Project Report
Graphical Models LAB

Maurice Wenig
Friedrich Schiller University Jena
Germany
maurice.wenig@uni-jena.de

1 INTRODUCTION

Learning graphical models is a big aspect of machine learning. But for the parameters of the model to be learned, it is often assumed that the structure is already given, which includes information about dependencies and independencies between features. However in practice, that is not the case. Sometimes even the opposite might be true, that the structure between features itself is much more relevant than the actual amplitude of their interactions, with the hope of being able to interpret the interactions (e.g. in a causal context). This is also an inspiration for our assignment, where our main task is coming up with a search strategy to find a sparse Bayesian Network that fits the data we were given.

For our assignment, we focused on Gaussian Bayesian Networks (GBN). A GBN is a Bayesian Network where the conditional distributions of a feature given their parents is a Gaussian. Another constraint to the GBN is that the mean paramater of the Gaussian, that is the conditional distribution, can only depend linearly on the values of the feature's parents in the Network Graph. So with n as the total number of features, the problem of learning the distribution $p(x_i|x_{\mathrm{pa}(i)})$ for a feature $i \in [n]$, its value $x_i \in \mathbb{R}$, and the value of their parents $x_{\mathrm{pa}(i)} \in \mathbb{R}^{n_i}$ is equivalent to learning $\hat{\beta}_i^{\,*} \in \mathbb{R}, \hat{\beta}_i \in \mathbb{R}^{n_i}, \hat{\sigma}_i \in \mathbb{R}$ such that $p(x_i|x_{\mathrm{pa}(i)}) \sim \mathcal{N}(\hat{\beta}_i^{\,*} + \hat{\beta}_i^{\,\top} x_{\mathrm{pa}(i)}, \hat{\sigma}_i{}^2)$. Now for these parameters, the ML-estimates have a simple closed-form solution (it is just linear regression), so finding the best parameters for any given structure is simple and efficient.

The dataset we were given spans different attributes of Portuguese "Vinho Verde" wine, which was originally used in [1]. It consists of twelve features: eleven continuous physicochemical attributes and one discrete sensory attribute (quality). Here, i will treat quality as a continuous feature.

2 METHODS

For the structure learning algorithm, i implemented a tabu search, because it generally performs well in accuracy and runtime [3]. I also combined the tabu search with random restarts. The entire parameterized algorithm is illustrated in algorithm 1. It is important to note that the function arguments are all passed by reference, so mutations inside of functions will affect the passed argument even outside the function.

First we do a hill climb. After the first hill climb, we do t_0 tabu walks followed by another hill climb each. Then we do a random restart and do it all again. The random restart procedure is repeated t_1 times.

- t_0: number of tabu walks
- s_0: maximum number of steps of a tabu walk
- l: length of the tabu list

Algorithm 1: Tabu Search with Random Restarts

```
input : dataset D, initial DAG G
G_max ← copy of G;
HillClimb(D, G_max);
for t_0 times do
    TabuWalk(D, G_max, s_0, l);
    HillClimb(D, G_max);
end
for t_1 times do
    RandomRestart(D, G_max, s_1);
    HillClimb(D, G_max);
    for t_0 times do
        TabuWalk(D, G_max, s_0, l);
        HillClimb(D, G_max);
    end
end
```

- t_1: number of random restarts
- s_1: number of random steps during a random restart

2.1 Hill Climbing

In every iteration, the hill climbing algorithm computes the best possible elementary change among all changes. If this change improves the objective function, it is applied. This algorithm is illustrated in function 2.

Function 2: ClimbHill

```
input : dataset D, current DAG G
while S_max increases do
    S_G ← Score(G, D);
    c_max ← IMPOSSIBLY_BAD_CHANGE;
    C ← AllChanges(G);
    foreach c ∈ C do
        apply the change c to G;
        S_c ← Score(G, D);
        undo the change c in G;
        if S_c > S_max then
            S_max ← S_c;
            c_max ← c;
    end
    if S_c > S_G then
        apply the change c_max to G;
end
```

In order to avoid copying the adjacency matrix every time a change is applied, the adjacency matrix is changed in-place, and then the score is evaluated. This is also improves the efficiency of generating all possible changes, which is illustrated in function 3. The evaluation of the score can be sped up by only evaluating local changes of the changed nodes. This is further discussed in subsubsection 2.4.1.

Function 3: AllChanges(G)

```
input  : DAG G
output : all possible changes to G such that G is still a DAG
         after the application of the change
C ← ∅;
foreach (u, v) ∈ E_G do
    c ← FLIP(u, v);
    apply the change c to G;
    if G does not have a cycle then
        C ← C ∪ {c};
    undo the change c in G;
end
foreach (u, v) ∈ E_G do
    c ← DELETION(u, v);
    // deletions do not add cycles
    C ← C ∪ {c};
end
G^t ← TransitiveClosure(G);
foreach u, v ∈ V_G, u ≠ v do
    if (u, v) ∉ E_G and (v, u) ∉ E_{G^t} then
        c ← ADDITION(u, v);
        C ← C ∪ {c};
end
return C
```

Another key point to the efficiency of function 3 is the efficiency of the cyclicity check. For additions, this can be done by computing the transitive closure G^t of G, which can be done in $O(|V_G|^3)$. Then if $(v,u) \notin E_{G^t}$, the addition of (u,v) will not create a new cycle. For flips, i didn't find a more efficient way than completely checking the resulting graph for cycles. This can be done in $O(|V_G| + |E_G|)$ with Kosaraju-Sharir's algorithm [2]. But because we use ajdacency matrices instead of adjacency lists, Kosaraju-Sharir's algorithm takes $O(|V_G|^2)$. Therefore, the resulting time for generating all flips is $O(|E_G| \cdot |V_G|^2)$. It follows that the overall time for generating all possible changes is $O((|V_G| + |E_G|) \cdot |V_G|^2)$.

2.2 Tabu Walks

For the tabu walks, we use a FiFo-Queue that keeps track of hashes of each visited adjacency matrix. This queue is called the tabu list. The rest of the algorithm is very similar to hill climbing, just with minimal adjustments:

- changes that result in visited adjacency matrices are not considered
- in each iteration, the top change is applied, disregarding whether it improves the objective function
- if the current value of the objective function exceeds the initial value of the objective function, the tabu walk is stopped

To efficiently check which changes result in visited adjacency matrices, a counter with the number of occurencces in the tabu list for each hash is used. This counter can efficiently be updated whenever hashes of adjacency matrix are appended to or removed from the queue.

2.3 Random Restarts

For the random restarts, a list of all possible changes is generated at each step. Then one of them is chosen from a uniform distribution.

2.4 Choice of Optimized Score

For the score function $p(G|D) \propto p(D|G) \cdot p(G)$, i used the approximation

$$p(D|G) \approx p(D|G, \hat{\theta})$$
$$\hat{\theta} := \arg\max_{\theta} p(D|G, \theta)$$

and the prior

$$p(G) \propto \frac{1}{|E|^{\lambda}}$$

With this, the objective function can be decomposed into the sum of independent node scores and a regularization term.

$$\begin{aligned}\arg\max_G p(G|D) &= \arg\max_G \log p(D|G) + \log p(G) \\ &\approx \arg\max_G \log p(D|G, \hat{\theta}) - \lambda |E_G| \\ &= \arg\max_G \sum_{i \in [n]} S_i(G) - \lambda |E_G|\end{aligned}$$

where

$$S_i(G) := -|D| \cdot \log \hat{\sigma}_i - \frac{1}{2} \sum_{x \in D} \left(\frac{x_i - (\hat{\beta}_i^{\top} x_{\mathrm{pa}(i)} + \hat{\beta}_i^{*})}{\hat{\sigma}_i} \right)^2$$

and $\hat{\theta} := \left(\hat{\beta}_i, \hat{\beta}_i^{*}, \hat{\sigma}_i\right)_{i \in [n]}$ are the respective ML estimates for

$$p(x_i | x_{\mathrm{pa}(i)}) \sim \mathcal{N}(\hat{\beta}_i^{\top} x_{\mathrm{pa}(i)} + \hat{\beta}_i^{*}, \sigma_i^2)$$

A derivation of this can be found in subsection A.1.

2.4.1 Implications for Hill Climbing. Elementary changes (addition, substraction, flip of an edge) only influence local distributions. That means if we construct a graph G', where G' was made by applying an elementary change to an edge (u,v) in G, the comparison $p(G'|D) > p(G|D)$ can be evaluated locally:

$$\begin{aligned}&\sum_{i \in [n]} S_i(G') - \lambda |E_{G'}| > \sum_{i \in [n]} S_i(G) - \lambda |E_G| \\ \iff &\sum_{i \in [n]} S_i(G') - S_i(G) > \lambda(|E_{G'}| - |E_G|) \\ \iff &\underbrace{\sum_{i \in \{u,v\}} S_i(G') - S_i(G)}_{:=\Delta_S(G',G)} > \underbrace{\lambda\,(|E_{G'}| - |E_G|)}_{:=\Delta_E(G',G)}\end{aligned}$$

Where the second equivalence holds because $S_i(G)$ only depends on node i and its parents. Therefore $S_i(G') = S_i(G)$ for $i \notin \{u,v\}$.

Note that $\Delta_E(G', G)$ only depends on the type of change applied to G:

$$\Delta_E(G', G) = \begin{cases} 1 & \text{addition} \\ 0 & \text{flip} \\ -1 & \text{deletion} \end{cases}$$

$\Delta_S(G', G)$ measures the improvement of node scores when the change is applied to G.

Similarly, two alterations G_1 and G_2 of G can be compared:

$$\sum_{i \in [n]} S_i(G_1) - \lambda \left|E_{G_1}\right| > \sum_{i \in [n]} S_i(G_2) - \lambda \left|E_{G_2}\right|$$
$$\iff \Delta_S(G_1, G) - \Delta_S(G_2, G) > \lambda \left(\Delta_E(G_1, G) - \Delta_E(G_2, G)\right)$$

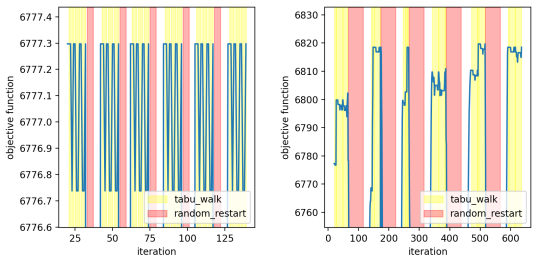
A derivation of this can be found in subsection A.2.

The interpretation of this is that a change has to bring an improvement of at least λ per edge in order to improve the whole objective function (which makes sense looking at the objective function). But more importantly, this allow a very efficient comparison of two different changes, which we need to efficiently find the best possible change for hill climbing and tabu walks.

2.5 Parameter Fine-Tuning

The parameters with the highest impact on the outcome are λ (in the objective function), s_0 (length of the tabu walks), and s_1 (length of the random walks). The impact of λ will be analyzed in subsection 3.1. With s_0 and s_1, we can fine tune the ability to walk out of slim local maxima, and stay in wider maxima. In order to do this, we can analyze the score history over time after the first hill climb.

In Figure 1, we look at two examples of a score history where the parameters still need some tuning. In Figure 1a the tabu walks do not manage to escape slim maxima, because tabu walks are too short. In Figure 1b the random restarts distort the top adjacency matrix too much and return to worse local maxima, because the random walks are too long.



(a) Short tabu walks **(b) Long random walks**

Figure 1: Score history after the first hill climb

2.6 Analysis

2.6.1 Parameterization. For all measurements, i parameterized the greedy search in three in different ways for different numbers of expected parameters (see Table 1).

Table 1: Parameterization of the greedy search

Parameterization	λ	t_0	s_0	m	t_1	s_1
big	0 - 3	3	150	2000	5	10
medium	3 - 25	3	80	400	5	5
small	25 - ∞	3	20	100	5	5

2.6.2 Likelihood. In order to measure the likelihood of the generated structures, i split up the data we were given into a training set and a test set. I trained the models with different λ on the train set to generate the adjacency matrices. Then i used cross validation on the test set to measure the likelihood of the ML-estimate of the parameters for each adjacency matrix.

2.6.3 Performance. While generating the structures with my train set, i also measured the runtime of the greedy search. The specifications of the machine the time was measured on can be found in Appendix B.

2.6.4 Submissions. We were also given a submission site where we could anonymously upload our adjacency matrices and see how we compare to others. The submission site computes the likelihood of our submitted adjacency matrices with a secret test set.

2.6.5 Parameterization Comparison. For the comparison of the small parameterization with the other two, i measured the likelihoods and runtimes on the same train and test sets, on the same machine, but on different days. For this reason, the runtime of the smaller models in its intended λ-region is included in the results section.

3 RESULTS

3.1 Generated Structures

The greedy search generates different structures depending on the regularization constant λ in the objective function. The number of paramaters in the generated structures is $\#params = 2\left|V_G\right| + \left|E_G\right|$ for any structure G. Because of the relationship between edge importance and λ discussed in subsubsection 2.4.1, this term is expected to decrease as λ increases. This was confirmed in practice. The results are illustrated in Figure 2.

3.2 Parameterization

As seen in the score history in Figure 3, the tabu walks are long enough to climb out of local maxima and the random restarts are short enough to not distort the adjacency matrix too much. Sometimes the random restarts even manage to jump into better maxima (second restart in Figure 3).

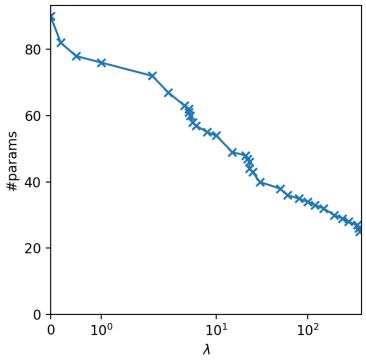
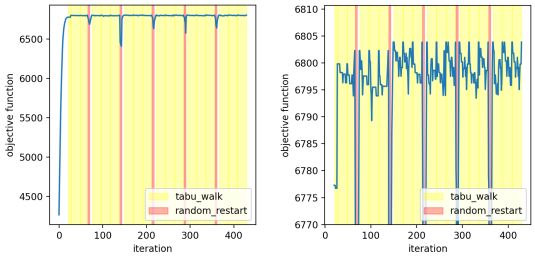


Figure 2: Number of parameters for structures generated by different λ



(a) with the first hill climb **(b) after the first hill climb**

Figure 3: Score history with successful tabu walks and random restarts

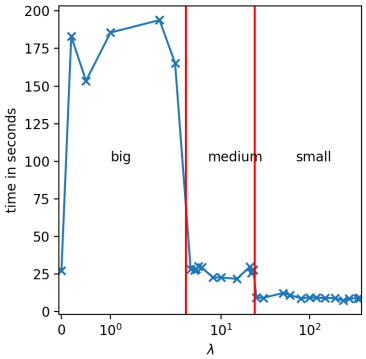


Figure 4: Runtime of greedy search with different λ

3.3 Performance

The results of the runtime measurements are illustrated in Figure 4. The small, medium, and big models are seperated with the red vertical lines.

The bigger models take much more time than the smaller models, while λ barely has an impact. The means that the limiting factor in the runtime of the greedy search is the length of the tabu walk.

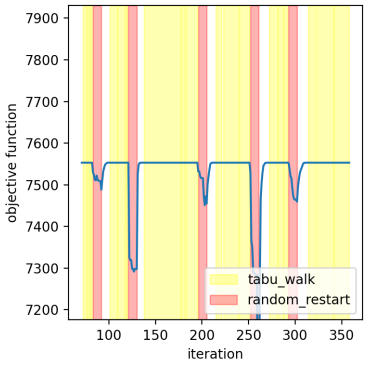


Figure 5: Score history after the first hill climb, big model with $\lambda = 0$

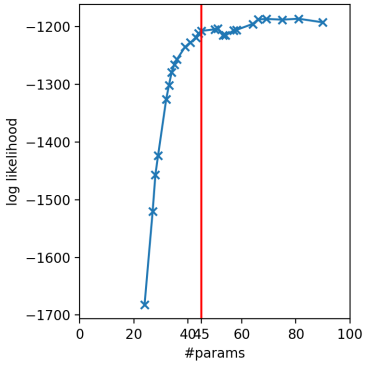


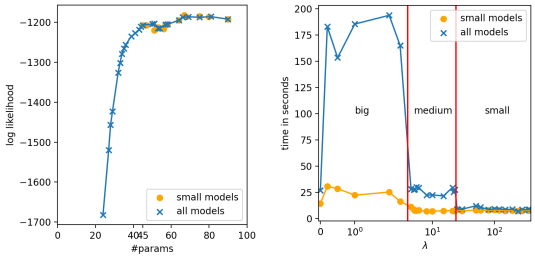
Figure 6: Likelihood of adjacency matrices with different *#params*

Also for $\lambda = 0$, the runtime is strangely much shorter compared to the other big models. The reason for this could be that a lot of possible changes improve the objective function by a very small amount after the first hill climb. This would result in the tabu walks that are significantly shorter, which reduces the runtime. A score history of the big model with λ is illustrated in Figure 5. The score history confirms that the tabu walks are significantly shorter than they could be.

3.4 Likelihood

The results of the likelihood measurements are illustrated in Figure 6.

The likelihood of the generated structures drastically increases up to $\#params = 45$. After that, there is a small dip in likelihood and then it slowly increases again until $\#params \approx 66$ and then barely changes anymore. Finally it reaches a likelihood that is not significantly better than at $\#params = 45$. The reason for the small dip could be a bad parameterization of the medium sized greedy search model (see subsubsection 2.6.1), even though it was fine-tuned the most out of the three models.



(a) Likelihood **(b) Runtime**

Figure 7: Comparison of different parameterizations

3.4.1 Submission Scores. The adjacency matrices i submitted to the submission website are competitive with the other top scoring adjacency matrices. On the sparse side ($\#params \leq 33$), my adjacency matrices even hold a small, yet significant and consistent advantage over the other top scores.

3.5 Are Longer Tabu Walks Worth It?

In Figure 7 the parameterization of models for medium sized and big adjacency models are compared to the parameterization of small adjacency matrices.

The shorter tabu walks of the small models lead to a huge improvement in runtime, and only lead to slightly worse likelihoods than medium models, and sometimes even slightly better likelihoods than the big models. This means that the medium and big models only get slightly better results at a huge runtime cost.

4 CONCLUSION

I made an efficient, well-performing greedy search algorithm with tabu walks and random restarts. By changing the regularization constant λ in the objective function, the sparsity of the generated adjacency matrices can be adapted up to a very fine degree. The greedy search performs especially well in the generation of sparse adjacency matrices.

With this greedy search, i found an adjacency matrix that is both sparse and has a high likelihood, with 45 parameters. It is possible to get higher likelihoods with more parameters, when the tabu walks are made longer. But this comes at a cost of high runtime.

Future work could include parallelizing the search for the best performing change, finding a way to make tabu walks more efficient, and finding an efficient way of checking if a flip introduces new cycles.

REFERENCES

[1] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47, 4 (nov 2009), 547–553. https://doi.org/10.1016/j.dss.2009.05.016

[2] John E Hopcroft, Jeffrey D Ullman, and Alfred Vaino Aho. 1983. *Data structures and algorithms*. Vol. 175. Addison-wesley Boston, MA, USA:.

[3] Marco Scutari, Catharina Elisabeth Graafland, and José Manuel Gutiérrez. 2019. Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning* 115 (dec 2019), 235–253. https://doi.org/10.1016/j.ijar.2019.10.003

A CALCULATIONS

A.1 Score Function

Here we derive

$$\max_G \log p(D|G,\hat{\theta}) = \max_G \sum_{i\in[n]} S_i(G)$$

$$\begin{aligned}
&\max_G \log p(D|G,\hat{\theta}) = \max_G \log\left[\prod_{x\in D} p(x|G,\hat{\theta})\right] \\
&= \max_G \log\left[\prod_{x\in D}\prod_{i\in[n]} p(x_i|x_{\mathrm{pa}(i)},\hat{\beta}_i,\hat{\beta}_i^*,\hat{\sigma}_i)\right] \\
&= \max_G \sum_{x\in D}\sum_{i\in[n]} \log p(x_i|x_{\mathrm{pa}(i)},\hat{\beta}_i,\hat{\beta}_i^*,\hat{\sigma}_i) \\
&= \max_G \sum_{x\in D}\sum_{i\in[n]} \log\left[\frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{1}{2}\left(\frac{x_i-(\hat{\beta}_i^\top x_{\mathrm{pa}(i)}+\hat{\beta}_i^*)}{\sigma_i}\right)^2\right)\right] \\
&= \max_G \sum_{x\in D}\sum_{i\in[n]} \left[-\frac{1}{2}\left(\frac{x_i-(\hat{\beta}_i^\top x_{\mathrm{pa}(i)}+\hat{\beta}_i^*)}{\sigma_i}\right)^2 - \log\hat{\sigma}_i - \frac{1}{2}\log 2\pi\right] \\
&= \max_G \sum_{i\in[n]} \underbrace{\left[-|D|\cdot\log\hat{\sigma}_i - \frac{1}{2}\sum_{x\in D}\left(\frac{x_i-(\hat{\beta}_i^\top x_{\mathrm{pa}(i)}+\hat{\beta}_i^*)}{\hat{\sigma}_i}\right)^2\right]}_{=S_i(G)} \qquad \square
\end{aligned}$$

Note that $\hat{\theta}$ ($\hat{\beta}_i, \hat{\sigma}_i$) depends on G ($\mathrm{pa}(i)$).

A.2 Graph Comparison

For a graph G' that was made by applying one elementary change to a graph G, it holds that

$$\sum_{i\in[n]} S_i(G') = \sum_{i\in[n]} S_i(G) + \Delta_S(G',G) \tag{1}$$

and

$$|E_{G'}| = |E_G| + \Delta_E(G',G) \tag{2}$$

Therefore

$$\begin{aligned}
&\sum_{i\in[n]} S_i(G_1) - \lambda\left|E_{G_1}\right| > \sum_{i\in[n]} S_i(G_2) - \lambda\left|E_{G_2}\right| \\
&\overset{(1),(2)}{\iff} \Delta_S(G_1,G) - \lambda\Delta_E(G_1,G) > \Delta_S(G_2,G) - \lambda\Delta_E(G_2,G) \\
&\iff \Delta_S(G_1,G) - \Delta_S(G_2,G) > \lambda\left(\Delta_E(G_1,G) - \Delta_E(G_2,G)\right)
\end{aligned}$$

B SPECIFICATIONS USED FOR PERFORMANCE ANALYSIS

The runtime was analyzed in Python 3.11.2 on the following hardware:

- CPU: Intel® Core™ i7-7700HQ (2.80GHz)
- GPU: NVIDIA® GeForce™ GTX 1060
- RAM: 16GB DDR4 (2400 MHz)