 it means only cells with 100% certainty of being erroneous will be repaired.

Because of the different scores and the threshold having very little effect on each other we can optimize them separately. This takes the complexity of the problem from $O(n^3)$ to $O(n)$, where n is the amount of FDs and 3 is the amount of optimizations we do.

[3]https://scikit-optimize.github.io/stable/modules/generated/skopt.Optimizer.html

7. EXPERIMENTS

In order to evaluate our method, we start off by testing our approach on 3 synthetic data sets and 1 real data set. The results we will measure and discuss are:

- The precision, recall and F1 scores of the optimized set of FDs.
- The time required to perform the data cleaning operation.

Whilst the duration of the optimization is measured. The results will vary a lot due to low end hardware and usage of the hardware during the optimization process.

8. EXPERIMENTAL SETUP

8.1 Existing Methods

We will compare the performance on the real data set of our method to 3 existing methods.

8.1.1 PYRO

This is the state-of-the-art discovery method for FDs in the database community [9]. This method seeks to find all syntactically valid FDs in a data set. A java code released by the authors is used to test on the real data set[4].

8.1.2 TANE

"TANE is based on partitioning the set of rows with respect to their attribute values, which makes testing the validity of functional dependencies fast even for a large number of tuples." [7] This method is included in the java code for PYRO[4].

8.1.3 Reliable Fraction of Information (RFI)

This is the state-of-the-art discovery method for FDs in the data mining community [10]. RFI uses an information theoretic score to find FDs and uses an approximation scheme to optimize the performance. The method has a hyperparameter α that controls the approximation ratio. This functions the same as the error detection threshold in holoclean where 0 includes all FDs that have above 0% likelihood of being a good FD and 1 only includes FDs that have a 100% likelihood of being a good FD.

As the real data set we use has been tested often in other papers. We can use the results of an earlier published paper [11] to compare the results of RFI.

8.2 Data Sets

First, we will test the synthetic data sets. These data sets are generated using often used data generation methods and can be seen as a benchmark for our method. First we will take a relational schema R and use a standard method that uses probability distribution P_R In order to generate a data set D. For the relational scheme we will use Bayesian Networks. After which we use an R package to generate data set D[5]. We have chosen to generate 1000 rows of data per data set.

After the data set D is generated we can generate noise. In order to generate the noise, we first set the amount of noise we want in data set D'. For thorough testing we have to select a range of noise settings. We use N = {0.01, 0.02, 0.03, 0.04, 0.05, 0.10, 0.15}.

[4]https://github.com/sekruse/pyro

[5]http://www.bnlearn.com/bnrepository/

This allows us to see the performance of our method on an increasing amount of noise. We generate the noisy data set by iterating through every cell. Every cell has a chance equal to N to become erroneous. When a cell (c) with value (v) on attribute (A_i) is selected to become erroneous, we select a random value (v') that is not (v) from the same attribute in different rows. Thus, we take v' where v, v' $\in A_i$ and v' $\neq$ v.

The order we have decided on optimizing is first the weight, then the error threshold and finally the mutual information. We have chosen this order because the error threshold requires a set of FDs with a good performance in order to meaningfully optimize it. After the threshold is set we can then perform optimization on the mutual information with the optimal performance.

In order to be able to discuss the results of testing later. We should first know the kind of synthetic data sets we are working with.

8.2.1 Cancer

This data set is a Bayesian Network consisting of 5 nodes and 4 arcs. We can see in figure 2 that the Cancer node depends on both Pollution and Smoker.

probability (Cancer | Pollution, Smoker)

(low, True) 0.03, 0.97

(high, True) 0.05, 0.95

(low, False) 0.001, 0.999

(high, False) 0.02, 0.98

Upon inspection of the network, the chances for Cancer depends heavily on the output of these 2 nodes. However, the chance of Cancer still remains very low. This makes it difficult to perform any repairs on these 2 nodes, as they do not have any strong FDs.

probability (Xray | Cancer)

(True) 0.9, 0.1

(False) 0.2, 0.8

probability (Dyspnoea | Cancer)

(True) 0.65, 0.35

(False) 0.3, 0.7

For both Xray and Dyspnoea Their output is very dependent on the Cancer node. This dependency means they have strong FDs. We can expect good repairs between these 3 nodes because of this. In conclusion we would only expect to be able to repair 3 out of 5 nodes accurately. With this data set we do not expect a very good quality repair.

8.2.2 Asia

This data set is a Bayesian Network consisting of 8 nodes and 8 edges. When inspecting the arcs between the nodes in figure 3, we can see strong FDs between most of the nodes.

probability (xray | either) {

(yes) 0.98, 0.02

(no) 0.05, 0.95

probability (either | lung, tub) {

(yes, yes) 1.0, 0.0

(no, yes) 1.0, 0.0

(yes, no) 1.0, 0.0

(no, no) 0.0, 1.0

Xray and Either are good examples for nodes that have strong FDs with the nodes that they take as input.

probability (tub | asia)

(yes) 0.05, 0.95;

(no) 0.01, 0.99;

probability (lung | smoke) {

(yes) 0.1, 0.9;

(no) 0.01, 0.99;

probability (bronc | smoke) {

(yes) 0.6, 0.4;

(no) 0.3, 0.7;

When looking at the nodes: Tub, Lung and Bronc. We can see that they are not very dependent on their input. Bronc being the most dependent, but still not enough for a strong FD. In conclusion, because 6 out of 8 nodes have strong FDs, we expect a good quality of data cleaning.

8.2.3 Child

This data set is a medium sized Bayesian Network consisting of 20 nodes and 25 arcs. In figure 4 we can see the network. Upon inspection of this network. We can see that most of the nodes in the network have a lot of different inputs.

probability (LowerBodyO2 | HypDistrib, HypoxiaInO2)

(Equal, Mild) 0.1, 0.3, 0.6;

(Unequal, Mild) 0.4, 0.5, 0.1;

(Equal, Moderate) 0.3, 0.6, 0.1;

(Unequal, Moderate) 0.50, 0.45, 0.05;

(Equal, Severe) 0.5, 0.4, 0.1;

(Unequal, Severe) 0.60, 0.35, 0.05;

Taking LowerBodyO2 as an example. The node has two different inputs and more possible outputs than the nodes in the other data sets. With more complex dependencies it would likely be possible to perform repairs on data sets like this. However, with only FDs containing 2 attributes, we will likely be unable to perform many correct repairs. In conclusion we expect a very low quality repair on this data set.

8.2.4 Real data set

For the real data set we will be using an often used benchmark data set called hospital. This data set has 19 attributes. For the data cleaning process only 17 of them are relevant due to 2 of the attributes not containing any values.

This data set is a fairly standard data set containing hospital procedure information. The data contains some attributes with quite strong FDs. A few examples of this are: When the name of a hospital is the same, it often means that the zip code, address, city and phone number are the same. This allows Holoclean to perform very high quality data cleaning on this data set. This data set is included in the source code of the open source version of holoclean[1].

9. RESULTS

Table 1: Results of synthetic data sets.

Data Set	N	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
Cancer	P	1.000	1.000	1.000	1.000	1.000	1.000	1.000		
	R	0.183	0.202	0.255	0.168	0.189	0.204	0.198	-	-
	F_1	0.310	0.336	0.406	0.288	0.318	0.339	0.331		
Asia	P	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	
	R	0.598	0.521	0.494	0.484	0.513	0.117	0.123	0.120	-
	F_1	0.748	0.685	0.661	0.653	0.678	0.209	0.219	0.214	
Child	P	1.000	0.875	1.000	1.000	1.000	1.000	1.000		
	R	0.030	0.017	0.015	0.005	0.003	0.006	0.001	-	-
	F_1	0.057	0.033	0.029	0.010	0.006	0.013	0.003		

"-" no correct repairs have been performed.

Table2: Average time taken by FDT.

Data Set	Duration
Cancer	0:28:31
Asia	0:36:03
Child	2:59:48

Table 3: Results of real data set. (error threshold of 0.5 for other methods)

Data Set		FDT	PYRO	TANE	RFI (0.3)	RFI (0.5)
Hospital	P	0.956	0.923	0.923	1.000	1.000
	R	0.853	0.825	0.827	0.424	0.424
	F_1	0.901	0.871	0.873	0.596	0.596

Table 4: Time taken for the real data set.

Algorithm	Duration
FDT	4:01:22
PYRO	0:00:03
TANE	0:00:00
RFI (0.3)	1:47:37
RFI (0.5)	1:50:03

9.1 Synthetic Data

In table 1 we can see that just as predicted, the Asia data set performs the best, followed by the Cancer data set and the Child data set being the worst for our data cleaning method.

Next to this we can see that cleaning data using FDs works well up till the point where there is too much noise to be able to recognize the FDs, at which point the cleaning performance drops off very quickly, often becoming unable to perform any useful repairs after still performing quite well one noise percent lower. An example is the Cancer data set having an F_1 of 0.331 at 0.07 noise and not performing any useful repairs at 0.08.

The most interesting part of the optimization is the FD selection. Due to the splitting of our optimization, we can see how many FDs the bayesian optimization adds to the set of optimal FDs. For the first optimization round we can see in table 6 that in most cases, a large amount of the FDs is selected. Then when the error threshold is optimized, the mutual information optimization will add the remaining FDs. This means that in most cases all the FDs are useful as long as the error threshold is optimized.

When the noise of the data set increases, the optimizer often starts picking less FDs. This has mostly happened to the Cancer and Child data set. This can be seen in table 6.

The FD selection has an effect on the time required by our method. In some cases the default threshold already allows for all the FDs to be added in the first optimization round. When there are no FDs left to add, the optimization stops early. Taking examples from the raw data for optimizations with only one FD thresholding round and the error threshold round:

Cancer: 5 attributes, 0:29:56

Asia: 8 attributes, 0:42:55

Child: 20 attributes, 2:10:22

We can see that because of the splitting of the attributes the time required by our method scales linearly with the amount of attributes.

9.2 Real Data

In order to generate fair results we have used the error threshold recommended by holoclean for this data set [10]. We can see that our method pulls ahead of the other existing methods on the F_1-score. However, our method requires much more time than the existing methods. Whilst the other methods are tested on much more powerful hardware, the difference is very likely still quite large when run on equal hardware.

10. DISCUSSION

In conclusion, thresholding FDs using bayesian optimization is a viable method of discovering FDs for the purpose of data cleaning. The tests have shown that our method provides the expected results on synthetic data sets and competitive results on the real data set. However, whilst the time required by this method scales linearly with the amount of attributes. The time required to perform the optimization is still quite long. This means that this method is only viable for cases where the frequent generation of a new set of FDs is not required.

11. FUTURE WORK

Much can be done to improve our method. Currently we are only looking at one of the simplest dependencies. If we were able to threshold multi-valued dependencies we would be able to perform much higher quality repairs, as a lot of the attributes are affected by more than just one other attribute. This could be achieved with better scoring methods. In general, whilst the weight and mutual information are working well as a way of showing the viability of this method, there likely exist much better methods of scoring dependencies.

It would also be interesting to test the amount of evaluations that are necessary before the optimal threshold is reached. Bayesian optimization is not able to stop automatically when the optimal threshold is found. It will continue testing different hyperparameters until a maximum amount of evaluations or when there are no hyperparameters left to test. Finding the optimal amount of evaluations for different amounts of attributes will allow the method to execute much faster.

12. REFERENCES

1. Bergstra, J. *et al.* (2011) 'Algorithms for Hyper-Parameter Optimization', in Shawe-Taylor, J. et al. (eds) *Advances in Neural Information Processing Systems*. Curran Associates, Inc., pp. 2546–2554. Available at: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.
2. Cover, T. M. and Thomas, J. A. (2005) *Elements of Information Theory*. Wiley. doi: 10.1002/047174882X.
3. Douglas, K. M. and Sutton, R. M. (2010) 'Kent Academic Repository', *European Journal of Social Psychology*, 40(2), pp. 366–374. Available at: https://doi.org/10.1016/j.ress.2012.12.021.
4. Gelbart, M. A., Snoek, J. and Adams, R. P. (2014) 'Bayesian Optimization with Unknown Constraints', *Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014*, pp. 250–259. Available at: http://arxiv.org/abs/1403.5607.
5. Guo, Z. and Rekatsinas, T. (2019) 'Learning Functional Dependencies with Sparse Regression', *arXiv*. Available at: http://arxiv.org/abs/1905.01425.
6. Halpin TA, M. T. (2008) *Information Modeling and Relational Databases*, *Information Modeling and Relational Databases. 2nd ed.* Elsevier. doi: 10.1016/B978-0-12-373568-3.X5001-2.
7. Huhtala, Y. *et al.* (1999) 'TANE: An efficient algorithm for discovering functional and approximate dependencies', *Computer Journal*, 42(2), pp. 100–111. doi: 10.1093/comjnl/42.2.100.
8. Kruse, S. and Naumann, F. (2018) 'Efficient discovery of approximate dependencies', *Proceedings of the VLDB Endowment*, 11(7), pp. 759–772. doi: 10.14778/3192965.3192968.
9. Mandros, P., Boley, M. and Vreeken, J. (2017) 'Discovering Reliable Approximate Functional Dependencies', in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, pp. 355–363. doi: 10.1145/3097983.3098062.
10. Rekatsinas, T. *et al.* (2017) 'HoloClean', *Proceedings of the VLDB Endowment*, 10(11), pp. 1190–1201. doi: 10.14778/3137628.3137631.
11. Zhang, Y., Guo, Z. and Rekatsinas, T. (2020) 'A Statistical Perspective on Discovering Functional Dependencies in Noisy Data', in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, pp. 861–876. doi: 10.1145/3318464.3389749.

13. APPENDIX

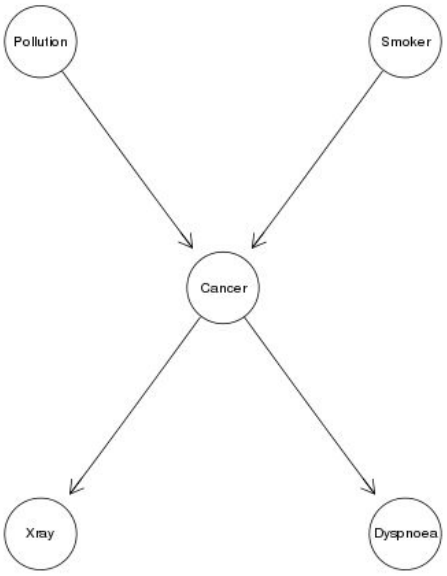

Figure 2: The nodes and arcs of the Cancer Bayesian Network[4].

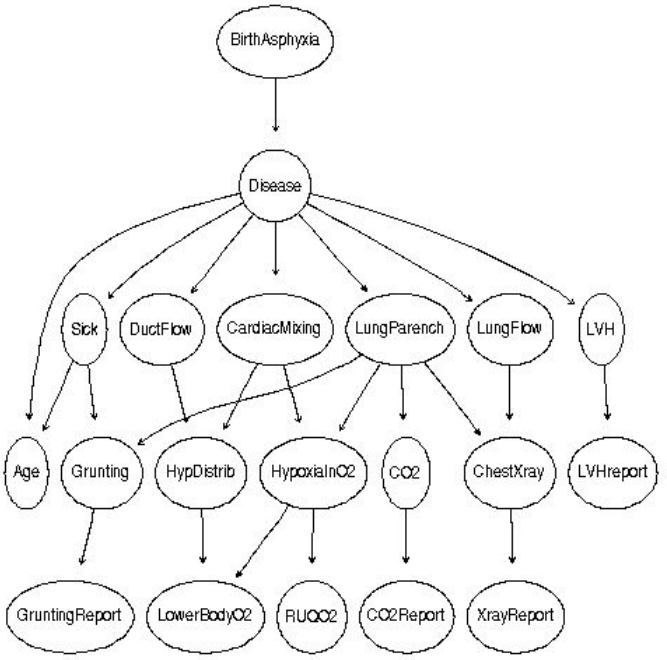

Figure 4: The nodes and arcs of the Child Bayesian Network[4].

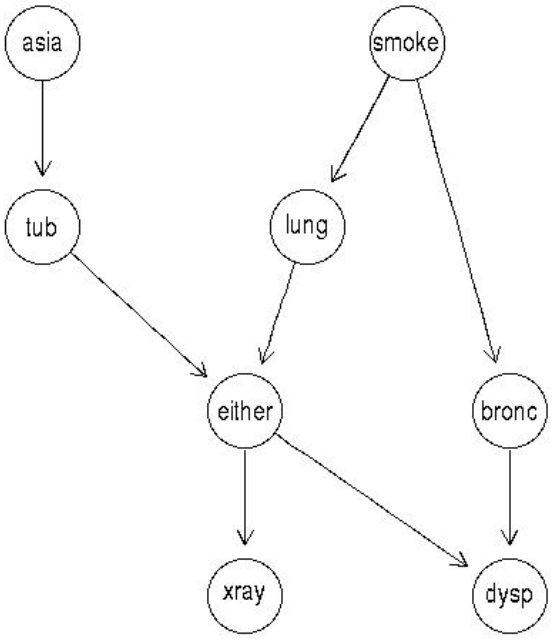

Figure 3: The nodes and arcs of the ASIA Bayesian Network[4].

Table 5: The amount of possible FDs in the data sets.

Data Sets	Possible FDs
Cancer	20
Asia	56
Child	380
Hospital	272

Table 6: Size of selected FD set and selected threshold for synthetic data sets.

Data Set		0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
Cancer	W	11	11	11	11	11	4	4	11	
	T	0.498	0.498	0.498	0.498	0.498	0.498	0.498	0.498	-
	MI	16	16	16	16	16	13	13	16	
Asia	W	48	56	49	49	56	56	56	56	29
	T	0.498	0.498	0.498	0.190	0.319	0.498	0.498	0.498	0.498
	MI	53	56	56	53	56	56	56	56	43
Child	W	380	1	1	1	1	1	1	191	
	T	0.498	0.498	0.498	0.498	0.498	0.498	0.498	0.498	-
	MI	380	2	2	2	2	2	2	286	

Table 7: Size of selected FD set and selected threshold for real data set.

Hospital	W	267
	T	0.558
	MI	270