# Presentation

Hello everybody, it is a pleasure to meet you all here on this busy day for me. First of all, am I audible to everyone, also to the people online? (wait for thumbs up or something). Okay great. So this presentation is part of my application here at SIOUX MathWare, in which I will present a technical topic I have indulged myself in for the past months. My graduation project is about “structure learning for high-dimensional data”. Although I will only be graduating in approximately four months’ time, I feel confident enough to share some of our current findings with you. It is also a great practice run for my final presentation, so I hope you will enjoy it!

If you have short questions, please feel free to raise your hand. If you have longer questions, we can discuss those at the end of the presentation. Okay, so let us get started.

**Table of Contents**

So, what will we be discussing today. First, I will explain the problem setting of our the graduation project and sketch some motivating examples. Secondly, I will be discussing two methodologies that we have developed and compare them against the state of the art on some benchmark datasets. I will then show you some results to see how these methods compare against each other. Lastly, I will mention some directions that we will investigate in the coming months.

**Problem Setting**

Now, what problem are we trying to solve? Simply put, we are trying the learn the structure of a specific type of graphical model, purely based on observational data. This graphical model visualizes the interactions between the different variables. You can see variables here as nodes, or vertices, and edges quantify the strength of the directed relations between variables. So, an edge from variable *x* to variable *y* indicates that *x* is useful in prediction *y*. Lastly, we want to make sure that the inferred graph is acyclic, meaning that it contains no cycles or feedback loops, which is a common constraint in such graphical models.

**Example**

To see what is meant with such a graphical structure, let us consider a setting where we have five variables. These variables represent the season of the year, the rainfall, whether the sprinklers are on, the wetness of the pavement, and the slipperiness of the pavement. Suppose we are given observational data of these five variables, the goal is to infer the directed relations between these variables.

From real-life, we are already able to infer this graph structure. For example, we know that the season greatly influences the sprinkler’s uptime, as well as the rain, but not the other way around. Turning on the sprinklers does not change the season. Furthermore, both the rain and the sprinkler will make the pavement wet. Note that although the season influences the wetness of the pavement, we have already fully captured this relation in the rain and sprinkler variable. Lastly, the wetness of the pavement influences the slipperiness of the pavement. In the end, the corresponding graphical model with directed edges and no cycles will look like this. To infer such a graph from data, is the goal of this graduation project.

**Motivation – Root Cause Analysis**

Now, what is the use of learning such a graphical model? First and foremost, learning the structure of such a graphical model allows us to gain insights into the relationships and dependencies between components in a complex system. Suppose we have a complex system with nine interacting components, all represented by a node here. Now, suppose that the machine breaks down, and we are able to see that variables *c, f, and j* exhibit unusual behavior. Now, what is the most likely cause of the defect of this machine? We do not really know, right? It is most likely *c, f,* or *j* or all three, but completely breaking down the machine and inspecting all three components can be rather expensive or time consuming.

Now, suppose that we had used the observational data to infer the structure of this complex system, and suppose that this is the learnt structure. Now that we know this structure, what do you think the root cause of the malfunction of this system is? (Wait a bit). Indeed, the most likely root cause would be component *c* here, as this is the first variable in the chain that starts exhibiting strange behavior. This is a rather oversimplified example, but knowing such a structure can be extremely useful in these type of root cause analysis problems.

**Model**

Now, let us consider the graphical model that we will be discussing in this presentation. We will be discussion a Structural Equation Model, which can be expressed as follows. We have a vector *x* consisting of *p* variables. In the first example, we had a vector *x* that had five variables, representing, season, rain, etc. Now, all variables in *x* are a linear combination of the other variables. In the first example, we had that the wetness of the pavement could be seen as a linear combination of the rainfall and the sprinklers. Since we cannot expect to capture the behavior of all variables perfectly, we also model some residual error term epsilon here, which we assume to be a zero-mean Gaussian.

Now, given some observational data capital X, consisting of *n* samples, we are interested in inferring the coefficient matrix W, which can also be seen as the weighted adjacency matrix of the corresponding graph. The goal now is to find such a weighted adjacency matrix W that minimizes the mean squared error such that the graph induced by W is still a directed acyclic graph.

**Methodologies**

Now that we have explained the problem setting and the corresponding type of graphical model, let us explain some methodologies we have developed to solve this problem. The first method is a combinatorial method, where we need to search over the space of all possible orderings. The second method is a greedy method based on Orthogonal Matching Pursuit, where we will greedily update the graph edge by edge. Lastly, we have NO TEARS, which is a state of the art method that we use to compare our two methods against.

**Greedy Random Walk**

So, let us first take a look at order search. As mentioned before, the most difficult hurdle to overcome is to make sure that the matrix is acyclic. A useful decomposition for this is that when the matrix is acyclic, then there must exist an ordering of the variables such that all edges or coefficients only appear in an upper triangular matrix U. Given a specific ordering, we can easily find a suitable matrix W using Ordinary Least Squares or LASSO for example. In fact, estimating the coefficients is not difficult at all, it is determining which coefficients to estimate that is the most difficult problem here.

To see what I mean which such an ordering, suppose we have this adjacency matrix on four variables. An entry (I, j) is one if and only if there is an edge from I to j. Now, since this graph is acyclic, we can reorder the variables as follows, and when we rearrange the coefficients in W accordingly, we see that all coefficients are in the upper triangular part. Now, what we can do, is exhaustively try all possible orderings, estimate the corresponding upper triangular matrix U, and see which achieves the lowest loss.

Unfortunately, the number of orderings is factorial with respect to the number of variables, so we can never expect to try out all orderings within a reasonable amount of time. Therefore, we can try to do an efficient search through the space of orderings, such as a random walk. In such a random walk, we will randomly swap two variables in the ordering, and estimate the corresponding coefficients.

To improve things a bit, we can do a more guided search than a random walk. We can even be quite greedy, as in the Order Search algorithm which we know present here. We randomly swap two variables in the ordering, estimate the coefficients, and transition to this new ordering if and only if it achieves a lower loss. We keep on sampling from this greedy random walk until we have reached a certain time-out criterion.

To visualize this algorithm let us see a quick example on three dimensions. Suppose we start with the ordering 1, 2, 3. Then, we can only estimate the upper triangle, and estimate the loss of this estimate matrix. We can now randomly swap the second and third variable, and now we can estimate the variables in green, and see if this is an improvement. For this example, we suppose it is now, so we do not accept this matrix, and try another swap, 2, 1, 3. Estimating these coefficients yields an improvement, so we accept this matrix. Now, from this ordering, we can try 2, 3, 1, which supposedly does not give an improvement, so we try the ordering 3, 2, 1, which here gives an improvement, so we accept this matrix. For the sake of this example, we cannot find a swap that yields an improvement, so we stop and output these estimated coefficients.

So, that was how the first method worked, let us now consider the second method, which we call Orthogonal Matching Pursuit. For this method, we start with an empty matrix W. At each iteration, we will check which potential edge has the largest correlation with the current residual. If adding this edge creates a cycle, we exclude it, and if not, we add it to our matrix and re-estimate the residuals. We keep on doing this until we have again reached a stopping criterion.

Let us again see a simple example in three dimensions. We start with an empty graph, and all possible candidates are coloured in blue. Edges that will create a cycle are already colored red here. Now, suppose that edge (1, 2) has the largest correlation with the residual. Since this does not create a cycle, we add the estimated coefficient to the matrix, and re estimate the residuals. Then, edge (2, 1) is the best candidate, but since this will create a cycle, we exclude it. Our second best candidate then was edge (2, 3), who we can add since it does not create a cycle. Reestimating the residual then reveal edge (3, 2) as the best candidate, but since this creates a cycle, we exclude it. The second best candidate also creates a cycle, so the next best edge would then be edge (1, 3). Now that there are no edges to add or exclude, we terminate and this is the estimated matrix that we return.

Lastly, let us quickly mention NO TEARS, which is an interesting method that can be considered state of the art, to which we can compare our methods. Instead of explicitly enforcing G(W) to be a DAG, they developed a smooth continuous function h that is equal to zero if and only if we have a DAG. So, they focused on solving this equality constrained program on the right.

So these were the three methods I wanted to discuss with you. Now, let us see how they compare against each other on experimental data. We randomly generate adjacency matrices with a prespecified number of edges. Each edge will get a weight uniform between plus and minus 0.5 until 2.0. We then use this matrix to sample data where we add standard Gaussian noise, generate a data matrix X consisting of one thousand of these samples.

Now, we try for varying values for *p*, and for each *p,* we redo the experiment ten times to get reliable estimates. We compare the three methods based on a structural performance metric, the structural hamming distance, and a predictive performance metric, the expected excess risk.

So, let us first consider the structural hamming distance, which represents the number of edge additions, deletions, and reversals between the original and the true graph. The lower this value is, the better.

For the setting where we have fewer than fifty variables, the Greedy Random Walk approach seems to be outperforming the state of the art NO TEARS. For Orthogonal Matching Pursuit, however, the greediness of the method seems to be working well when the number of dimensions is quite large.

Now, let us consider the expected excess risk, which quantifies how well our matrix W would predict in the population setting, compared to the true data generating matrix W. Again, we see that the Greedy Random Walk performs best, and NO TEARS seems to be also performing well for small dimensions. However, when we have 40 or more variables, Orthogonal Matching Pursuit seems to be the better choice than NO TEARS, which is an interesting result.

So, to wrap up, what have we seen in this presentation? I have discussed with you the problem of learning the structure of a graphical model in high-dimensional data. We have seen three methods, the Greedy Random Walk, Orthogonal Matching Pursuit, and NO TEARS, and we see that our two relatively simple methods are competitive with the state of the art. The Greedy Random Walk seems to perform extremely well, and we expect it to be the most competitive in low dimensions, where there are relatively fewer orderings. Orthogonal Matching Pursuit also performed well, but was mostly competitive in the high-dimensional setting, and as a sidenode, it is also much faster than NO TEARS, so that is a big plus as well.

Now, what will we be working on the coming months? We will also be looking at different types of graphical models, for example for dependent time series data. Examples are Vector AutoRegressive models to model time series data. Furthermore, we will also be investigating how to apply regularization to the aforementioned methods. We also want our model to generalize well, and therefore a method to prevent “overfitting” should also be investigated. Lastly, we also hope to give some statistical guarantees for the performance of our methods, for example that the recovered solution is statistically close to the true solution.

So, that was my presentation! I thank you all for listening, do you have any questions?